Under consideration for publication in Math. Struct. in Comp. Science

Category Theory Foundation For Engineering Modelling

Henson Graves,

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

E-mail: Henson.graves@hotmail.com

Received May 2013

Category theory provides a formal foundation for engineering modelling, as well as, mathematics and science. Both structure and behaviour, as they occur in engineering models for manufactured products and biomedicine, can be embedded as axiom sets within a mathematical formalism, called Algos. The Algos language is a two sorted first order Horn clause theory based on topos language constructions. An Algos theory, generated by Horn clause axioms is an elementary topos. The Horn clause formalism lends itself to automated reasoning. Algos has both a linear syntax and a graphical syntax based on the engineering modelling language SysML. The use of Algos for axiom development is illustrated with axioms for two classes of engineering models, one called Structure Descriptions and the other called Composite Structure models. An example of a Structure Description is the class of 2-amino acids. The problem exhibits common issues of constraining realizations of descriptions to have a specific graph theoretic structure. Algos contains the language of a Description Logic and generalizes several formalisms which have been used for modelling structure descriptions. Composite Structure models represent systems which have behaviour, as well as component structure. Following the topos lead, the terminal object of these models have a time structure. State machines, as well as equations representing physical laws, can be represented and are used to axiomatize these models. An example of a vehicle test system illustrates how behaviour is represented. A description of the formalism including soundness and decidability results for restricted axiom sets is presented, together with comparisons to other logic based formalisms.

Contents

1. Introduction

This paper describes how engineering models, as they are constructed for manufactured products and biomedicine, can be embedded as axiom sets within a logic-based formalism, called Algos (Graves and Blaine 1985; Graves and Blaine 1986). By embedding engineering models as axiom sets automated reasoning can be used to solve everyday engineering problems. In this sense the formalism, Algos, provides a formal foundation for engineering modeling. Algos is based on elementary topos theory and follows in the path of topos foundations for mathematics and physics. Algos has a graphical syntax based on the engineering modeling language, SysML, as well as a linear syntax. Algos has been implemented as a computer based reasoning system. Examples of axiomatic descriptions in Algos have been given in (Graves and Bijan 2011; Graves 2012). Algos, with its validation by axiomatizing a variety of engineering models, offers a practical approach to developing logic-based formalisms for engineering, as well as mathematics and science.

Embedding an engineering model as an axiom set provides the means to integrate automated reasoning with product development and analysis. Reasoning occurs throughout a system lifecycle. In the design process, before a product is built, care must be taken that design modifications do not lead to inconsistent designs; in verifying that a product meets its requirements it may not be possible or feasible to verify all requirements by test; in product maintenance and medical diagnostics one is attempting to infer the cause of a fault from symptoms. These situations all require a precise use of inference based on engineering models. Much of the reasoning and analysis from engineering models is informal and manual. As a result the analysis is error prone and inefficient due to the complexity of the models.

When axioms in a logic are used to describe an application domain, the theory of the domain is the collection of statements derivable from the axioms by the inference mechanism of the logic. Application domains may be broad, such as a domain of physical laws, or narrow, such as molecules with a specific structure. There are a wide variety of logic-based formalisms candidates. A formalism provides the specifics for a language of terms, formulae, and inference rules. Embedding an engineering model as an axiom set within a logic-based formalism offers the potential for precise efficient ways of solving engineering problems. The problem is to find a suitable formalism.

Perhaps the best known logic-based formalisms are subsets of first order logic which only use variables of a single type. Patrick Suppes calls these formalisms Standard Formalisms (Suppes 2002). Suppes makes the argument that giving axioms in a Standard Formalism for most domains in the empirical sciences is not possible (Suppes 2002) page 27. His argument is based on the fact that one needs to include mathematics and physics. At a more abstract level the argument is that an empirical science theory uses constructions that are higher order in a standard formalism.

Logic Programming and Description Logic (Baader *et al.* 2007) formalisms, when viewed as a fragment of first order logic, are standard formalisms. Their advantage for use with application domains is that reasoning is not only computationally tractable, but in many cases the consistency of the theory generated by an axiom set is decidable. Description Logic and the Logic Programming formalisms have been employed for modelling in engineering, human anatomy, and molecular biology (Motik *et al.* 2008; Magka *et al. 2012*; Hastings *et al.* 2010). Finding an axiomatic description in these formalisms which sufficiently constrain the possible interpretations has proven difficult (Magka *et al. 2012*). This difficulty is consistent with Suppes's argument that axioms in a Standard Formalism are not possible for many applications.

Suppes argues for the use of set theory for empirical foundations to overcome the limitations of a Standard Formalism. However, set theory is not the only alternative. There there are mathematical formalisms (Lambek and Scott 1980; Bell 1986; Graves and Blaine 1985; Graves and Bijan 2011) which generalize set theory and have language constructions familiar from set theory, but are more algebraic in form. In these formalisms axiomatic properties of directed graphs which represent class and object models in programming and modelling languages can be expressed as first order statements. Further, these formalisms can provide an axiomatic representation for behaviour and can incorporate physics. In Standard Formalisms some of these properties cannot be expressed as they would be higher order.

William Lawvere (Lawvere 1964) advocates use of category theory to serve as not only a foundation for mathematics, but physics. The first order axiomatization of topos theory by Lawvere (Lawvere 1964) is a notable example of an alternative to set theory. An elementary topos is a first order theory, but not a Standard Formalism as it uses variables of two kinds, maps and types (objects), term constructions familiar from set theory, and axioms for these language constructions. A considerable amount of physics has been worked out in the topos context.

If a Standard Formalism is extended to mean a first order logic formalism with a type system then Extended Standard Formalisms are also candidates for axiomatic descriptions in the empirical sciences. A type system is a collection of term and type constructions with axioms for its term constructions. The theory of an application domain in an Extended Standard Formalism is the theory generated by the application axioms and the axioms for the term constructions. Elementary topos theory is an Extended Standard Formalism. In elementary topos theory the objects in the language are the types of the type system. The type constructions include Cartesian product of types and the power type construction. Other type constructions, such as sum and exponential are definable. [†]

While an elementary topos is a first order theory, it is not directly amenable to implementation as a computer based reasoning system. The elementary topos axioms include the axioms for a typed lambda calculus which are known to be implementable as a computation and reasoning system. However, the elementary topos axioms also include axioms for subobject classification which postulates an isomorphism between certain types, but does not provide a construction for the isomorphism. However, with a slightly stronger axiom, known to be satisfied in many topos examples, a canonical construction for the isomorphism can be given which makes these systems amenable to implementation (See Section 3).

[†] In addition to the Extended Standard Formalisms, type theories have also been suggested as foundation formalisms for mathematics and for software. A type theory is a type system with an inference mechanism. Generally type theory deductions are represented as entailment from axioms and inference rules. Type theories are closely related to topos theories. This relationship is discussed in (Lambek and Scott 1980).

1.1. *Engineering Modelling*

Engineering modeling as it is practiced for manufactured products and biomedicine is in need of a logic-based semantic formalism. Engineers have always built models for their systems of interest. Engineering models are used to specify or describe systems and their interaction with their environment. The model may describe a unique system such as an oil refinery, but the models are often used to describe a class of systems such as cell phones which satisfy a specific design model. In both cases engineering models are used to analyse and reason about the systems that they describe. The size and complexity of models leads to the need to reason within and about these models as a way to understand the systems. However, valid reasoning rests on the formal semantics being in accord with the informal semantics. The valid use of automated reasoning rests on embedding the models within a sound logic-based formalism.

The idea of using axiom sets to describe systems of interest to engineers has been around since the mid 1960s, but has not gained much traction. Axiom sets are difficult to construct and do not always capture the class of intended interpretations correctly. However, with the advent of engineering modelling languages in the UML family such as SysML (SysML 2010) the situation has changed. Attempts to use automated reasoning in the context of engineering models has generated interest in converting or embedding engineering models into axiom sets within a logic-based formalism. The axioms are intended to describe the same thing as the model, but more precisely, and provide justification for integration with automated reasoning. Embedding a model as an axiom set often surfaces implicit assumptions made by the modellers.

While science develops broad physics-based theories engineering specific domains generally develop circumscribed theories for a specific class of systems or manufactured products. From the formalization viewpoint this amounts to adding application axioms for the specific theory to any general theory needed, for physics assumptions. Engineering models in a formal language such as SysML (SysML 2010) have term constructions which are a type system. These language constructions are used to represent the systems of interest in terms of their component decomposition and connections between components. For some applications, considerable physics is needed while for other applications no physics may be needed. The molecular descriptions considered here are sufficiently abstract that no physics needs to be included. When these descriptions are expanded to include material properties and dynamics physical laws are needed. Engineering modelling confronts these issues on a day-to-day basis.

Two examples of engineering modelling are used for illustration here. One case, common to manufactured products, human anatomy, and molecular biology, is how to represent an axiomatic description for a class of structures each of which conforms to a specific graph theoretic pattern of components and interconnections between components. A realization of a description is a graph structure which conforms to, or satisfies the description. Given the complexity of many applications manual analysis of a structure description is time consuming and error prone, if it is even possible. When analysing a specific structure it may not be possible to take it apart; one must rely on prior knowledge that it conforms to a known description. This problem has resisted attempts to give an axiomatization which can be used in the context of automated reasoning (Magka *et al. 2012*).

The other case illustrated here is a model that represents the behaviour of a vehicle system operating within its environment. This example uses the concepts of Structure Descriptions but extends them. The behavior of the vehicle is a composite of the behaviour of its subsystems, as mitigated by the physical laws of the operating environment. The solution to this kind of behavioral modeling build on and extends the structural descriptions encountered in the static structural models of biomedicine and manufactured products. Modeling languages provide language constructions such as state machines which are effective for constructing complex models, but they lack a formalized semantics. Finding a formalized semantics for behavioural constructions is currently a topic of interest to the Object Management Group (OMG) which maintains standards for a number of modelling languages. The Algos solution, following topos theory, employed for behaviour representation, uses axioms which imply that the terminal object has a space-time structure.

By axiomatizing a description in a logical formalism, with or without a type system, one has the potential to use automated reasoning to answer questions about the structures described. For example one might want to know if every structure satisfying a molecule description contains a carbon ring. For a manufactured product one might want to know what components are connected to the electrical system. Many engineering problems translate directly into whether the theory of the axiom set is consistent (Graves 2012). In product design the addition of a mechanical device component to a design model may render the design inconsistent, as the component may violate constraints such as the total amount of power that the electrical system can supply.

When an axiom set in a logical formalism is used as a description for a class of structures the logical formalism supplies a precise definition of what constitutes a valid interpretation (model) of the axiom set. Reasoning from an axiom set applies to all valid interpretations of the axioms. Theoretically, the problem is to find a logical formalism to represent structural descriptions as axiom sets in which the axioms can constrain the models sufficiently. Practically the problem for converting an informal description to an axiom set is to avoid under or over specifying the models one is attempting to describe.

When using any automated inference procedure to reason about valid interpretations (models in the logicians sense) of the axioms, one is concerned that the inference procedure is sound. Are the conclusions derived from the axioms satisfied in the structures being described? In a Standard Formalism, with the absence of term constructions, soundness is not generally a problem, as commonly employed inference procedures are known to be sound. The reasoning in a logic-based formalism may be sound, yet an application axiom set may be inconsistent or contradictory. Detecting that an axiom set is inconsistent is a primary application of reasoning in engineering, as many practical engineering problems are equivalent to the consistency of a theory.

However, with an Extended Standard Formalism which contains term constructions soundness becomes an issue. Can the term constructions be interpreted in the application domain and are they non-contradictory? The approach in logic has been to show that the consistency of a mathematical formalism is equivalent to the consistency of set theory. For a rich term language there may be real questions of whether there are mathematical formalisms, in which the term constructions can coexist, without contradiction. For example, Bertrand Russell used a term construction in his type theory which assigns to a formulae of the form $\forall x.\exists y R(x, y)$ an operator τ_P which satisfies the formula. The operator τ_P is called a description operator. At that time the validity of the description operator axiom was considered problematic. As the description operator was needed this had a negative impact on the acceptance of his type theory. The description operator can be defined within a topos or Algos theory. This operator is used to replace a functional relation with a map, as is done in set theory.

When an Extended Standard Formalism is used for application domains then in addition to logical soundness there is a question of physical soundness. The term and type constructions have to be interpreted in the physical world and the term construction axioms have to be satisfied. The ability to interpret the terms and axioms of an Extended Standard Formalism in the application domain can be viewed as a question of physical soundness. If the constructions do not reflect the reality then the type system is inappropriate for the application. When interpreting an axiom set of an Extended Standard Formalism in an application domain one is concerned that the term constructions have a well defined interpretation in the application domain, that the description axioms do not violate the assumed knowledge about the domain, and that the description is sufficiently precise for application usage. While these kinds of questions have not been in the forefront of logic research, they are critical questions for science and engineering modelling. [‡]

When choosing an Extended Standard Formalism for applications the considerations start with formalisms that are well developed and debugged, such as set theory, in which the application mathematics can be done. The considerations also include what specific choice of language primitives are suitable for physical interpretation. In addition, what language works practically for modelling in science and engineering. Further, to use automated reasoning the formalism needs to be as directly implementable as possible. While a complete discussion of these topics is beyond the scope of this paper, the design of Algos meets these criteria.

1.2. *Algos*

Algos (Graves and Blaine 1985) is an Extended Standard Formalism based on topos theory (Graves and Blaine 1986). Topos theory uses types for Cartesian product, exponential (function), and Power types, with corresponding map constructions. Other language constructions are definable. Topos theory uses two kinds of terms, maps and types. Each map has a domain and a range type. A map is analogous to a function in set theory, but is a more general concept. For example, the paths of arrows in a directed graph satisfy the properties of a category (Lambek and Scott 1980) where the nodes are types and the

[‡] One well known example is the interpretation of attributes that represent measurable quantities of a system. Engineering models which represent physical quantities such as weight or distance have to prescribe the units of measure to be applied in the interpretation for the interpretation to be precise.

maps are paths of arrows. The characteristic property of a map f is that it assigns to an individual a in its domain a value $f(a)$ which may also be written as $a.f$ in its range. There is no presumption how this assignment is made. The Algos primitives, following topos theory, have been validated first for algorithms (Graves and Blaine 1986) and more recently in engineering applications (Graves and Bijan 2011; Graves 2012).

The Algos term language uses two sorts (Zarba 2007), maps and types (objects for category theorists). The abstraction of sets and functions in set theory to types and maps in category theory places a different burden upon physical interpretation. For example, manufactured products may be described in terms of types of component with maps used for describing component decomposition and connections between components. The physical interpretation of a type is a recognition procedure for instances of the type. In many applications the procedure to recognize instances measures attribute values of an instance to determine how to classify it. The physical interpretation of a map is often a procedure to recognize that a component serves a particular role, or recognize a particular connection between components. This is the kind of physical interpretation used on an everyday basis. For example, in servicing manufactured products the physical product is compared to a design specification for malfunction diagnosis. An Algos theory contains a natural numbers type N which satisfies the Lawvere axioms (Lawvere 1964). This type is not used in the molecular examples beyond its implicit use to enable the definition of integer cardinalities. However, the type N enables abstract data types to be defined in terms of the Algos term constructions. Abstract data types are used as the range types of attributes defined for application types such as a hydrogen atom.

The map and type terms can be used to represent individuals, classes, and relations as terms in the language. The term language uses a number of topos language constructions such as images of maps, as well as a form of the description operator. The term language includes Description Logic concept (class) and role (relation) constructions as type constructions. The axioms for the map and type term constructions and application axioms are Horn clauses written as universally quantified logical implications in literals of atomic formulae. The inference rules, while satisfied by first order inference systems, correspond to type theory inference rules for entailment. As a result Algos is computationally tractable and has been implemented as a computational and reasoning system (Graves and Blaine 1985). The physical interpretation of Algos theories fits well with engineering modelling practice.

An Algos theory, generated by application axioms and the language construction axioms, satisfy the axioms for an elementary topos. In addition to the logic in which the term construction axioms and application axioms are expressed (the external logic), an internal logic is defined within the term language. An internal formula is a map whose range type is a logic type, Ω. The internal logic of an Algos or topos theory is a higher order logic, which without other assumptions, satisfies intuitionist rules of deduction. The Algos term construction axioms are derived from elementary topos axioms by use of Skolemization to eliminate the existential quantifiers of the topos axioms. The resulting topos generated by an Algos theory is a topos with canonical subobjects (Lambek and Scott 1980). The mathematical soundness of topos theory is accepted. Reasoning in an Algos theory is sound in that formulae derived from an axiom set are true in any structure

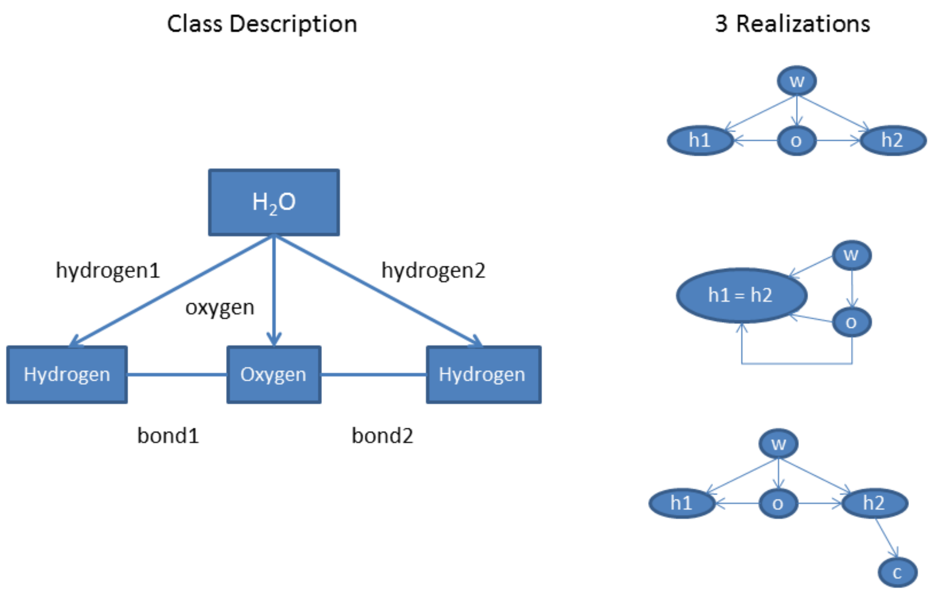

Fig. 1. Water

which models the axioms. The fact that an Algos theory generates a topos enables use of constructions such as the description operator without concern for logical soundness.

Restrictions on axiom sets are introduced which yield decidability results which apply to the molecular biology use cases. The axiom sets used for the structure descriptions such as amino acids are a very restricted class of Algos axioms. The language constructions with their axioms provide an axiomatic semantics for a generalized Description Logic. The subtypes of any type in Algos have the properties of a Description Logic. The symbols in the water axiomatization are identified as classes, in that they are subtypes of a universal type symbol, $Thing$. $Thing$ can be taken to be molecules. The translation of informal structure descriptions into a language using the topos type constructions serves as evidence that these constructions not only make sense, but are needed for applications. The language can represent composite structures which have states and actions to change states. Algos, following topos theory, can be used to express and reason about properties of objects that vary in time.

1.2.1. *Physical Interpretation* To use a mathematical formalism for an application domain requires describing how the terms in the language are to be interpreted in the domain. The water molecule provides an example of how Algos terms are used to describe structure. A more detailed and complete axiomatic description is worked out in Section 2. Each water molecule has three atoms, one oxygen and two hydrogen atoms which are bonded appropriately. A description of a class of water molecules may use a type for H_2O and types for the atoms, $Oxygen$ and $Hydrogen$. Conceptually the physical interpretation of a type is a recognition procedure which can recognize if an individual is a water molecule or a particular kind of atom. The recognition procedure for H_2O is that the molecule has the specified components and connections. The H_2O molecule has three maps which assign a respective atom to each water molecule. For example a map $Oxygen$ assigns an oxygen atom to each water molecule. This map has domain H_2O and range $Oxygen$ which is written as

$$oxygen : H_2O \to O. \tag{1}$$

The interpretation of the map $oxygen$ is again a procedure which can recognize which oxygen atom has been assigned to a specific water molecule. This interpretation does not make the physical assumption that the domain of the application contains a set of water molecules. Nor are the maps assumed to be set theoretic functions whose domains are water molecules and whose ranges are the sets of atoms. A realization of the description of a water molecule consists of four individuals, a water molecule, one oxygen atom and two hydrogen atoms.

Figure 1 is a graphical illustration of the water molecule description and possible realizations. These diagrams are used to illustrate some of the language concepts. In Figure 1 the nodes in the graph on the left are types of molecules rather than individual molecules. In the terminology of object oriented programming the diagram is a class diagram rather than an object diagram (Kuske 2009). The right side of Figure 1 shows three possible realizations of the left side description. The diagrams on the right side are object diagrams. The nodes are individuals. The top diagram on the right is the desired realization.

In the Algos representation of the left side diagram the symbols H_2O, $Oxygen$,and $Hydrogen$ are types and the symbols $hydrogen1$, $hydrogen2$, $oxygen$, $bond1$, and $bond2$ are maps. The use of maps in the axioms for water enable a simple way of ensuring that the oxygen atom of a water molecule is bonded to a hydrogen atom of the same water molecule. The formula

$$x \in Water \Rightarrow x.oxygen.bond1 = x.hydrogen1. \tag{2}$$

says that the oxygen bond of a molecule x is bonded to the hydrogen atom of the same water molecule. The membership predicate $\in$ is justified as the types are subtypes of a universal type, $Thing$. In this formula composition is written from left to right with a dot notation rather than the more usual applicative notation. Thus $oxygen.bond1$ is the composition of $bond1$ with $oxygen$.

A relation such as $hasPart$ can be defined to be a subtype of the product type $(Thing, Thing)$. Formula (2) is a Horn clause Algos formula where X is a type variable. As the variable is a subtype of $Thing$ one can call it a class variable. The expression $x : Thing$ is used for an individual and $x \in Oxygen$ says that x is an oxygen atom. Similarly the expression $< x, y >$: $hasPart$ states that $< x, y >$ is an instance of $hasPart$. These concepts are made precise in Section 3.

For example the map $hydrogen1$ has domain H_2O and range $Hydrogen$. The map assigns for any water molecule $w : H_2O$ an individual $w.hydrogen1$ which is a hydrogen atom. In Algos functional relations and maps are in one-one correspondence. For a map such as $hydrogen1$ the notation $|hydrogen1|$ is used for the functional relation defined by $hydrogen1$. When representing a non-functional relation with a multiplicity k then k maps are introduced.

One can add individual constants $w, h1, h2, o$ to the axiom set with the equations such as $w.hydrogen = h1$ which relate these constants to the values of the part maps evaluated for w. With these equations the top graph on the right is a realization of the class diagram on the left in that the four nodes are individuals of the specified classes. The arrows on the right represent ordered pairs. For example, the pair $< w, h1 >$ is a member of the functional relation $|hydrogen1|$ which is called the graph of the map $hydrogen1$. Similarly the other arrows represent ordered pairs which are members of the appropriate functional relations. In the second diagram only one hydrogen atom is used. In the third one an extra carbon atom is attached to a hydrogen atom. Additional axioms beyond those introduced are needed to conclude that all realizations have the expected configuration for a water molecule. The complete axiom set for water and the 2-amino acid axiom set in section 2 are special cases of axiom systems for which decidability of consistency is decidable (Section 4). Also all of the valid realizations are structurally isomorphic.

The example of axiomatizing a vehicle test model makes use of the full Algos language constructions. The structural descriptions used for molecules are used here for component decompositions. However, for the vehicle model the values of maps used to describe a component may vary in time as the vehicle is tested. A class of axiom sets, called Composite Structure Models is singled out as this class encompasses many engineering models. By using an axiom that the terminal object has a time structure one is able to represent both state machines and physical which are employed in engineering models which represent not only a product but its operation within its operating environment. Simulations of the engineering models which are critical for engineering analysis become valid interpretations of the Algos axiom sets. For the class of engineering models considered the semantics can be identified with a version of a Labelled Transition System such as found in (Knight *et al.* 2012). Considerably more work needs to be done in this area.

1.2.2. *Practical Consequences* The application modelling use cases for science and engineering generally include a directed graph as part of the description. These graphs are included in the signature of the resulting Algos application axiom set. The graph in Figure 1 has a hierarchical decomposition and the axioms express constraints on how components are connected. Application domains generally have domain specific graphical modeling tools to represent structure descriptions. Each domain uses implicit assumptions which may not be represented within the modeling tool. However, in some domains such as manufactured products, engineering modeling tools provide checking of syntactic correctness of a model. Directly representing graphical structures as axioms has the advantage of providing a user friendly interface for development and communication of descriptions. Graphical authoring tools can be interfaced with automated reasoning tools to provide semantic as well as syntactic analysis of the axioms. Fortunately the syntax of Algos and SysML are very similar with considerable overlap. The result is that SysML with its graphics based syntax can be used to develop complex Algos axiom sets which would be almost impossible without a graphics syntax. The language constructions and modelling principles are general and have been applied to a variety of structural description applications (Graves and Bijan 2011; Graves 2012). The extension of these concepts for behavior provides not only a foundation for reasoning but a foundation in which simulation becomes applied model theory, in the logician's sense of model theory.

1.2.3. *Theoretical Consequences* A slight specialization of the elementary topos axioms enable the multi-sorted Horn clause presentation to be easily converted into a reasoning and computation system. The computational techniques extend the rewriting techniques used for typed lambda calculus and Cartesian closed categories. The specialization of Algos axiom sets first to Description Axiom Sets and then engineering Composite Structure Model Axiom Sets enable correspondences between the external Horn clause axioms and the internal formulas which do not hold in general. In the case of Description Axiom Sets the external logic can be reformulated to use the sorts for individuals, classes and binary relations. This can be used to characterize the (restricted) theories generated by the Description Axiom sets. In the case of Composite Model Axiom Sets the external logic can be restricted in a different way to what amounts to state variables for the theory. This leads to a topos model theory for this class of modules which can be identified with a form of path semantics within the state space. This restriction leads to the use of Labelled Transition Structures and bisimulation relations on the state space.

1.3. *Structure of the document*

Section 2 presents the 2-amino acid example before the presentation of the Algos formalism. The rationale is to illustrate that the language constructions used are very intuitive, and can be used without a full understanding of their axiomatics. This section develops an Algos axiom set for the class of 2-amino acids starting from the diagram in Figure 2 and the accompanying text. The axiom set can be used to determine whether a specific configuration graph of atoms and bonding relationships is an 2-amino acid and whether a molecule has specific substructure of components and connections such as a carbon ring. In the axiom set nodes are types which are defined to be classes and edges are maps. Figure 3 uses the syntax of the modelling language SysML (SysML 2010). SysML does not have all of the constructions used in Algos to express the amino acid axioms, but these constructions could be added to SysML.

Section 3 describes the Algos formalism. The Algos languages uses two sorts map and type with a collection of map and type constructions. These constructions use first order Horn clauses to express for the language constructions. [§] These axioms are called the Algos axioms. They are derived from topos theory (Lambek and Scott 1980). Axioms for an application description are also Horn clauses in the Algos language. The application axioms together with the Algos axioms generate an Algos theory. The Algos term

[§] The axioms all have the form $p_1 \wedge \ldots \wedge p_n \Rightarrow p_{n+1}$ where the p_i are literals which contain variables for maps and types.

construction axioms include the axioms for a category. These axioms are a simple generalization axioms for a directed graph (Lambek and Scott 1980) where types are nodes and maps are edges. Fragments of SysML have been embedded into Algos (Graves and Bijan 2011). As the Algos axioms imply that for any functional relation there is a corresponding map which has the functional relation as its graph. The logical soundness of Algos follows as any Algos axiom set may be faithfully embedded within a topos, called the syntactic topos of the axiom set.

Section 4 singles out a class of axioms sets which are sufficient for the 2-amino acid and water descriptions, but do not use the full expressiveness of Algos. The class of Algos axiom sets are embedded within a theory which contains a distinguished type constant, $Thing$. No special assumptions are made about $Thing$. However, class symbols introduced in the signature of an axiom set are embedded as subtypes of $Thing$. Application axioms are restricted to Horn clauses which use individual, map, class, and relation variables. The theory generated by a structure description axiom set is the resolution closure of the application axioms together and the Algos axioms. The Algos Description Logic has the Algos axiomatic semantics. The description theories enable both the representation of class and object diagrams (Kuske 2009) in the same language. The model theory for these axiom sets is a generalization of Description Logic model theory in that the classes are mapped by an interpretation to subtypes of a domain Δ. Comparisons of Algos to other approaches including Description Logic (Baader *et al.* 2007) extensions (Motik *et al.* 2008; Magka *et al. 2012*) are given.

Decidability results for restricted classes of axiom sets which include the amino acid axiom set. In the use case examples the maps in the signature of the axiom set divide into two classes, part maps and connection maps. The restrictions on Algos axiom sets which ensure decidability of consistency are based on an acyclic condition for part maps. The connection axioms define map equations each of which can be represented by a unary predicate. These conditions enables the restricted axiom sets to be represented as monadic Ackermann formulae which is known to be a decidable class (Ackermann 1954).

Section 5 outlines the Algos approach for engineering modelling system behaviour. Behavioural modelling in Algos follows topos theory in making use of axioms which imply the terminal type has a space-time structure. language constructions in SysML provide component modelling constructions provide a good basis for modelling composite behaviour. The Algos approach follows engineering modelling practice closely. Engineering modelling language with their graphical syntax can be used for axiom development in the Algos context. The topos framework enables the integration of dynamic systems with structural decomposition.

Section 6 gives background on the engineering of topos axioms to produce Algos axioms. Algos is a computational logic formalism in the sense that the Algos axioms and the application axiom sets are all Horn clauses which are readily implemented within a theorem proving computational system. Properties of directed graphs such as the graph having a root can be expressed in first order Algos axioms where their expression in a single sorted Logic Programming framework would be higher order.

2. Amino Acids

This section constructs an axiom set that represents the structural aspects of a class of amino acids. The axiom set is simply a more precise version of a model such as an engineer might construct. Even though the amino acid example is simple it illustrates the utility of the foundational topos language constructions for everyday modelling. These language constructions are everyday modelling concepts. An informal discussion of their semantics is given when the langauge constructions are introduced. The discussion of their use is

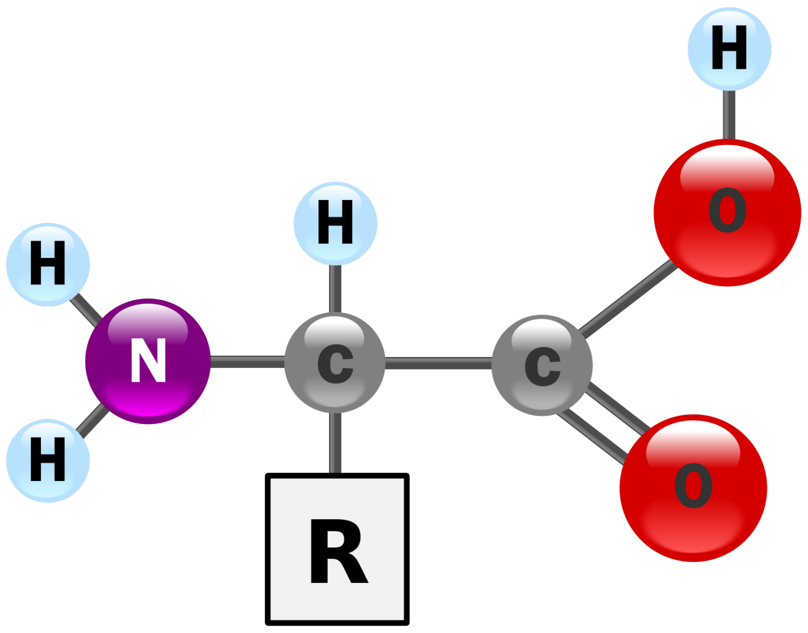

Fig. 2. Amino Acid

intended to be readable without knowledge of the specifics of the Algos formalism. Each of these language constructions will be footnoted when they are introduced to reference their formal semantics in Section 3.

The axioms are represented using in part the graphical systax of SysML. It is not presumed that the reader is familiar with SysML. However, additional language constructions beyond SysML are needed to constrain the interpretations of the axioms so that they describe only the intended structure. The topos constructions have direct physical interpretations. Their use provides evidence for using an Extended Standard Formalism and illustrates how axiomatic descriptions can build on informal graphical description conventions, convert the informal descriptions to the modelling language SysML, and add axioms to constrain the realizations to those with the intended structure. A formal presentation of the language constructions and their semantics is in Section 3. Proofs of the properties of the interpretations of this axiom set are given in Section 4. Their axioms provide a semantics for the corresponding SysML language constructions. The same language constructions work for manufactured products.

While the example was chosen for its simplicity, amino acids illustrate many of the issues confronting giving axiomatic descriptions for a class of structures. An axiomatic description is developed for the subclass of 2-amino acids. The structure of a molecule is its component decomposition and the connections between components. The description will not cover attributes of molecules such as atomic weight. The axioms do not cover the measurable attributes of the molecules and to not represent the dynamic aspects of these molecules. However, this additional information can be represented in SysML and axiomatized within Algos.

The description of the 2-amino acid class contains a place holder (variable). In molecular biology these place holders are called substitutients. As a result the 2-amino acid molecules satisfying this axiom set may have multiple structures as the place holder may be filled with molecules provided they satisfy the place holder conditions. In the Algos formalism the place holder is represented as a variable. The description admits variant realizations by replacing the substitutient with a molecule description with no place holers. Such applications are common in manufactured products where, for example, a vehicle model may have many variants which have different equipment such as different engine choices. Then the place holder(variable) is replaced a concrete molecule description and becomes a template. All of its realizations are structurally the same.

The amino acid class description starts with Figure 1 and a textual description taken from Wikipedia. The next step is to construct an engineering model (Figure 2) of the description using the syntax of SysML. Figure 2 contains two diagrams, the one on the left corresponds to the textual description; the one on the right corresponds to Figure 1. An informal description of the semantics of this model is given. After this discussion it becomes clear that additional axioms beyond what is expressed in the SysML model are needed to capture the intent of the model description.

2.1. *The engineering model*

The Wikipedia text describes the hierarchical decomposition for the 2-amino acid molecules. Figure 1 describes bonding relations which occur between atoms of an amino acid molecule. This informal description is represented by an engineering model in Figure 3 which contains two diagrams. The diagram on the left labelled BDD. BDD stands for Block Definition Diagram which is the SysML name for this kind of diagram. The BDD describes the hierarchical decomposition. The diagram on the right, labelled the IBD for Internal Block Diagram, describes the bonding relations between atoms and is similar to Figure 2.

The amino acid axiom set is intended to describe the class of 2-amino acids. An individual amino acid is to have the graphical structure of Figure 1. The intent of Figure 3 with its two diagrams is to describe the class of amino acid molecules rather than an individual molecule. To achieve this intent the model has additional axioms beyond what are expressed graphically in Figure 3. The axioms are added to represent implicit assumptions in this domain. To describe the class of molecules the SysML model uses

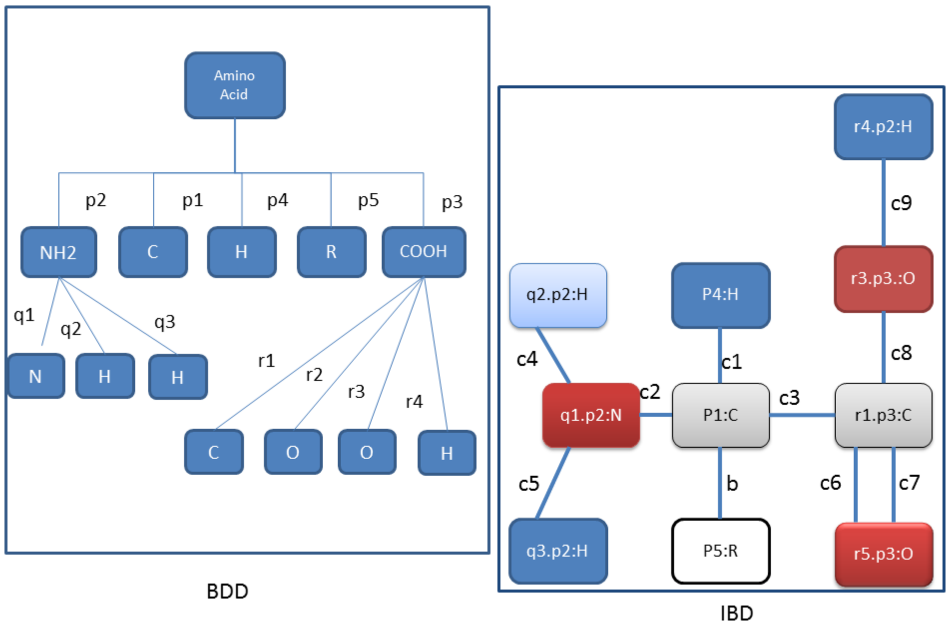

Fig. 3. Amino Acid SysML Model

directed graphs of nodes and arrows. Nodes represent the types of the molecule and its components and arrows represent, in general, relations between the nodes. For example component relationships and bonding relations. In this case the arrows are functional relations. The BDD and IBD diagrams use different notational conventions as they capture different aspects of the description.

The physical interpretation in the BDD diagram requires, for a type X, the ability to recognize if an object a has the type X. We write this as $a : X$. For an arrow $f : X \to Y$, the interpretation requires the ability to recognize that the application $a.f$ for $a : X$ has type $a.f : Y$. In this sense the arrow assigns individuals of type X to an individual $x.f$ of type Y. For example, the interpretation of the 2-amino acid description requires the ability to recognize that an atom is a carbon atom, that any 2-amino acid molecule w has a carbon atom $w.p1$ as a component. This model implicitly assumes that one has a recognition procedure for an atom. The model will provide a procedure to recognize a 2-amino acid molecule in terms of its components and their bonding relationships. ¶

¶ In the Algos terminology the arrows are maps and the nodes are types. In Algos an *individual* $t : X$ is a map whose domain type is a special type One and so $t : One \to X$. This representation enables individuals to be represented as maps. More generally it is useful to require that for any map h with range type X the composition $h.f$ has range type Y. Path composition of the arrows in a directed graph satisfies the associativity conditions of the axioms for a category (Section 3.2.1). The type $Null$ is called the initial type. $Null$ is a subtype of any type.

2.2. *Classes and Individuals*

The language in which the axioms are expressed has a type symbol, $Thing$ and a type symbol, $Null$. All of the classes are subtypes of $Thing$. A class $Null$ is an "empty" type that is a subtype of any type. Individuals, classes, and subtypes are defined in Section 3. In the amino acid application we can assume that the class $Molecule$ is $Thing$. A membership predicate $\in$ is used to state that an individual is a member of a specific class. The predicate and its relation to type containment satisfies properties familiar from set theory. The classes, i.e., subtypes of $Thing$ have the usual boolean operations and subclass ordering between classes. For example the equation

$$C \sqcap H = Null. \tag{3}$$

expresses that the classes of carbon and hydrogen atoms are disjoint which means that the classes do not have members in common.

An individual is a map $t : One \to Thing$. One is a special type called the terminal type. The notation $a : Thing$ is an abbreviation for $a : One \to Thing$. By representing individuals as maps, a composition $a.f$ of a map $f : A \to B$ with an individual a is defined and $a.f$ is an individual. The semantics for a map $f : A \to B$ where A and B are classes is expressed in terms of a composition operation as:

$$x \in A \Rightarrow x.f \in B \tag{4}$$

where $x.f$ is the amine group for the molecule x. [‖]

2.2.1. *The BDD: Component Structure*

The 2-amino acids have both an amine group NH_2 and a carboxylic acid group $COOH$. The general formula is: $NH_2CHRCOOH$. In this formula the N, H, C, O are abbreviations for nitrogen, hydrogen, carbon, and oxygen atoms. The carbon atom next to the carboxyl group is called the $\alpha-carbon$ atom. R is a place holder for an organic substituent known as a "side-chain". For 2-amino acids a substituent is an atom or group of atoms with a hydrogen atom component which can be bonded to the α-carbon atom. If R is substituted by H the result is glycine.

The graph of nodes and arrows labelled BDD in Figure 2 describe the pattern of component decomposition for amino acid molecules. The *AminoAcid* node and the group nodes in Figure 2 are reifications of these types. The amino acid description contains these nodes together with atom nodes and the edges representing the part decomposition and the binding relationships. The nodes in the BDD diagram include *AminoAcid*, nodes for subcomponents, and atoms.

The arrows in this diagram, such as the arrow $p2 : AminoAcid \to NH_2$, and the arrow $q1 : NH_2 \to N$ are called part arrows. The BDD describes a part decomposition pattern which applies to individual amino acid molecules. Each arrow has a domain and

[‖] In Algos Section 3.2.4 the subtype relation $\sqsubseteq$ for a type A and a type X is defined, as is the membership relation $\in$ for a map and a type. A subtype A of X is a type construction. The subtype construction has the form $A = \{x : X | p(x) = true\}$ where $p : X \to \Omega$. Ω is the truth value type. The axioms enable the definition of algebra of subtypes which includes $\sqcap$, $\sqcup$, $\neg$. As well for subtypes A and B of a type X, one has that if $f : X \to Y$ and if $a \in A$ then $a.f \in B$.

a range node. The path composition of these arrows is written as $p2.q1$. The path $p2.q1$ has domain and range

$$p2.q1 : AminoAcid \rightarrow N \tag{5}$$

A usual graph theoretic "dot" notation is used with left to right ordering to compose maps, rather than the more usual right to left order with parenthesis for composition within mathematics. The absence of parenthesis reflects an associativity assumption for composition.

The component structure of each amino acid instance is described as having the two groups NH_2 and $COOH$ together with the α-carbon atom, a hydrogen atom, and the side chain $\mathbf{R}$. There are five top level components and each of the two groups has components. For each of the five components a map is introduced. Each of the five maps has domain the class *AminoAcid*. The range classes of these maps are respective classes of the five top level components. The notation for these 5 maps is:

$$p_1 : AminoAcid \rightarrow C \tag{6}$$
$$p_2 : AminoAcid \rightarrow NH_2 \tag{7}$$
$$p_3 : AminoAcid \rightarrow COOH \tag{8}$$
$$p_4 : AminoAcid \rightarrow H \tag{9}$$
$$p_5 : AminoAcid \rightarrow \mathbf{R} \tag{10}$$

The subcomponents of NH_2 and $COOH$ are also specified in the Block Definition Diagram (BDD) of Figure 2. For NH_2 as a standalone entity its components are:

$$q1 : NH_2 \rightarrow N \tag{11}$$
$$q2 : NH_2 \rightarrow H \tag{12}$$
$$q3 : NH_2 \rightarrow H. \tag{13}$$

The NH_2 components of *AminoAcid* are the compositions:

$$p2.q1 : AminoAcid \rightarrow N \tag{14}$$
$$p2.q2 : AminoAcid \rightarrow H \tag{15}$$
$$p2.q3 : AminoAcid \rightarrow H. \tag{16}$$

The subcomponents for $COOH$ are specified similarly. They are displayed on the BDD of Figure 2.

While a map such as $p1 : AminoAcit \rightarrow C$ assigns a carbon atom to an amino acid molecule m, not all carbon atoms are a component of an amino acid. Further a molecule may have multiple carbon atom components. These considerations lead to introducing the construction for the image $Im(f)$ of a map $f : X \rightarrow Y$. The image of a map enables describing the bonding relations which are defined in the IBD of the amino acid model.

2.2.2. *The IBD:Bonding Structure* The bonding relationships are described in the diagram labelled IBD in Figure 2. The axioms for the connections ensure that a component of one molecule is bonded to a specific component of that same molecule. This will be achieved by map composition equations. Note that the rectangles in the IBD are labeled with expressions such as $p4 : H$ and $p1 : C$. informally $p4 : H$ is the type of hydrogen atoms that serve as the $p4$ component of an amino acid molecule. Thus for any molecule $a : AminoAcid$ the composition $a.p4$ is the hydrogen atom of a. The interpretation of $p4 : H$ is as the image of the map $p4$. We use both the graphical syntax $p4 : H$ and the syntax $Im(p4)$ for the image of the map.

The IBD has ten connections between the atoms. They are represented as maps whose domains and ranges are the image classes of the part maps. Connection equations are used to ensure that the components of a molecule are connected by bond maps. The IBD defines these equations. For example, in the $COOH$ group the carbon atom has a double bond with the oxygen atom. The two bonding equations are:

$$p3.r1.c6 = p3.r5 \tag{17}$$

$$p3.r5.c7 = p3.r1. \tag{18}$$

The BDD and IBD diagrams are an incomplete description of the amino acid component decomposition. Additional axioms are needed to reflect implicit assumptions of Figure 1.

2.2.3. *Components are distinct* We expect that all of the components of an amino acid molecule are distinct. We can say that the classes representing atoms are pairwise disjoint, but this doesn't suffice as multiple part paths may have the same range. Axioms for disjointness of atom classes are given in terms of composing individuals with maps. The distinctness of the values of two maps such as $q2 : NH_2 \to N$ and $q3 : NH_2 \to N$ can be expresses by

$$x \in NH_2 \Rightarrow x.q2 \neq x.q3. \tag{19}$$

In expression (19) the symbol x is an individual variable. The orthogonality condition can be written as:

$$q2 \perp q3 \tag{20}$$

and is equivalent to $Im(q2) \perp Im(q3)$.

2.2.4. *No sharing of atoms between molecules* We haven't yet ruled out that a carbon atom in an amino acid can not belong to some other molecule. To ensure that a component does not belong to any other molecule we assume for each part property and any two molecules $m1$ and $m2$ that

$$m1.p = m2.p \Rightarrow m1 = m2. \tag{21}$$

A map with this property is called a monic. [††] Each map $f : A \to B$ has an image $Im(f)$. The image of f is a subtype type of B. This property can be expressed as:

$$b \in Im(f) \Rightarrow \exists x.x.f = b. \tag{22}$$

The characterization with the existential quantifier is not needed as the maps are monic, which allows a more algebraic characterization. For a monic f there is an inverse map

[††] The Algos monic property provides the isomorphism of the monic with its image.

f^{-1} whose domain is $Im(f)$ and whose range is A. The map f^{-1} has the property that

$$f.f^{-1} = id_A. \tag{23}$$

In the amino acid example $Im(p1)$ are the carbon atoms which are a component of a 2-amino acid molecule. Presumably $Im(p1)$ is small in comparison with C. All of the maps used in this example are assumed to be reversible in this way. That is they have an inverse from their image back to their domain. The inverse part maps imply no two realizations intersect. In the IBD diagram the notation $f : B$ is used in place of $Im(f)$. However, for the linear syntax we will continue to use $Im(f)$ for the image.

Further each of these connection maps is assumed to be monic. For example a direct transcription of the typing of $c1$ in the IBD of figure 4 is

$$c1 : (p1 : C) \to (p4 : H). \tag{24}$$

Using the image notation this becomes

$$c1 : Im(p1) \to Im(p4) \tag{25}$$

with inverse

$$c1^{-1} : Im(p4.c1) \to Im(p1). \tag{26}$$

Note the degree of a component such as the $\alpha-$carbon is the number of connections the atom participates in. The $\alpha-$carbon atom of a molecule in this axiom set is the value of $p1.m$ where m is an amino acid molecule. A 2-amino acid only contains one cycle. The composition

$$p3.r1.b1.b^{-1} \tag{27}$$

is a cyclic in the sense that its domain and range are the same. Checking that a map is cyclic is trivial. Inverse maps may be used in the cycle.

2.3. *Axioms with variables*

Up to this point variables haven't been needed to express axioms. The axioms to exclude extra connections and the treatment of substitutients use variables. The variables are typed by "classification" predicates. These predicates are $Individual, Class, Property$, $Node, Part$, $PartPath$, and $Connection$. When a user constructs an axiom set, constant symbols like *AminoAcid* and variables like R are declared as part of the axiom set using classification predicates. These declarations add axioms to the axiom set as described in Section 3, which are used for reasoning. The type symbols *Thing* and *Null* are always included, as is the image construction.

A variable $\mathbf{x}$ with type *PartPath* written as $\mathbf{x} :: PartPath$. For example,

$$\begin{gathered}\mathbf{x} :: Individual, \mathbf{x} \in AminoAcid, \\ \mathbf{p}, \mathbf{q} :: PartPath, \Rightarrow \mathbf{x}.\mathbf{p} \neq \mathbf{x}.\mathbf{q}\end{gathered} \tag{28}$$

can be used to state that all part paths from a node have distinct values. As an example the property that a class doesn't have any non-trivial subclasses defined as a unary predicate with:

$$Atom(\mathbf{C}) \equiv \mathbf{A} \sqsubseteq \mathbf{C} \Rightarrow \mathbf{A} = Null \tag{29}$$

where $\mathbf{A}$ and $\mathbf{C}$ are class variables. The predicate *Atom* can be used to characterize atoms in a mathematical sense.

2.3.1. *Exclusions* In the axiom set for amino acids the atoms,N, H, C, O are all explicitly declared as classes that have type *Node*, for example $N :: Node$. To exclude an amino acid molecule from having other components or being a part of any other structure we put

$$\begin{aligned} \mathbf{p} :: Part \Rightarrow domain(\mathbf{p}) :: Node&, \\ range(\mathbf{p}) :: Node&. \end{aligned} \tag{30}$$

To exclude a carbon atom in an amino acid molecule from being connected to any other atom we add the clause

$$\mathbf{f} :: Conn \Rightarrow Range(\mathbf{f}) \neq Carbon \tag{31}$$

A class *AminoAcid* has the property that no map has *AminoAcid* as its range. A class with this property is called a start class; it is defined as

$$Start(\mathbf{A}) \equiv \mathbf{p} :: Part, \Rightarrow Dom(\mathbf{p}) \neq \mathbf{A}. \tag{32}$$

The axiom set contains the axiom that *AminoAcid* is a start class:

$$Start(AminoAcid). \tag{33}$$

2.3.2. *Substitutients* Both molecular structure descriptions and automobile design specifications are used to describe variants. A variant description is one in which there are “substituents” which can be replaced by descriptions which have specific structure. The amino acid description describes the class of 2-amino acids which are obtained by replacing $\mathbf{R}$ by a “side-chain” which has a hydrogen atom and by connecting the hydrogen atom to the $\alpha - carbon$ atom. If $\mathbf{R}$ is replaced by N the result describes the lysine class. The side chain has to have a hydrogen atom available for bonding connections. From the BDD the part map for $\mathbf{R}$ is

$$p5 : AminoAcid \to \mathbf{R}. \tag{34}$$

In the IBD there is a connection map

$$b : p1.C \to \mathbf{R}. \tag{35}$$

A part relation *hasPart* is introduced to represent the conditions which enable a hydrogen molecule which is a component of $\mathbf{R}$ and satisfies conditions which make it bondable to the α-carbon atom. The condition that $\mathbf{R}$ is a side chain can be expressed by:

$$\begin{gathered} SideChain(\mathbf{R}) \equiv \\ \mathbf{R} \sqsubseteq \exists hasPart(\mathbf{R}, H) \end{gathered} \tag{36}$$

The inclusion statement is equivalent to

$$x \in \mathbf{R} \Rightarrow \exists y. < x, y > \in hasPart, y \in H. \tag{37}$$

However, this expression can be replaced with the class inclusion statement for purposes of computation.

2.4. *Relations, Multiplicities, object diagrams*

The graphical notation we have used permits arrows to have integer multiplicities. If no multiplicity is present it is assumed to be one. In the amino acid example all multiplicities are assumed to be one. The multiplicity 1 of an arrow $f : A \to B$ means that if $a : Thing$ then $a.f$ has a single value. The arrow notation $f : X \to Y[k]$ stands for a multivalued map $f : X \to Pow(Y)$ where $Pow(Y)$ is called the power type. For any $a : W$ the value $a.f$ has type $Pow(Y)$. The type $Pow(Y)$ has a cardinality map $card : Pow(Y) \to N$. The value of the cardinality $card(a.f)$ is k, if the multiplicity is k. If an arrow $f : X \to Y$ has a multiplicity of k then the multivalued map $f : X \to Pow(Y)$ can be replaced by k maps $f_1, \ldots, f_k$.

The amino acid model is, in the terminology of object-oriented programming, a class model, as the nodes represent types of obejcts rather than individual objects. In object-oriented programming languages class diagrams and object diagrams are both directed graphs. An object diagram may be obtained from a class diagram by replacing each class X in the diagram with an instance $a : Thing$ where $a \in X$. Each map $p : X \to Y$ in a class diagram is replaced by an ordered pair $< a, a.p >$. An individual amino acid molecule is a graph whose nodes are individual members of the appropriate class and the edges connect the individuals as in the diagram. For an individual a with $a \in AminoAcid$ the part path composition terms such as $p_i.a$ are members of the designated class and the pairs such as $< a, p_i.a >$ are the edges of the graph. [‡‡]

2.5. *The Amino Acid Description*

The amino acid graph in Figure 2 declares map and class symbols together with the typing axioms for classes and maps. The amino acid axiom set further contains the disjointness axioms for classes and the orthogonality axioms for the part maps together with the exclusion and substitutuent axioms. The axiom set for the amino acid class can be represented in a linear syntax as well as a graphical syntax. As a result we can say, for example, that any amino acid molecule is not a hydrocarbon in the sense that it only contains carbon atoms and contains a four-membered ring. The axioms exclude unintended additional components. The axiom set is a schema in that the rectangle

[‡‡] The product and tuple constructions are used to represent binary relations (roles). A binary relation P is a subtype of a product type (X, Y) which means that it representable as a subtype $\{< x, y >: (X, Y) | r(x, y) = true\}$ for some $r : (X, Y) = \Omega$. Products are discussed in section (3.2.2). In Section 3.2.5 the correspondence a binary relation r a multi-valued map $r^* : X \to Pow(Y)$ is established. This correspondence enables functional relation $P \sqsubseteq (X, Y)$ to be replaced by map $f_P X \to Y$ (Section 3.2.7).

labelled R in Figure 1 can be substituted for to obtain specific configurations such as glycine. The amino acid axioms represent $\mathbf{R}$ as a class variable. In Section 4 these axioms will be used to verify that any realization of a 2-amino acid axiom set without variables has a canonical structure.

3. Algos Formalism

This section outlines the Algos formalism (Graves and Blaine 1985; Graves and Blaine 1986). Algos is an Extended Standard Formalism derived from the axioms for an elementary topos. This formalism has been implicitly used for the 2-amino acid axiomatization in Section 2. Section 2 is intended to illustrate the naturalness of the language constructions in applications. Section 3.1 presents the formal language. Section 3.2 presents the axioms for the term constructors, consequences of the axioms, and the definition of the theory generated by an axiom set. Section 3.3 provides the embedding of an Algos axiom set within a topos, called the syntactic topos generated by the axiom set. This embedding establishes the mathematical soundness of Algos.

The choice of language constructions and axioms for Algos was motivated by the idea of developing a logic-based formalism to be used interactively with automated theorem proving and proof checking by scientists and engineers. The system would be used for everyday construction of axiom sets to be used as models in the engineering sense. A model as axiom set can be either a description of a system of interest or a specification for a system to be built. Many engineering problems translate into logic questions regarding the engineering model. This has been elaborated in (Graves and Bijan 2011; Graves 2012). Intended applicationS include design specification for manufactured products and biomedical engineering.

The following criteria were used in the construction of the formalism.

1 The formalism generates a topos.
2 The language constructions and axioms are directly verifiable in applications.
3 The formalism can be implemented effectively as a computational system with automated reasoning.
4 The formalism is practically usable by engineers and scientists to build and analyse axiom sets for complex applications.

The first condition is for mathematical soundness. The Algos language constructions include the topos construction of product (X, Y), subtype $\{x : X|p(x) = true\}$ and power $Pow(X)$ types with corresponding map constructions. Algos also includes the sum $X + Y$ and the exponential Y^X types, as well. the complete description includes n-ary sum and product types, and axioms for the natural number type, N. However, these constructions are not discussed in as much detail, as they can be defined from the ones given. These constructions are not used in the biomedical example, but are used in the Composite Structure Models (Graves 2012). The Algos term constructions contain a truth value type Ω. The maps with range type Ω are called internal formulae. The first order formulae in which the axioms for the terms are expressed is called the *external logic*. The resulting Algos constructor axioms imply that an Algos axiom set generates a topos, but are stronger as the generated topos has canonical subobjects (see Section 3.3) (Lambek and Scott 1980), page 138.

The second condition is physical soundness. The physical soundness of Algos rests on the ability to interpret the language and axioms in applications. The scope of applications is the macro physical world as opposed to the world of quantum mechanics. Remarkably sufficient conditions for a topos occur directly in applications. Section 2 gave an informal example of application axioms which use a set of language constructions that are sufficient to generate a topos. From this viewpoint toposes abound in applications. As the language constructions are presented, the rationale for their choice is discussed. The axioms are presented in an "incremental" order. Each axiom enables an increasing expressiveness for applications. Further, each axiom enables additional correspondence between the external logic in which the axioms are presented and the internal logic.

The third condition is mechanization of reasoning, i.e., can the axioms be used computationally for automated reasoning. The language axioms have the form of a Horn Clauses with a single consequent in the form of an equation. When the axioms were first developed the ability to give equational axioms for the Cartesian Closed categories (CCC) was known and the fact that the CCC terms have a unique canonical form. A form on an equalizer axiom (Axiom 3.2.4) similar to the one expressed in (Lambek and Scott 1980) page 22 is used. The power type axiom similar to that in (Lambek and Scott 1980) page 163 is used. However, it did not appear to be known at the time how a subobject classification axiom could be expressed equationally, as is done in Axiom 3.16. Beyond presenting the axioms in a form familiar from mechanization of logic, and beyond noting that the system has been successfully implemented (Graves and Blaine 1985; Graves and Blaine 1986) this topic is not discussed here.

The fourth condition is practical usability. The actual adoption of a formalism depends on whether there are software tools available in which engineers and scientists can develop axiom sets for applications of interest. The graphical syntax of SysML corresponds closely to the linear syntax of Algos. The detailed design of large complex manufactured products are routinely produced in SysML augmented with models in other special purpose modelling languages. The SysML syntax does not include all of the Algos constructions, but they could be added. Algos and its implementation predated SysML by about 30 years. Complex SysML models have been developed which have been informally translated into Algos (Graves 2012). As a result existing tools can be adapted to use for axiom development in Algos. Since the original publications regarding Algos the axioms have been modified somewhat based on engineering experience.

One difference between the SysML syntax in Section 2 and the linear syntax of Algos, as presented in this section, are that map composition in this section is written in reverse order from arrow composition in SysML. When maps occur as arrows in diagrams composition will be written in left to right order without parenthesis as is done for path composition in directed graphs. Otherwise composition will be written with parenthesis. In the amino acid example all of the compositions were written as path compositions. However, the axioms will be given in the more usual mathematical notation as they will look more familiar. Applications which use applicative operations with arguments, such as a calculator which takes inputs and produces outputs, typically use the traditional mathematical syntax. Comments on differences in notation and formal presentations of topos theory and the relation of Algos to Type theory are given in footnotes. These issues of graphical syntax are not the focus of this discussion.

Other issues which are also not discussed are the logical incompleteness of the Algos axioms. More importantly, applications generally involve operating with multiple theories, combining theories, and refining them. This is characteristic of typical engineering modelling processes. This can be referred to as physical incompleteness. This topic is not addressed.

3.1. *Language*

As an Extended Standard Formalism, Algos uses a language of terms and formulae. The inference rules are a subset of first order proof constructions. The inference rules will be stated explicitly as they were implemented directly within an automated reasoning system in that form. The language uses several notational conventions to simplify the syntax of formula. These syntactic conventions are sometimes different from mathematics, but will be familiar to users of programming and modelling languages. They will be introduced and commented on below.

3.1.1. *Signature* An Algos signature consists of the two sorts Map and $Type$, with a collection of constant map and type symbols, a collection of function symbols, used as constructors for the map and type terms, and predicate symbols. The notation will correspond to informal mathematics notation. Rather than listing the signature symbols here they will be introduced as they are used. The type symbols include One the terminal type, $Null$ the empty type, Ω the truth value type, and N. The map and type constructions are first order function symbols which include map constructions to construct ordered pairs of maps and Cartesian Product types are function symbols in the signature. The function symbols include *Domain* and *Range*. We use the notation $Domain :: Map \to Type$ and $Range :: Map \to Type$ to indicate the sort of arguments and values of the domain and range function symbols. Having function symbols in the signature does not imply that they are defined for all values of their arguments. An Algos signature the symbols informally described with possibly additional symbols. When a map constant symbol is included in a signature its domain and range types must be specified.

It is worth noting that when engineering models are developed and computer programs are written they use declarations to introduce symbols with typing information. The symbols introduced by declarations together with the symbols used for the language constructions are the signature of the model as axiom set.

3.1.2. *Terms* The language of term constructions is the language of maps and types (objects) in a topos. Terms may contain variables with sorts Map and $Type$. Generally lower case letters f, g, h, t, s are used for map variables, and upper case letters A, B, C, X, Y, Z are used for type variables. Occasionally the notation $f :: Map$ may be used to indicate that f is a map variable, and $X :: Type$ to indicate that X is a type variable. Terms are constructed from map and type symbols using the term constructions. For example

$< a, b >$ is the notation for tuple and (A, B) is the notation for Cartesian product. However, the axioms for the term constructions, such as $<, >$ have antecedent conditions for the term constructions to be well formed. Since the constant map symbols are typed and the map constructors provide typing conditions for being well-formed the well formed map terms have a unique typing. In informal presentation the types are not always given explicitly. However, they can be inferred from the context.

3.1.3. *Formulae* The formulae are Horn clauses constructed from literals (atoms and negation of atoms) using the predicate symbols include $\in$, $\sqsubseteq$, $:$, and $=$, together with auxiliary predicates introduced here and application specific predicates. Free variables in a formula are implicitly universally quantified. The notation

$$P1, \ldots, Pn \Rightarrow Q \tag{38}$$

is used for a Horn clause where Pi and Q are literals. [§§] The terms include variables typed with the sorts *Map* and *Type*. The map terms are related to the type terms through the *Domain* and *Range* functions. The notation

$$f : X \to Y \tag{40}$$

is a three argument predicate which is an abbreviation for the two binary predicates

$$Domain(f) = X, Range(f) = Y. \tag{41}$$

The atomic formula $f : X$ is an abbreviation for $Range(f) = X$. The typing of map symbols in the signature is specified. A map term f is well-formed if there are types X and Y for which $f : X \to Y$ is provable. An equation $t1 = t2$ for two terms $t1$ and $t2$ is well-formed provided the terms have the same typing. The equality predicate $=$ has the usual properties for terms with the same type.

The map term constructors are first order function symbols whose arguments are types and whose values are maps. For example, there is a constructor which assigns an identity map to each type. The symbol *id* is a function symbol whose argument is a type. The notation id_X is used for the identify map of a type X. The identity function symbol has typing $id :: Type \to Map$. Map and type variables may be further subtyped using relations defined by predicates. For example, if f is a map variable and $Range(f) = X$ for a type term X the notion $f :: X$ will be used as an abbreviation. Any symbols in a formula which are not constant symbols are variables.

Additional predicates, such as the binary type predicate for isomorphism, are introduced as Horn clauses for which they are the consequent. These predicates will be called definitions, but they are not all equivalences. The antecedent formula contains variables, which, when closed terms are supplied, enables the conclusion of the predicate to be derived.

[§§] While we write the formulae as Horn clauses they can also be viewed as entailment relations and be written using the notation

$$P1, \ldots, Pn \vdash_S Q \tag{39}$$

where S is the set of free variables in the literals. By representing inference using entailment Algos is a type theory.

3.1.4. *Deduction system* While the inference rules are a subset of first order proof construction rules, they can also be viewed directly as a deduction system. A deduction system, as defined in (Lambek and Scott 1980) page 47, consists of formulae and deductions constructed from inference rules. In this case the formulae are Horn clauses, the deductions are the axioms and deductions constructed from inference rules. The inference rules are presented in a numerator-denominator form where both numerator and denominator are Horn clauses. These rules are stated explicitly below. The notation $P[f/t]$ is used for substituting the term t for the variable f in the literal P. Substitution of a map term for a map variable is well-formed only if they have the same typing. The notation Γ will be used for a sequence $P1, \ldots, Pn$ of literals.

1 **Thinning**

$$\frac{\Gamma \Rightarrow P}{Q\ \Gamma \Rightarrow P} \tag{42}$$

2 **Cut**

$$\frac{P1 \ldots Pn \Rightarrow P, P1 \ldots Pn, P \Rightarrow Q}{P1 \ldots Pn \Rightarrow Q} \tag{43}$$

3 **Substitution**

$$\frac{\Gamma \Rightarrow P}{\Gamma[f/t] \Rightarrow P[f/t]} \tag{44}$$

4 **Equality**

$$\begin{aligned} f &= f \\ f = g &\Rightarrow g = f \\ f = g, g = h &\Rightarrow f = h \end{aligned} \tag{45}$$

A map variable in a formula may be replaced by a map term provided any type assumptions of the map variable are satisfied by the map term. These inference rules correspond to structural deduction rules used by type theories. The Algos language construction axioms below can also be viewed as inference rules. Together these rules enable Algos to be viewed as a type theory under the usage of the term in (Lambek and Scott 1980). Map constructions have axioms which give antecedent conditions for the term to be well-formed.

3.2. *Axiom Sets and Theories*

The axioms for the term constructions (Algos axioms) provide the semantics for the term constructions. The Algos axioms are derived from the first order axioms for a topos by adding as term constructors first order function symbols to replace existential quantifiers. The Algos term construction axioms have the form that an antecedent is a conjunction of literals formed from the atomic formulae and that the consequent is an equation.

For application axiom sets we keep the restriction that the formula are Horn clauses. An Algos axiom set is a collection of Horn clauses in the language generated by an Algos signature. An Algos signature may contain additional map and type constants, and atomic predicates. The Algos theory generated by an axiom set is the closure of the axioms with the axioms for the term constructions using the inference rules. The formulae in the theory of an axiom set are Horn clauses.

3.3. *Category Axioms*

The first choice to be made when developing an Extended Standard Formalism is whether to use a type system, and if so what kind. The Algos formalism uses a type system with two kinds of terms, map and types. Map terms have two types, a domain and a range type rather than a single range type as is done in many type theories. The choice of maps with both a domain and a range type rather than terms with a single "range" type is based on the fact that most engineering applications use a graph-theoretic interpretation of maps and composition as path composition. Often engineering modelling languages treat maps as functional relations. The relations in these languages have domain and range types. This suggests that the axioms should be written with the path notation for composition. However, in the interest of mathematical familiarity the axioms will be given in the usual left-to-right ordering.

The first order axioms for a category using the two sorts Map and $Type$ are the basic map construction for Algos. The three place predicate $f : X \to Y$ is an abbreviation for the conjunction of the two binary predicates $Domain(f) = X$ and $Range(f) = Y$. Thus

$$f : X \to Y \equiv Domain(f) = X, Range(f) = Y. \tag{46}$$

Axiom 3.1 (Category).

$$\begin{aligned} f : A \to B \Rightarrow& \\ Domain(f) = A, Range(f) = B& \qquad (47) \\ f : A \to B, g : B \to C& \\ \Rightarrow g(f) : A \to C& \qquad (48) \\ f(g)(h) = f(g(h))& \qquad (49) \\ f(id_A) = id_B(f) = f.& \qquad (50) \end{aligned}$$

For the composition of a map $f : X \to Y$ with a map $g : Z \to X$ to be interpreted requires only that the term $f(g)$ is well defined and for which $Range(f(g)) = Y$. For each type X the identify map constructor $id_{.}$ provides a map id_X for each type X with $id_X : X \to X$.

The category axioms are a simple generalization of the axioms for a directed graph where the maps are path compositions of arrows and the types are the nodes. The justification for associativity comes from viewing map composition as path composition in directed graphs. Keep in mind that in the amino acid the compositions are written in reverse order using the 'dot' notation which implies the associativity of composition. The same argument applies to the Composite Structure Models such as a vehicle and its test environment.

An application such as a design for a vehicle specifies that any vehicle of a type $Vehicle$ has an engine of type $Engine$. The descriptive property of having an engine can be represented directly as a map $hasEngine : Vehicle \to Engine$. When attempting to verify in an application that an object satisfies its specification, one of the tasks would be to determine if the object has an assigned engine of the appropriate type. When complex manufactured products are delivered to a customer it is very common for this task to be performed before the product is accepted. When constructing an axiom set for a molecule an interpretation of a map in a molecule axiom set such as $oxygen : Water \to Oxygen$ simply says the each water molecule has a well defined oxygen component. There is no requirement as to how this oxygen component is assigned to the water molecule.

The concept of two types being isomorphic is introduced with the following rule.

Definition 3.1 (Isomorphism).

$$f : A \to B, g : B \to A, f(g) = id_B, g(f) = id_A \Rightarrow A \simeq B \tag{51}$$

In Algos two types are said to be isomorphic only when the two maps are provided.

The category theory version of a map being one-one is called *monic*. Monics occur frequently in applications such as manufactured products and biomedicine. For example, in the amino acid axioms of Section 2 the part and connection maps of amino acid are declared to be monic. More generally in engineering models that have a unique parts decomposition the part maps are monics.

Definition 3.2 (monic). For $f : X \to Y$

$$Monic(f) \equiv f(h) = f(g) \Rightarrow h = g \tag{52}$$

Note that the composition of monics is a monic.

3.4. *Products and tuples*

For types X and Y the construction (X, Y) is a type called the product type of X and Y. The choice of the product and tuple constructions represent the ability to reify two distinct maps $f : Z \to X$ and $f : Z \to Y$ as a map $< f, g >$ called the ordered pair or tuple of the two maps. The tuple construction is well-formed provided that their domains of the maps f and g are the same. The type of the tuple is $< f, g >: Z \to (X, Y)$. For each product type (X, Y) has two projection maps maps $pr1_{X,Y} : (X, Y) \to X$ and $pr2_{X,Y} : (X, Y) \to Y$. The projection maps are constructors which have the product type as an argument. Notation will be introduced to allow users to name the projection maps. As we will see the projection maps are variables and the renaming conventions will be familiar from logic and computer science.

The terminal type *One* is a zero-ary product type. If $f : Z \to X$ and $g : Z \to Y$, the notation $< f, g >$ is used for an ordered pair and has type $< f, g >: Z \to (X, Y)$. For each type X the map ! has type $! : X \to One$. ! is a map constructor with a type as argument and is precisely written as $!_X$. However, the subscript will generally be omitted. Product types and tuples are used to represent relations which are subtypes of a product type; instances of relations are tuples. Maps have graphs which are relations.

The product and tuple axioms state that two tuples are equal if their components are equal, that a map whose range type is a product is equal to a tuple of maps, and for each type the terminal map ! is unique.

Axiom 3.2 (Tuple).

$$t1, r1 :: X, t2, r2 : Y, < t1, t2 >=< r1, r2 >\Rightarrow t1 = r1, t2 = r2 \tag{53}$$

$$f : Z \rightarrow (X, Y) \Rightarrow$$
$$f =< pr1_{X,Y}(f), pr2_{X,Y}(f) > . \tag{54}$$

$$< f, g > (h) =< f(h), g(h) > \tag{55}$$

$$f : X \rightarrow One \Rightarrow f =! \tag{56}$$

The pairing two maps $f : Z \rightarrow Y$ and $g : Z \rightarrow X$ has the property that $< f(h), g(h) >$ has a well defined type provided h has range type Z. The need for the domains of the two maps to be the same is to ensure that the composition of the tuple yields a well-formed map tuple map $< f(h), g(h) >$.

A consequence of the tuple axioms is that the identity of an ordered pair is the ordered pair of the projection maps.

$$id_{(X,Y)} =< pr1_{X,Y}, pr2_{X,Y} > . \tag{57}$$

Another consequence of the tuples and products is that a map $t : One \rightarrow X$ can be identified as a map with no arguments. This is the concept of an individual of type X. The notatin $t :: X$ is used for an individual of type X. The product type construction can be iterated to form n-ary products. The operation of constructing products and tuples is associative in that the resulting products are isomorphic and so the parenthesis will be omitted. A product of the form (One, X) is isomorphic to X. The isomorphism follows from the fact that the identity of (One, X) is $<!, pr2_X >$. The map $<!, id_X > (pr2) =< !, pr2 >$ and $pr2_X(<!, id_X >) = id_X$.

Algos uses a notational convention familiar from modelling and programming language to introduce names for projection maps. New names can be introduced for projection maps with the notation

$$(x : X, y : Y, z : Z). \tag{58}$$

This notation introduces a product type (X, Y, Z) where the projection maps have been renamed to be x, y, z. For example,

$$y : (X, Y, Z) \rightarrow Y. \tag{59}$$

is the projection of (X, Y, Z) onto Y. Algos contains a natural number type, N and arithmetic functions such as $+$. The usual infix notation

$$f + g = +(f, g) \tag{60}$$

is used for the composition. A consequence of the notational conventions for projection maps is that they have the properties associated with variables. For $x : (x : X, y : Y)$ a product type with projections x and y the notation

$$(x := f) =< f, y > \tag{61}$$

defines a substitution operation. For example, where x and y are the projection maps declared above

$$x + y(x := f) = +(< x, y >)(< f, y >) =$$
$$+(< x, y >)(< f, y >) = +(< x < f, y >, y < f, y >) =$$
$$+(f, y) = f + y. \tag{62}$$

This example shows that projections have the properties of variables and gives an example to illustrate that the Algos axioms can be used for symbolic computation. Since Algos has a natural number type N it has an iteration construction defined for a map $f : X \to X$.

$$Do(f, n) = for\, x \in [1. \dots .n]\, do\, f \tag{63}$$

iterates the map f n times. The construction is derived from the Lawvere axioms for the natural numbers type. The natural number axioms can be used to prove properties of maps by induction as a form of symbolic computation.

3.4.1. *Equality* Topos theory and Algos have a truth value type Ω with the maps $true : One \to \Omega$ and $false : One \to \Omega$. A map f with type $f : X \to \Omega$ is called an *internal formula*. The choice of using an equality map $eq : (X, X) \to \Omega$ enables a correspondence between testing whether $eq(f, g) = true$ and verifying the external equality $f = g$. The equality map constructor yields a map $eq_X(X, X) \to \Omega$ for each type X. The equality axiom provides a "reflection" between the internal logic of terms with type Ω and formulas in the external logic. The external logic is the Horn clause logic in which the Algos axioms are presented. For a topos the external logic used a full first order logic. The internal logic for Algos consists of the maps whose range type is Ω.

Axiom 3.3 (Equality).

$$true \neq false \tag{64}$$
$$t1, t2 :: X, eq(t1, t1) = true \equiv t1 = t2 \tag{65}$$

Note that while $eq : (z1 : Z, z2 : Z) \to \Omega$ we can tuple maps with domain Z and different ranges. For $f : X \to Z$ and $g : Y \to Z$ the map

$$< f(x), g(y) >: (x : X, y : Y) \to (Z, Z) \tag{66}$$

and so

$$eq(< f(x), g(y) >) : (x : X, y : Y) \to \Omega \tag{67}$$

is well formed.

The signature of an axiom set will always be the two truth values $true$ and $false$ but there can be additional maps with type $e : One \to \Omega$. An application axiom set may add other truth values or axioms which imply additional truth values. The logical operations: negation, meet, and join may be defined for truth values and have expected properties.

Definition 3.3 (propositional connectives). For $p, q : X \to \Omega$

$$\neg p = eq(p, false) \tag{68}$$
$$p \wedge q = eq(< p, q >, < true, true >) \tag{69}$$

$$p \vee q = \neg(p \wedge q) \tag{70}$$
$$p \Rightarrow q = eq(p \wedge q, p) \tag{71}$$

From the axioms introduced so far the truth values have the structure of a Heyting lattice. One example property is:

Lemma 3.1. For $x : Z \to X, p, q : X \to \Omega$

$$(p \wedge q)(x) = true \Rightarrow p(x) = true, q(x) = true \tag{72}$$

Proof.

$$(p \wedge q)(x) = p(x) \wedge q(x) = eq(< p(x), q(x) >, < true, true >) = true \tag{73}$$
$$< p(x), q(x) >=< true, true > \tag{74}$$
$$p(x) = true, q(x) = true. \tag{75}$$

$\square$

The converse of the implication is also true, i.e.,

$$p(x) = true, q(x) = true \Rightarrow (p \wedge q)(x) = true \tag{76}$$

The logical operations defined for each type enable the expression of pre and post conditions for maps. For $p : X \to \Omega, f : X \to Y, q : Y \to \Omega$ the formula

$$p(x) = true \Rightarrow q(f(x)) = true \tag{77}$$

represents a pre-post condition for the map f where x is an arbitrary map with range type X.

The relationship between the external equality and the internal equality map enable a correspondence between truth-valued maps, sometimes called internal formulae and (external) formulae in the language of an Algos signature. This correspondence will be used to characterize axiom sets and relate derived statements to model theoretic truth provided there are no other truth valued maps. Note that an axiom set can add map constants whose range is Ω. For example, adding a map $maybe : One \to \Omega$ results in four truth value maps as there will be its negation $maybenot = \neg(maybe)$.

3.5. *Equalizers and subtypes*

Two constructions, equalizers and subtypes, which are equivalent in the presence of equality occur frequently in applications. The equalizer type construction comes from the need to define subtypes of a type X where the values of two maps defined on X agree. The equalizer is a type construction parametrized by two maps. For $f, g : X \to Y$ the expression $X\{f, g\}$ is a type, called the equalizer of f and g. The type $X\{f, g\}$ has an inclusion map $incl_{f,g}$ with $incl_{f,g} : X\{f, g\} :\to X$. The characteristic property of the equalizer is that any map $h : Z \to X$ for which $f(h) = g(h)$ factors into the composition of the inclusion map $incl_{f,g}$ with a map $fac_{h,f,g} : Z \to X\{f, g\}$. The equalizer construction enables subtypes of types to be defined in terms of "commutative" diagrams between compositions of maps.

In an axiom set for the molecule water mentioned in the introduction, the type $Water$ is a subtype of an equalizer type $Water\{oxygen.bond1, hydrogen\}$. The property specified by the equalizer says that the composition $oxygen.bond1.h = hydrogen.h$ for any $h : Z \to Water$. The axiom is:

Axiom 3.4 (Equalizers). For two maps $f, g : (x : X) \to Y$

$$\begin{aligned} incl_{f,g} : X\{f,g\} \to X, \\ monic(incl_{f,g}), \\ h : Z \to X, f(h) = g(h) \Rightarrow \\ fac_{h,f,g} Z \to X, h = incl_{f,g}(fac_{h,f,g}) \end{aligned} \tag{78}$$

The notation

$$X\{f,g\} = \{x : X | eq(f(x), g(x)) = true\} \tag{79}$$

is also used for the equalizer. In the subtype the symbol x is an identity map of type X. Identity maps and more generally projection maps are internal variables. Projection maps are variables in that they can be substituted for and usual properties hold.

Subtypes are a special case of the equalizer construction. This construction is also called abstraction and comprehension. The subtype construction is weaker than its set theoretic counterpart. Limitation to defining subtypes in terms of internal formulae enables the avoidance of the usual set theoretic paradoxes. In set theory a subtype can be defined from an external formula. However, one can use internal formulae to define external formulae (the reflection principle). These external formulae can be used to simplify the notation for defining subtype.

When the two maps defining the equalizer are an internal formula $p : X \to \Omega$ and $true_X : X \to \Omega$ the notation

$$X\{p\} = \{x : X | p(x) = true\} \tag{80}$$

is a subtype. For this case the notation for subtypes is simplified to:

$$\begin{aligned} incl_p : \{x : X | p(x) = true)\} \to X, \\ fac_{h,p} : X \to \{x : X | p(x) = true)\} \end{aligned} \tag{81}$$

The axioms become:

$$\begin{aligned} monic(incl_p), \\ p(incl_p) = true \end{aligned} \tag{82}$$

$$p(h) = true \Rightarrow h = fac_{h,p}(incl_p) \tag{83}$$

Conversely, subtypes and the equality map can be used to define the equalizer. If $f, g : X \to Y$ then

$$\{x : X | eq(f(x), g(x) = true\} = X\{f,g\}. \tag{84}$$

The Algos/topos axioms imply that the subtypes of X have the properties of a Heyting lattice. The proofs of these properties make use of the subtype axiom.

Definition 3.4. For two subtypes A and B of a type X defined by internal formulae respectively $p : X \to \Omega$ and $q : X \to \Omega$ the binary type predicate $A \sqsubseteq B$ is defined as

$$A \sqsubseteq B \equiv q(incl_p) = true. \tag{85}$$

From the subtype property this implies that $incl_p$ factors through $incl_q$ to yield

$$incl_p = incl_q(fac_{incl_p,q}). \tag{86}$$

For subtypes A and B of X the following properties hold for the subtype predicate:

$$Null \sqsubseteq A \tag{87}$$

$$A \sqsubseteq X \tag{88}$$

$$A \sqsubseteq B, B \sqsubseteq A \Rightarrow A = B \tag{89}$$

$$A \sqsubseteq B, B \sqsubseteq C \Rightarrow A \sqsubseteq C \tag{90}$$

If $A = \{x : X|p(x) = true\}$ and $B = \{x : X|q(x) = true\}$ the Subtype algebra operations are defined

$$A \sqcap B = \{x : X|p(x) \wedge q(x) = true\} \tag{91}$$

$$A \sqcup B = \{x : X|p(x) \vee q(x) = true\} \tag{92}$$

$$\neg A = \{x : X|\neg p(x) = true\}. \tag{93}$$

For example one can show that $\sqcap$ is the greatest lower bound.

Lemma 3.2. For $A = \{x : X|p(x) = true\}$ and $B = \{x : X|q(x) = true\}$

$$A \sqcap B \sqsubseteq A, A \sqcap B \sqsubseteq B \tag{94}$$

$$M \sqsubseteq A, M \sqsubseteq B \Rightarrow M \sqsubseteq A \sqcap B \tag{95}$$

Proof. To show that $A \sqcap B \sqsubseteq A$ note that

$$p(incl_{p,q}) = true \tag{96}$$

$$(p \wedge q)(x) = true \Rightarrow p(x) = true \tag{97}$$

$$p \wedge q(incl_{p\wedge q}) = true \Rightarrow p(incl_{p\wedge q}) = true \tag{98}$$

To show $A \sqcap B$ is the greatest lower bound of A and B assume $M = \{x : X|r(x) = true\}$ and

$$M \sqsubseteq A, M \sqsubseteq B. \tag{99}$$

Then

$$p(incl_r) = true, q(incl_r) = true \Rightarrow (p \wedge q)(incl_r) = true. \tag{100}$$

□

One of the Algos predicate symbols is $\in$ which has a map and a type argument.

Definition 3.5. For any type X and any map $t : Y \to X$ a membership relation is defined as

$$t \in \{x : X|p(t) = true\}. \tag{101}$$

The logical connectives can also be represented by maps. For example if $A \sqsubseteq B$ where A and B are subtypes of X defined by $p, q : X \rightarrow \Omega$ then

$$t \in A \Rightarrow t \in B \tag{102}$$

then

$$A \sqsubseteq B \Rightarrow (q \Rightarrow p)(t) = true. \tag{103}$$

These results will be improved on significantly by making use of the power type construction.

3.6. *Power Type*

The power type $Pow(X)$ construction for a type X defines a correspondence between binary relations, i.e., subtypes of a product type and individuals in the power type and maps from *One* to $Pow(X)$. The internal logic quantifiers enable the Description Logic class constructions $\exists P.B$ and $\forall P.B$ to be defined where P is a subtype of a product type and B is subtype of a type X. The power type construction is equipped with a constant maps $\epsilon_Y : (Y, Pow(Y)) \rightarrow \Omega$ which is an internal form of a membership predicate.

As usual, ϵ is an abbreviation for the constructor ϵ_X whose argument is a type. The power type construction also has an operator $*$ which provides a correspondence between a map $r : (x : X, y : Y) \rightarrow \Omega$, a non-deterministic map $r^* : X \rightarrow Pow(Y)$. The use of the notation of $\{x : X|p(x) = true\}$ for a subtype of X differs from the notation of (Lambek and Scott 1980) where that notation is used for a member of the powertype. In both systems there is a one-one correspondence between subtypes of a type and the members of the power type of the type. The use of the subtype construction is more in keeping with set theory usage.

The power type axiom corresponds a binary formula $r(x : X, y : Y) \rightarrow \Omega$ or equivalently a binary relation subtype $\{< x, y >: (X, Y)|r(x, y) = true\}$ with a map $r^* : X \rightarrow Pow(Y)$ which is called a non-deterministic map.

Axiom 3.5 (Power Type).

$$\epsilon : (y : Y, z : Pow(Y)) \rightarrow \Omega \tag{104}$$

$$\epsilon(y, r^*(x)) = true \equiv r(x, y) = true \tag{105}$$

Note that

$$h(x, y) = true \equiv \; y \in \{y|\epsilon(y, h^*(x)) = true\} \tag{106}$$

The universal quantifier for a map $r : (x : X, y : Y) \rightarrow \Omega$ is defined as:

$$\forall y.r = eq(true^*, r^*) \tag{107}$$

and is typed as

$$\forall y.r : X \rightarrow \Omega. \tag{108}$$

The existential quantifier is defined as

$$\exists y.r = \neg eq(false^*, r^*). \tag{109}$$

The quantifiers $\exists ! y.r$ and $\exists y[k].r$ can be defined as well. The internal logic is a quantification logic. The language uses the Algos types and projection maps for variables. Any internal formula is the characteristic map whose domain type is the product of the range types of the variables occurring in map term defining the characteristic map.

Using the quantification maps the existential and universal type constructions can be defined where $R = \{< x, y >: (X, Y)|r(x, y) = true\}$.

$$\forall P.B = \\ \{x : X|\forall y. < x, y >\in P \Rightarrow y \in B\} \tag{110}$$

$$\exists P.B = \\ \{x : X|\exists y. < x, y >\in P \wedge y \in B\}. \tag{111}$$

where R is an subtype

$$R = \{< x, y >: (x : X, y : Y)|r(x, y) = true\} \tag{112}$$

The domain and range of R are defined by

$$Domain(P) = \{x : X|\exists y.r(x, y) = true\} \tag{113}$$

$$Range(R) = \{y : Y|\exists x.r(x, y) = true\} \tag{114}$$

For a subtype R the relation predicate $R :: Relation(A, B)$ is defined as

$$R :: Relation(A, B) \equiv Domain(R) = A, Range(R) = B. \tag{115}$$

Note that each map $f : X \rightarrow Y$ has an image defined by

$$Im(f(x : X)) = \{y : Y|\exists x.eq(y, f(x)) = true\} \tag{116}$$

With the image map orthogonality can be defined for $f : X \rightarrow Y, g : Z \rightarrow Y$ as

$$f \perp g \equiv Im(f) \sqcap Im(g) = Null. \tag{117}$$

The preceding definitions of subtype operations in terms of characteristic maps can be summarized in terms of a theorem.

Theorem 3.1. The subtypes of a type X are closed under the DL operations of $Null$, X, $\sqcap$, $\sqcup$, $\neg$, $\exists P.A$, and $\forall P.A$.

The use of the power type enables a correspondence between subtypes and relations and individuals of the power types. For example if $A = \{x : X|p(x) = true\}$ and

$$A \sqsubseteq X \tag{118}$$

the map

$$p : (One, X) \rightarrow X \tag{119}$$

corresponds to the individual

$$p^{'} = p^{*} : One \rightarrow Pow(X). \tag{120}$$

The correspondence of a subtype A of X with an individual $A^{'}$ uses the fact that (One, X)

is isomorphic to X. The predicate $\sqsubseteq$ corresponds to a map

$$\sqsubseteq : (Pow(X), Pow(X)) \rightarrow \Omega \tag{121}$$

defined in terms of the membership predicate. In Algos notation $\{x : X|p(x) = true\}$ is used for subtypes whereas in (Lambek and Scott 1980) this notation is used for the corresponding element of $Pow(X)$. The Algos notation corresponds closely to the informal set theory usage the subtype construction.

Note that for a type X with subtypes A and B and $P \sqsubseteq (A, B)$ the formula

$$P \sqsubseteq (A, B) \tag{122}$$

is equivalent to the Description Logic formula

$$A \sqsubseteq \exists P.B \tag{123}$$

3.7. *Monic Classification*

The monic classification axiom enables the image of a monic $m : Y \rightarrow X$ to be characterized as a subtype of the range of the monic and provides an inverse map $m^{-1} : Im(m) \rightarrow Y$ so that the domain of the monic is isomorphic to its image. In engineering models representing component structure the maps specifying the components of a type are often monics. In the water model the map from the water type to the oxygen type which assigns an oxygen atom to each water molecule is a monic. This construction enables distinguishing between those atoms which are a component of a water molecule and those which are not a component of a water molecule and ensures that the oxygen atom of a water molecule is not a part of some other water molecule as well. Graph theoretically the monic property translates into bi-directional arrows.

A major consequence of the monic axiom with the prior axioms is that a functional relation $F :: Relation(A, B)$ can be replaced by a map $f : A \rightarrow B$. The map determined by the functional relation is Russell's description operator. The replacement of a functional relation can be generalized to allow for the replacement of a relation with multiplicity k by k maps.

For $m : Y \rightarrow Z, monic(m)$ let

$$char_m(z : Z) = \exists! y.eq(z, m(y)). \tag{124}$$

note that this is well-formed as

$$eq(z, m(y)) : (z : Z, y : Y) \rightarrow \Omega. \tag{125}$$

Further

$$char_m(m)(y2 : Y) = \exists! y.eq(m(y2), m(y)). \tag{126}$$

However, $monic(m)$ implies $\exists! y.eq(m(y2), m(y)) = true$ and

$$char_m(m) = true. \tag{127}$$

By the subtype axiom $char_m(m) = true$ implies

$$fac_{m,char_m} : Y \rightarrow \{x : X|char_m(x) = true\} \tag{128}$$

The subtypes of a type X and their inclusion maps are canonical in the sense that for any monic $m : A \to X$ the map $fac_{m,char_m} : A \to \{x : X|char_m(x) = true\}$.

The Monic axiom says that any monic $m : Y \to X$ has a characteristic map $char_m : X \to \Omega$ and has an inverse map $m^{-1} : \{x : X|char_m(x) = true\} \to Y$ which is an isomorphism. The subtype axiom applied to the characteristic map $char_m$ provides both the subtype and a map $fac_{m,char_m} : Y \to \{x : X|char_m(x) = true\}$.

Axiom 3.6 (Monics). For $m : Y \to X$, $monic(m)$, $p : X \to \Omega$, the type of m^{-1} is

$$m^{-1} : \{x : X|char_m(x) = true\} \to Y \tag{129}$$

The maps satisfy the following equations:

$$p = char_{incl_p} \tag{130}$$

$$m^{-1}(fac_{m,char_m}) = id_Y \tag{131}$$

$$fac_{m,char_m}(m^{-1}) = id_{\{x:X|char_m(x)=true\}} \tag{132}$$

3.7.1. *Functional relations and maps* Description Logic and modelling languages in the UML family including SysML make use of binary relations (called roles in DL and associations in UML) within models. The amino acid axiom set only uses maps. In type and topos theory (Lambek and Scott 1980), and in Algos a functional property defines a unique map with the same domain and range types as the functional property. As a result relations with finite multiplicity can always be replaced by maps. To obtain the map one uses the functional relation to define an existentially quantified characteristic map where the existentially quantified variable corresponds to the map value for an element in its domain. In the Algos version of the proof the monic classification axiom enables the explicit definition for the description map. Bertrand Russell used an axiom to provide a description operator within his type theory, hence the name description map.

A map which represents the construction of forming a singleton set is defined as:

Definition 3.6. The map $sing : Y \to Pow(Y)$ is defined as

$$sing = eq^*(x : Y) \to Pow(Y) \tag{133}$$

where $eq : (x : Y, y : Y) \to \Omega$.

The singleton map is a monic and is used in the definition of functional relations and the proof that a functional relation can be replaced by a map.

Lemma 3.3. *sing* is monic

Proof. Assume $f : Z \to Y, g : Z \to Y, sing(g) = sing(f)$. Then

$$f \in sing(f) \Rightarrow f \in sing(g) \Rightarrow f = g. \tag{134}$$

Thus *sing* is monic. □

Recall a binary map $r : (x : X, y : Y) \to \Omega$ determines a binary relation

$$R = \{< x, y >: (X, Y)|r(x, y) = true\}. \tag{135}$$

We use the notation $R :: Relation(X,Y)$ for a relation.

Definition 3.7. A binary formula $r : (x : X, y : Y) \to \Omega$ and its relation R are called *Functional* if

$$R = \{< x, y >: (x : X, y : Y) | \exists ! y.eq(^*(x), sing(y)) = true\}. \tag{136}$$

equivalently

$$\exists ! y.eq(r^*(x), sing(y)) = true_X. \tag{137}$$

Lemma 3.4. $Functional(r)$ implies

$$char_{sing}(r^*)(x : X) = \exists ! y.eq(r^*(x, sing(y))) \tag{138}$$

For any map $f : X \to Y$, the type

$$|f| = \{< x, y >: (X, Y) | eq(y, f(x)) = true\} \tag{139}$$

is called the graph of f. The relation $|f|$ is functional.

Lemma 3.5. $Functional(|f|)$.

Theorem 3.2 (Description Operator). For a functional relation $R :: Relation(X,Y)$ the map τ_r is defined by

$$\tau_r = sing^{-1}(fac_{r^*,sing}) : X \to Y \tag{140}$$

and satisfies

$$r(x, \tau_r(x)) = true. \tag{141}$$

Proof. The map *sing* is monic and $char_{sing}(P^*) = true$. Thus the Algos subobject classification axiom provide the factoring map $fac_{P^*,sing}$ which satisfies $r^*(fac_{r^*,sing}) = true$. For $r(x : X, y : Y) \to \Omega$, $Functional(r)$ implies

$$q(r^*(x)) = \exists ! y.eq(r^*(x), sing(y)) = true \tag{142}$$

By the subclass axiom

$$r^* = fac_{r^*,q}(incl_q) \tag{143}$$

where

$$fac_{r^*,q} : X \to \{z : Pow(Y) | \exists ! y.eq(r^*(x), sing(y)) = true\} \tag{144}$$

From the monic classification

$$sing^{-1} : \{z : Pow(Y) | \exists ! y.eq(r^*(x), sing(y)) = true\} \to Y. \tag{145}$$

Let

$$\tau_r = sing^{-1}(fac_{r^*,sing}) \tag{146}$$

and

$$r(x, \tau_P(x)) = true. \tag{147}$$

□

The correspondence between functional relations and maps is the justification for representing an association with multiplicity k with k maps.

3.8. *Exponential Types and application maps*

The exponential type construction provides a type Y^X for two types X and Y and provides a correspondence between a map $f : (x : X, w : W) \to Y$ and a map $\lambda x.f : W \to Y^X$. If $f : (x : X, w : W) \to Y$ then $\lambda x.f$ is a map with type

$$\lambda x.f : W \to Y^X \tag{148}$$

In Algos the "lambda variable" x is a projection map with type

$$x : (X, W) \to X. \tag{149}$$

The map constant [ ] written in infix notation, is called apply . The map has type

$$-[-] : (Y^X, X) \to Y \tag{150}$$

If

$$f : W \to Y^X, t : W \to X \tag{151}$$

then

$$f[t] : W \to Y. \tag{152}$$

This construction enables the representation of maps which can have arguments. This construction was not used in the amino acid example, but is used in more complex examples. Applicative maps, i.e., maps with explicit arguments occur frequently in engineering modelling. Lambda terms are comparable to SysML operations. The corresponding SysML notation for a map with arguments is $f(x : X) : Y$ in place of $f : Y^X$.

Axiom 3.7. For $f : (x : X, w : W) \to Y$, $t : W \to X$,

$$\lambda x.f[t] = f(t, w) = f(x := t) \tag{153}$$

and

$$\lambda x.f[x] = f. \tag{154}$$

It follows that the map $\lambda y.y$ for $(y : Y)$ is the identity map on Y^Y. As noted in the discussion of products projection maps are variables in the sense that usual substitution rules hold.

3.9. *The Syntactic Topos*

The logical soundness for an Algos axiom set is a consequence of the fact that an Algos axiom generates a topos. A topos is defined as a Cartesian closed category (ccc) with a subobject classifier and a natural numbers type. As noted in (Lambek and Scott 1980) a topos may be characterized as a Cartesian category (products) with the power type construction and a natural numbers object together with axioms for the language constructions. The term language of Algos is closed under the topos map and type constructions. The Algos axioms and the topos axioms are similar. However, the Algos axiom for subtypes is more restrictive than the corresponding topos axiom. An Algos axiom set $\mathbf{A}$ generates a topos, $\mathbf{T}(\mathbf{A})$, called the syntactic topos in which any Algos formula derivable from $\mathbf{A}$ is true in $\mathbf{T}(\mathbf{A})$. The syntactic topos enjoys the property not satisfied by an arbitrary topos that subtypes are canonical subobjects.

Definition 3.8. For an Algos axiom set $\mathbf{A}$, let $\mathbf{T}(\mathbf{A})$ be the term language of $\mathbf{A}$ with the equivalence relation defined below.

The types of $\mathbf{T}(\mathbf{A})$ are the same as the types generated by the Algos language. The maps of $\mathbf{T}(\mathbf{A})$ are the equivalence classes of Algos maps defined by the equality predicate. The equality predicate defines an equivalence relation on the map terms in that for maps $f, g : X \to Y$ with the same typing the equality predicate is reflexive, symmetric, and transitive, and satisfies substitution rules.

The traditional statement of subobject classification is:

Definition 3.9 (Topos subobject classification).

$$\forall p : A \to \Omega, incl_p : Ker_p \to A \tag{155}$$

$$\tag{156}$$

$$\forall m : B \to A, monic(m), \exists !.char_m : B \to \Omega \tag{157}$$

$$p = char_{incl_p} \tag{158}$$

$$Ker_{char_m} \simeq B. \tag{159}$$

Note that $incl_h$ is a first order function whose argument is a maps p to truth valued maps and whose value is an inclusion map. Ker_h is a function whose argument is a map and whose value is a type. In Algos notation

$$Ker_h = \{x : A | h(x) = true\}. \tag{160}$$

Theorem 3.3. $\mathbf{T}(\mathbf{A})$ is a topos and a model of $\mathbf{A}$.

Proof. $\mathbf{T}(\mathbf{A})$ is a cartesian category, with equality, and power types. What remains to be shown is that $\mathbf{T}(\mathbf{A})$ satisfies the topos subobject classification axiom. Assume The Algos subobject axiom. Since $incl_p$ is monic and $p(incl_p) = true$

$$p = char_{incl_p} \tag{161}$$

Since m is monic For a monic $m : Y \to X$, the map $m^{-1}(fac_{m,char_m})$ is an isomorphism. □

Further, $\mathbf{A}$ has canonical subobjects in the sense that an explicit isomorphism is given between Ker_{char_m} and the domain of m.

An interpretation of an axiom set maps the term language of the signature $\Sigma_{\mathbf{A}}$ of the axiom set into the maps and types of the topos preserving the term constructions. A model is an interpretation which satisfies the axiom set. Any formula derivable from the axiom set is true in any interpretation of the axiom set which satisfied the axioms. The syntactic topos $\mathbf{T}(\mathbf{A})$ is an initial model of $\mathbf{A}$.

3.10. *Notes*

The theory of 2-amino acids and water generate simple topos theories. Both theories have few axioms beyond the signature and its typing relations. Both theories contain a constant *Thing*. However, the axioms make use of the properties of subtypes of *Thing* including the orthogonality relations. Products are used in the construction of models of the axioms. The image construction is used in connection with the "commutative diagram" equations. Section 4 abstracts the properties of these axiom sets to verify the decidability of consistency. In Section 4 properties of structural description models are abstracted to achieve decidability of consistency results.

Section 5 introduces a more representative class of engineering models. The Algos/topos language constructions needed to provide axioms for these examples are discussed. A simple engineering model with behaviour described by both a state machine and physics laws is used to illustrate the axiomatization approach. The example provides evidence that the Algos framework can place these more general engineering models in an axiomatic framework.

4. Structure Descriptions

This Section singles out a restricted class of Algos axioms sets, called Description Axiom Sets, which are sufficient for the 2-amino acid, water descriptions, and other structural examples. They are called Description Axiom Sets as their language is a very general Description language in the sense used in Description Logic (Baader *et al.* 2007). The restriction on Algos axiom sets is obtained by introducing a type constant *Thing* to the signature of the axiom set and using Algos term constructions restricted to subtype constructions for *Thing*, the product $(Thing, Thing)$ and $Pow(Thing)$. Individuals are maps from *One* to *Thing*. The restricted language can be viewed as a language whose terms are individuals, classes (concepts), relations (roles). Description formulae are Horn clause formulae in predicates which include the Algos predicates of $\in$, $\sqsubseteq$, $=$, and various defined predicates in term constructions using variables for individuals, classes, and relations and Horn clauses.

This approach to Description Logic has advantages over the traditional presentation of DLs as terms in the language generated by a grammar where the semantics is obtained solely in terms of the DL-model theory. The Algos axiomatic semantics restricted to Description Axiom Sets provides these axioms sets with an axiomatic semantics. The axiomatic semantics is consistent with the DL-model theoretic semantics, and of course the Topos semantics. The expressiveness of Descripiton Axiom sets, including the use of variables which is absent from DLs, is used in the characterization of the models of the structure examples. The Algos theory generated by a Description Axiom Set is again a topos and includes, for example, the internal logic. However, the external Horn clause Description Formulae correspond to a fragment of the internal logic. This correspondence can be used to characterize the Description formulae implied by a Description Axiom set (Subsection 4.1). This approach offers techniques not available in DL, but useful for describing axiom sets to constrain the possible models. For example, a Description Axiom Set corresponds to a Logic Program in the sense of a Horn clause in a single sorted first order logic. This correspondence can be used to obtain decidability results with further restrictions on the axiom sets.

Alternatively one can present a Description Language as the language generated from a signature with the three sorts,*individual*, *class*, and *relation*. This Description Language has variables which traditional Description Logics do not have. Such a Description Language is directly embeddable into an Algos theory. The embedding establishes consistency of the language. The fact that functional relations correspond to maps allows the language to be extended with maps whose domain and range are classes. With the embedding the Description Language acquires an axiomatic semantics. This will be left as an exercise.

In this section we again write composition in left-to-right "path" ordering with dots. We start by restricting Algos axiom sets to those generated from an Algos signature which has a type constant $Thing$ and where the terms, including variables, in the Horn clause formulae are restricted to individuals,i.e. maps from One to $Thing$, Classes, i.e. subtypes of $Thing$, relations, i.e. subtypes of a product of two classes, and maps whose domain and range are classes. From Section 3 classes and relations are closed under the usual Description Logic class and relation type constructions. Hence we have a Description Logic with variables.

Subsection 4.2 further restricts Description Axiom Sets to axiom sets called *Structure Diagrams*. A Structure Diagram is an abstraction of the amino acid example and other examples which have component decompositions and connections between components. In the examples the maps in the signature of the axiom set divide into two classes, part maps and connection maps. The signature of the axiom set form a graph of class nodes and map edges for which the part maps provide the tree like structure. As the graph is part of the signature the conditions for a Structure Diagram can be checked syntactically. The definition uses three finite types $Part$, $Conn$, and $Node$, for part and connection maps, and type symbols. The type $PartPath$ is used for the composition of part maps. The Structure Diagram axioms imply that part paths are finite. Decidability results are given for Structure Diagrams which include the amino acid axiom set. The restrictions on Algos axiom sets which ensure decidability of consistency are based on an acyclic condition for part maps. The connection axioms define map equations each of which can be represented by a unary predicate. These conditions enables the restricted axiom sets to be represented as monadic Ackermann formulae which is known to be a decidable class (Ackermann 1954).

Subsection 4.3 discusses the relationship between Algos Description Axiom sets and DL generalizations which have been used to represent structures.

4.1. *Description Axiom Sets*

Definition 4.1. A Description signature is an Algos signature with a constant type

Thing where the terms satisfy the classification predicates below.

$$t :: Invividual \equiv t : One \to Thing \tag{162}$$

$$C :: Class \equiv C \sqsubseteq Thing \tag{163}$$

$$P :: Relation(A, B) \equiv A, B \sqsubseteq Thing, P \sqsubseteq (A, B). \tag{164}$$

$$f :: Map(A, B) \equiv A, B \sqsubseteq Thing, f : A \to B. \tag{165}$$

The symbols in the signature are refered to as class, relation, map, and individual symbols. The Description language of a signature consists of Algos terms constructed from the individual, map, class, and relation symbols in the signature, with standard DL class and role constructions. From Section 2 the type terms generated from a collection of class symbols is closed under the Description Logic class constructions, including not only the logical operations, but the existential and universal class constructions.

Description Axiom Sets are characterized in terms of their corresponding internal truth valued maps. Recall, for example, that a subtype A of *Thing* corresponds to a map $char_A : Thing \to \Omega$. This correspondence enables the characterization of the theory generated by a Description Axiom Set in terms of the internal logic. By using the power type construction even the logical operators in a Description Axiom Set correspond to internal truth valued maps. This characterization in terms of the internal logic enables a correspondence between the Horn Clause Description Axioms in the individual, class, and relation variables and a set of first order single sorted Horn Clauses. These formulae can be though of as the Logic Program Determined by the Description Axiom Set. Any description formula is equivalent to an internal formula defined on the product type of classes and the power type of *Thing*. This characterization enables the truth of a Description Formulae to be equivalent to the truth of an internal formulae.

With the Description Axiom Set restrictions the axiom sets can be viewed as belonging to a language generated from a signature with three sorts, individuals, classes, and relations, with constant symbols for *Thing*. As the classes are subtypes of a type constant *Thing* the membership predicate $t \in A$ is defined for individuals and classes. The Description axiom sets are constructed from atomic formulae defined using predicates for membership, inclusion, equality, and well as auxiliary predicates for relation properties such as being functional. The Algos model theoretic semantics simplifies to the DL model theoretic semantics in that all of the language constructions are interpreted with respect to a domain Δ which interprets *Thing*. The resulting Description Logic is more general than are traditional Description Languages.

Definition 4.2. A Description Formula is a Horn clause in individual, class, relation, and map variables and with predicates for $\in, =, \sqsubseteq$, and $function$, and function symbols for *Domain* and *Range*. The Horn clauses may contain predicates in addition to the Algos primitive predicates. A Description Axiom Set is a collection of Description formulae. The theory generated by a Description axiom set is the Algos theory generated by the Description formulae and the Algos axioms.

The Horn clause variables are written in boldface:

$$\mathbf{t} :: Individual \tag{166}$$

$$\mathbf{C} :: Class \tag{167}$$
$$\mathbf{P} :: Relation(A, B) \tag{168}$$
$$\mathbf{f} :: Map(A, B) \tag{169}$$

The Algos theory of a Description Axiom Set contains the characteristic maps and inclusion maps for classes and relations as the symbols are Algos subtypes and the terms they generate are subtypes as well. As an Algos Axiom Set the predicates such as a relation being functional and a map being monic not only apply, but are expressible as Description formulae. A relation P is functional if

$$< x, y1 >\in P, < x, y2 >\in P \Rightarrow y1 = y2. \tag{170}$$

A map $f :: Map(A, B)$ is monic if

$$x1 :: Thing, x2 :: Thing, x1.f = x2.f \Rightarrow x1 = x2. \tag{171}$$

Any functional relation in an Algos Axiom Set determines a description operator which is a map with the same domain and range as the relation. More generally relations with finite multiplicity k can be replaced by k description operators. As a result we can extend Description Axioms sets which have functional relations with maps.

The first result is that a Description formula is equivalent to the equality of an internal formula with *true*. The application predicates can be eliminated to produce Horn clauses that use only Algos predicates. As a result a formula ψ is derivable for the axioms if its characteristic map ψ_{ch} is equal to *true* in the theory generated by the axioms.

Lemma 4.1. Any description predicate is equivalent to the truth of an internal formula whose domain is the Cartesian product of the types which correspond to the classification variables.

Proof. The individual, class, and relation variables whose atomic formulae use the Algos predicate symbols $=$, $\in$, $\sqsubseteq$ are equivalent to the truth of a characteristic map which uses the internal maps, eq,ϵ, and $\sqsubseteq$. The classification variables correspond to projection maps. The correspondence is defined by:

$$\begin{aligned}\mathbf{t1} = \mathbf{t2} \equiv \\ eq(t1 : Thing, t2 : Thing) = true\end{aligned} \tag{172}$$

$$\begin{aligned}\mathbf{x} \in \mathbf{A} \equiv \\ \epsilon : (x : Thing, A : Pow(Thing)) \\ = true\end{aligned} \tag{173}$$

$$\begin{aligned}\mathbf{A} \sqsubseteq \mathbf{B} \equiv \\ \sqsubseteq : (A : Pow(Thing), \\ B : Pow(Thing)) \\ = true\end{aligned} \tag{174}$$

□

To represent variables for maps one uses the exponential type B^A and variables whose range is an exponential type. To simplify the presentation exponentials are not introduced. However, maps can be replaced by their graphs which are relations. A map $f : A \to B$ is replaced by its graph $|f| :: \mathit{Relation}(A, B)$. In the amino acid example a relation $\mathit{hasPart} :: \mathit{Relation}(\mathbf{R}, H)$ was introduced. This is replaced by

$$\mathbf{R} \sqsubseteq \exists \mathit{hasPart}.H \tag{175}$$

corresponds to the characteristic map

$$\mathbf{R} \sqsubseteq \exists \mathit{hasPart}.H : (R : \mathit{Pow}(\mathit{Thing})) \to \Omega. \tag{176}$$

To prove the following theory one notes that the external logical operators for conjunction, implication, and negation corresponds to internal logical operators.

Theorem 4.1. A Horn clause formula $\psi(x1, \ldots, xn)$ where each xi is typed by a classification predicate defines a characteristic map $\psi_{ch} : (x1 : X1, \ldots, xn : Xn) \to \Omega$ for which

$$\begin{gathered}\psi(x1, \ldots, xn) \equiv \\ \psi_{ch} : (x1 : X1, \ldots, xn : Xn) = \mathit{true}\end{gathered} \tag{177}$$

using the correspondence between external formulae and truth-valued maps.

Proof. An external $\psi(x1, \ldots, xn)$ decomposes into an implication that is a conjunction of literals whose consequent is a single literal. While not all of the variables in ψ occur in the component formulae, for simplicity we write the component formulae as containing all of the variables. Thus, for any two atoms $\psi_1(x1, \ldots, xn)$ and $\psi_1(x1, \ldots, xn)$ the conjunction

$$\begin{aligned}\psi_1(x1, \ldots, xn) \sqcap \psi_2(x1, \ldots, xn) = \\ (\psi_1 \sqcap \psi_2)(x1, \ldots, xn)\end{aligned} \tag{178}$$

and so

$$\psi = (\psi_1 \sqcap \psi_2)(x1, \ldots, xn) \tag{179}$$

corresponds to

$$\psi_{ch}(\psi_{ch1} \sqcap \psi_{ch2}) : (x1 : X1, \ldots, xn : Xn) \to \Omega \tag{180}$$

The other cases are similar. □

Any Description formula for a Description signature is equivalent to the equality of an internal formula with true in the Algos theory of the axiom set.

Theorem 4.2. If a Description formula is derivable from a Description Axiom Set then its internal counterpart is equal *true*.

4.1.1. *Model Theory for Description Axiom Sets* Since a Description Axiom Set is an Algos axiom set topos model theory is defined for Description Axiom Sets. The restrictions on Description Axiom Sets imply that the model theory can be restricted to interpretations with a domain Δ which interprets $Thing$. The language constructions are confined to the type $Thing$, the product $(Thing, Thing)$ and the power type $Pow(Thing)$.

As a result equality preserving interpretations preserve derivability. An interpretation is a domain Δ and mapping of the class and relation symbols of the signature to subsets and sub-relations of Δ. A *structure* is an axiom set, and an interpretation of the axiom set in a domain Δ. The domain of a structure may be, but doesn't have to be a set. In general Algos model theory is defined with respect to a category of toposes. The Algos constructor functions and predicates map to class and relation operations. A structure *models* an axiom set if all of the formulae in the axiom set are satisfied in the structure and the mapping preserves logical equality. As the formulae in the theory of an axioms set are equivalent to equations they are true in a structure which models the axiom set.

Theorem 4.3. If a structure models a Description axiom set then any description formula in the theory of the axiom set is true in the structure.

4.1.2. *Logic Program for a Description Axiom Set* Description Logics are recognized as being equivalent to fragments of a single sorted first order logic, where classes correspond to unary predictes and relation correspond to binary predicates. For Algos Description Axiom Sets an explicit correspondence is defined as follows. We use the same notation for the unary or binary predicate defined by a subtype. For $x, y :: Thing$, $A, B :: Class$, and $P :: Relation(A, B)$ the membership predicate $\in$ is defined as:

$$A(x) \equiv x \in A \tag{181}$$

$$P(x, y) \equiv < x, y > \in P \tag{182}$$

Lemma 4.2. The correspondence preserves the logical operations:

$$A(x) \wedge A(y) \equiv x \in A \sqcap B \tag{183}$$

$$A(x) \vee A(y) \equiv x \in A \sqcup B \tag{184}$$

$$\forall y.P(x, y) \Rightarrow B(y) \equiv x \in \forall P.B \tag{185}$$

$$\exists y.P(x, y) \wedge B(y) \equiv x \in \exists P.B \tag{186}$$

$$\neg A(x) \equiv \neg A \tag{187}$$

The correspondence can be extended to maps by introducing Skolem functions for the maps. This correspondence will be used in the next section where Description Axiom Sets are further restricted to model the properties of structural descriptions such as the 2-amino acid class.

4.2. *Structure Diagrams*

The axioms below for a Structure Diagram abstract the part and connection properties of the 2-amino acid and other examples.

Definition 4.3. A Description Axiom Set is a Structure Diagram whose signature contains a set of class symbols called nodes, a class symbol $Start$, the map symbols contains two finite disjoint subsets *Part* and *Conn*. Each class which is the domain or range of a part map symbol is a node. A part path is a finite composition $p = p1 \dots pn$ where pi are part maps. A part path $p = p1 \dots pn$ is anti-cyclic if $Domain(pi) \neq Range(pj)$ for any i, j. The notation $p :: Part$, $p :: PartPath$, $r :: Conn$ and $A :: Node$ are used to indicate that a map is in one of these sets. The axioms are:

$$A :: Node \Rightarrow A \sqsubseteq Thing \tag{188}$$

$$p :: Part \Rightarrow Monic(p) \tag{189}$$

$$p :: Part \Rightarrow Range(p) \neq Start \tag{190}$$

$$p :: Part, Range(p) \neq S \Rightarrow S = Start \tag{191}$$

$$p :: PartPath \Rightarrow Anti - cyclic(p) \tag{192}$$

$$p, q :: PartPath, p \neq q, \Rightarrow p \perp q \tag{193}$$

Each part and connection map is monic. Each path connection r is of the form $r : Im(p) \to Im(q)$ where p and q are part paths.

$$r.p = q. \tag{194}$$

The first two axioms say that *Start* satisfies the start property that it has no part maps with it as range and the second axiom says that *Start* is the only node with that property. These conditions can be verified for an axiom set as one can show that only a finite number of maps have to be checked. This pattern is sufficient to establish decidability of consistency and if the axiom set is consistent to construct a minimal model and show that all minimal models are structurally isomorphic. The Herbrand construction can be used to construct models.

For a Structure Diagram one can construct the corresponding BDD and IBD graphs. *Start* is a root for the BDD. In the BDD if the nodes are replaced by the image types of the part maps then the BDD is a tree. The BDD consists of the nodes in the signature with the part arrows as edges. The full Structure Diagram is a directed graph is not a tree as the atomic nodes may occur as ranges of multiple part arrows. The IBD consists of the images of the part paths together with the part paths and connection maps.

Lemma 4.3. The amino acid axiom set is a Structure Diagram.

Proof. The amino acid axiom set has a Description signature. *AminoAcid* is a start symbol. All of the arrows are monics. The anti-cyclic condition is satisfied. The part arrow orthogonality condition is taken as an axiom, and all of the connection arrows have the prescribed form. □

Lemma 4.4. For a Structure Diagram each class node is reachable by a unique part path and the number of part paths is finite.

Proof. From the anticyclic condition all of the domain and range nodes occurring in the arrows of a part path are distinct. Hence any part path has finite length.The length of a part path is bounded by the number of nodes of the graph. For the unique reachability of each node by a part path, note that if p and q are part paths with $p = p_n, \dots, p_1$ and

$q = q_m, \ldots, q_1$ and they terminate at the same node then $Im(p) = Im(q)$. Thus, it is not the case that $p \perp q$. This implies $p = q$ provided $Im(p)$ has members. □

When axiom sets are restricted to be Structure Diagrams they can be mapped into the class of monadic Ackermann formulae. Each formulae is not only a Horn clause but has a single universally quantified variable. For each map $f : A \to B$ in the signature a first order Skolem function is introduced. The same notation is used for the map and the Skolem function. Thus for each monic f the Skolem function f^{-1} is introduced. The correspondence also satisfies:

Lemma 4.5. Characterization of Structure Diagrams in terms of monadic Horn Clauses.

$$P :: Relation(A, B), < x, y > inP \Rightarrow x \in A, y \in B \tag{195}$$

$$f :: Map(A, B) \mapsto A(x) \Rightarrow B(x.f) \tag{196}$$

$$f.p = q \mapsto Eq_{f,p,q}(x) \tag{197}$$

$$A \perp B \mapsto A(x) \wedge B(x) = false \tag{198}$$

$$p \perp q \mapsto x.p \neq x.q \tag{199}$$

Proof. The proof follows from the definition of the predicates and the fact that there are only a finite number of part paths and a finite number of connection equations. In each of these cases a binary predicate is replaced by a finite number of unary predicates. □

Theorem 4.4. For a Structure Diagram its Logic Program is equivalent to monadic Ackermann formulae. Thus, the consistency of a Structure Diagram is decidable.

Proof. The decidability follows from the equivalence with monadic Ackermann formulae. □

For a Structure Diagram one can construct its term model.

Definition 4.4. For a structure diagram $\mathbf{G}$ the directed graph $\mathbf{G}[s]$ is defined by adjoining an individual s with $s : Start$. The nodes are the type $\{p_i\}$ where $p_0 = s$, and the p_i are the part paths. The edges are the ordered pairs $< f.p_i, p_j >$ for each arrow f whose domain is the range of $\{p_i\}$ and whose range is the domain of p_j.

A realization of a Structure Diagram is one or more structures which satisfy the axioms. A realization can be constructed which satisfies the axioms by adjoining an individual s with $am \in Start$. By composing a map $f : A \to B$ with individual $a : A$ one obtains an individual $a.f$ in B.

Theorem 4.5. For a Structure Diagram $\mathbf{G}$ then $\mathbf{G}[s]$ is a minimal model.

Proof. For each node A in $\mathbf{G}$ there is a unique part path p_A with $p_A : Start \to A$. The correspondence defined by:

$$Start \mapsto s \tag{200}$$

$$A \mapsto p_A \tag{201}$$

$$p : A \to B \mapsto < s.p_A, s.p_A.p > \tag{202}$$

$$r : D \to E \mapsto < s.p_D.r, s.p_E > \tag{203}$$

is a structure which models $\mathbf{G}$. If $(\mathbf{I}, \Delta)$ is a structure which models $\mathbf{G}$ then for $a \in Start^{\mathbf{I}}$ the correspondence defined by mapping s to a defines an inclusion of $\mathbf{G}[s]$ into $\mathbf{G}^{I}[a]$. □

The 2-amino acid axiom set is a Structure Description with $Start = AminoAcid$. A realization of the 2-amino acid axiom set is obtained by adding an individual a which is a member of the class *AminoAcid* representing the class of amino acids and iterating the map compositions one obtains the graph whose nodes are

$$\begin{aligned} N = \{am, p_1.am, p_2.am, p_2.am,\\ q2.p_2.am, q3.p_2.am,\\ p_3.am, r1.p_3.am,\\ r2.p_3.am, r3.p_3.am, r4.p_3.am,\\ p_4.am, p_5.am\} \end{aligned} \tag{204}$$

and whose edges are the part edges in the BDD and the connections between the part components in the IBD is a realization of the amino acid axiom set. The nodes are distinct. The connection terms do not add any new nodes.

Theorem 4.6. In a structure $(\mathbf{I}, \Delta)$ which models a Structure Diagram an element of $Start^{\mathbf{I}}$ generates a directed graph which is isomorphic as a graph to the canonical graph of the term model.

Proof. Using the notation that p_A is the unique part path from $Start$ to a node A for any two distinct elements $a1, a2$ in $Start$ one has $a1.p_A \neq a2.p_A$. This implies the two realizations are disjoint. □

4.2.1. *Generalizations of Structure Diagrams* The 2-amino acid axiom set is a schema as it contains the side chain condition. For any ground terms that unify with the condition

$$\mathbf{R} \sqsubseteq \exists hasPart(\mathbf{R}, H) \tag{205}$$

one simply substitutes the ground terms and replace *hasPart* with a part map. The result will satisfy the Structure Description property. A 2-amino acid description is a set of ground terms which unify with the axiom set. The *hasPart* relation is replaced with a map. While the axioms involve class and part map variables the only language constructions are map compositions which are finite.

An axiom set may contain a Structure Diagram and have additional axioms for which consistency is still decidable. For example, the axioms may contain "propagation axioms" which can be used for fault detection and disease diagnosis.

4.3. *Relationship with other DL formalisms*

Both the first order logic (Krdzavac *et al.*2008) and the Description Logic formalisms (Baader *et al.* 2007) have been candidate formalisms for describing classes of structures such as molecules. In Description Logic one uses classes H_2O, $Oxygen$ and $Hydrogen$ for types of molecules, and binary relations such as $hasPart$ to identify the kind of relation. In first order logic one uses unary predicates as $Oxygen(x)$ to identify the kind of components and binary predicates $hasPart(x, y)$ to identify the kind of relation. For example the class H_2O has the property that

$$Range(hasPart) \sqcap H_2O = Null \tag{206}$$

which says that the intersection of $hasPart$ with H_2O is empty which is expressible in DL. Equivalently in first order logic one can say that

$$H_2O(x) \Rightarrow hasPart(y, x) = false \tag{207}$$

Both of these formalism have difficulty expressing conditions which imply any water molecule only has three component atoms, that the oxygen atom of a water molecule is bonded to the hydrogen atom which is the component of that water molecule, and that the atoms are not connected to any other molecules. Extensions of DL such as DGDL (Motik *et al.* 2008) and DGLP (Magka *et al. 2012*) have been introduced to address these issues. However, many properties of interest in constraining realizations are higher order with respect to these formalism and can not be directly expressed in these formalisms. For example, formula such as

$$Root(X) \equiv Range(hasPart) \sqcap X = Null \tag{208}$$

is higher order in Description Logic based formalisms as X is a class variable.

As noted in (Magka *et al. 2012*) 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. This limitation of DLs to represent cycles has been remedied (partially) by the extension of DLs with Description Graphs and rules (DGDL) (Motik *et al.* 2008). A Description Graph represents structures by means of a directed labelled graph. Figure 1 represents a description graph. The Description Graph part of a DGDL ontology is separated from the DL axiom set which complicates understanding and reasoning. DGDL axiom sets can not preclude additional components such as an oxygen atom in the case of cyclobutane. As a result, reasoning cannot give a positive answer to the question of whether cyclobutane is a hydrocarbon.

A logic programming formalism (Magka *et al. 2012*) called Description Graph Logic Programs (DGLPs) has been suggested as an approach to remedy deficiencies of DGDL. However, graph theoretic properties when expressed in a single sorted Logic Program such as DGLP are higher order which makes constraining axioms to produce realizations with specific graphical properties difficult. 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.

Algos by using a multi-sorted Logic Programming 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 (Dumontier 2007; Hastings *et al.* 2010 ;

Graves and Horrocks 2008). If one prefers to only use the concept and role constructions from DL without maps and other Algos constructions, then a grammar may be defined for a DL with the additions found in the Algos Description language with the Algos predicates such as $x \in A$ and $< x, y > \in P$ and have a DL with extended language constructions, variables, and an axiomatic semantics, as well as the standard model theoretic semantics.

5. Composite Structures

This section outlines the Algos axiomatic approach to models which specify component structure and behaviour. Behaviour is change with respect to space, time, or change with respect the states of a state machine. The Algos approach enables the integration of behaviour with structural decomposition. An aircraft system, for example, consists of components for the air frame, propulsion, navigation, and hundreds of thousands of other subsystems and parts. Many of these components have sensors and effectors. The sensors perceive change and the effectors respond to perceived change. The behaviour of an aircraft system is a composite of the behaviour of individual components, the interactions with its operating environment, and the physical laws which effect the results of actions performed by system components. Engineering models which have this kind of structure are sometimes called composite structure models. The Algos axiom sets which represent them are called Composite Structure Models.

The first subsection describes the Algos language constructions used in representing composite structures. The second subsection gives background on the engineering modelling perspective. The third subsection presents an exampleof a vehicle in a test environment. The example is first presented graphically and is then embedded in an Algos axiom set presented in linear syntax. The fourth subsection discusses inference and simulation in the context of Composite Structure Models, followed by notes on this topic.

Design analysis often consists of determining whether a system can achieve an outcome under specified preconditions. For example, one may want to determine if an aircraft system consisting of the aircraft and its crew can recognize and identify an object under preconditions that include distance to the object, atmosphere, flight motion, and many other variables. Whether the aircraft system can achieve a successful outcome depends on system's subsystem behaviour, as it is influenced by the environment. Analysis generally involves simulation to rule out non-feasibility and suggest what might be feasible. For Composite Structure Models a simulation is, as we shall see, a valid interpretation of the axiom set. Determination of whether the system being modelled can meet an objective generally involves inference. In many cases this inference depends primarily on whether, for example the aircraft can maintain a sufficiently steady state for a long enough period of time to make an identification, or whether its sensors can recognize an obstacle in the flight path in time for the aircraft to avoid it. In many of these situations the behaviour of the subsystems is well known. The reasoning primarily concerns composition of actions effected by subsystem operations with time duration.

The challenges from the axiomatic perspective are: how can change be represented, how are actions that cause change initiated, and how are associative or causal relationships involving change propagated. The Algos axioms for an engineering model with behaviour use a terminal object, *One*, to specify the state structure. Maps change with respect to the state structure. For example if the change is with respect to time then *One* has the structure of a time type, such as N for natural number time or R for real number time. If change is with respect to state machine states, then *One* has the structure of a product of finite type, which represents the machine states, with N. The Algos axiom sets for a model with behaviour use first order state variables in Horn clauses. The use of these state variables is analogous to the use of variables for classes and relations for structure descriptions. The specific form of state variables enables general map variables in Algos formulae to be first order theories with signatures restricted to the types of the state variables.

SysML and other modelling languages have good syntax for behaviour constructions such as state machines. The graphic syntax offers a practical way to develop large scale complex models with hierarchies of components. Some of the components may have behaviour constructions such as a state machine. While the informal semantics of these languages are well developed they do not in general have a formal semantics. For some classes of engineering models the model development tools can compile a model to executable code which can be integrated with physics and other code to produce high precision simulations. There is interest in providing a formal semantics for engineering modelling languages as evidenced by a request for a proposal for a formal executable semantics from the Object Management Group (OMG) Standards organization.

Algos language constructions for behaviour modelling are similar to SysML. SysML can be used for axiom development in the Algos context. The informal notions of model executability can be made precise in terms of interpretations of the axiom sets. The Algos approach makes the connection between execution semantics and the model theory for the axioms. A valid interpretation of an axiom set with state change behaviour is the interpretation of a theory which contains state space variables. The interpretation specifies functions of these variables which provide assignments for all possible values of the variables. The model theory is an execution semantics.

The examples of composite structure in this section, which include a vehicle operating in an environment, use the full power of the Algos language. Behavioural modelling in Algos use axioms which imply that the terminal type has a state space structure. The concepts of part components used for the molecular models are extended to include other kinds of components typically found in composite structure models. The components of an instance of a type X, in addition to parts, include attributes which are maps from X to a data type, operations which have arguments, and state machines' which effect state change. Both the state and operation constructions use the exponential type construction and lambda-abstraction, as well as case statements which are defined in terms of the sum type construction. The sum type construction is defined within Algos. The representation given ensures that a model has a unique part decomposition and that attributes and operations can be uniquely associated with a part component of the model. To achieve this uniqueness extensive use of monic maps is made. The result of embedding composite structure models within a logic-based framework is the integration of logical inference with engineering modelling analysis including simulation.

5.1. *Algos Representation of behaviour*

This subsection outlines constructions definable in Algos which will be used to represent composite structure models. Composition will be written in left-to-right order. For example, a map $f : X \to Y$ and $a : Z \to X$ then the composition $a.f$ is the composition of f with a. For an individual $a :: X$ the component value $a.f$ will be an individual of type Y. Algos following topos theory provides an algebraic way to represent behaviour which implies that the terminal type *One* has non-trivial substructure. Axioms which describe the behaviour of a model are, as all application axiom sets, Horn clauses. These axiom sets contain first order variables for states. These map variables are restricted to projection maps whose range type is the state space type. The valid interpretations of the axiom set describe all possible valid paths indexed by time in the state space of the variables.

5.1.1. *Subobjects of One* The basis for representing behaviour is extending an axiom set so that the terminal type, *One* contains subobjects other than $Null$ which is a subtype of any type and *One* which is a subobject of itself. The structure of *One* is used to define map change. Subobjects of *One* are closed under intuitionistic operations as is the case for any type. A subobject $U \sqsubseteq One$ has an inclusion map $incl_U : U \to One$. Any individual $a :: X$, i.e., $a : One \to X$ can be composed with the inclusion map to obtain a map $U.a : U \to X$ which localizes or restricts a to U. For an individual $f :: X$ to be restricted the notation

$$f|U = incl_U.f \tag{209}$$

is used for the composition of f with the inclusion map.

There are several ways add subobjects to *One*. One way is to add internal truth valued maps of the form $u : One \to \Omega$. For each truth valued map u the subtype

$$U = \{x : One | x.u = true\} \tag{210}$$

and the inclusion map $incl_U : U \to One$ are included in the Algos Theory. Another way is to add a type axiom such as $One = \{on, off\}$. In this case the Algos theory has the singleton types, $\{on\}$ and $\{off\}$ as subtypes of *One* together with their inclusion maps. For an individual $a :: X$ and $t :: \{on, off\}$ we use the notation

$$a@t = a|\{t\} = incl_{\{t\}}.a \tag{211}$$

The type $\{on, off\}$ is the sum type which is written as $sum(incl_{on} : \{on\}, incl_{off}\{off\})$. The maps inclusion $incl_{on}$ and $incl_{off}$ are called *tags*. The map construction corresponding to the sum type is the *case* statement. For $f : \{on\} \to X$ and $g : \{off\} \to X$ the case statement $case(incl_{on} : f, incl_{off} : g)$ has type

$$case(incl_{on} : f, incl_{off} : g) sum(incl_{on} : f, incl_{off} : g) : \{on, off\} \to X. \tag{212}$$

As an Algos theory contains the natural number type N satisfying the Lawvere axioms it contains the map

$$0 : One \to N \tag{213}$$

and the k-fold successor maps

$$k : One \to N \tag{214}$$

For the situation where $One = \{on, off\}$

$$k = case(incl_{on} : k, incl_{off} : k) \tag{215}$$

there are other individuals of N. The case statement construction can be used to construct other maps. For example the map

$$f = case(incl_{on} : k, incl_{off} : 0) \tag{216}$$

which is the successor map k on $\{on\}$ and 0 on $\{off\}$. Note that for $s :: \{on, off\}$ the map $f@s$ defines a sequence of on-off states. If this map is included within an Algos axiom set we use the notation $\mathbf{s}$ for the first order variable and $\mathbf{f}@$ for the first order function. The use of finite types enables us to define state machines as maps. These state machines can be combined with space-time change.

5.1.2. *Linear discrete time* A special case of non-trivial *One* is linear discrete time. For linear discrete time we assume

$$One = N \tag{217}$$

For each $i \in N$, $\{i\}$ is a singleton type with inclusion map $inc_i : \{i\} \to One$ and characteristic map $char_i : One \to \Omega$. More generally we have characteristic maps defined for intervals, e.g., $char_{[i,\ldots,j]} : \{i, \ldots, j\} \to \Omega$. For an individual $f : One \to X$ we use the notation $f@i$ for $incl_i.f$. Recall we use the notation $k+1$ for the successor function. Let

$$count@0 = 0 \tag{218}$$

$$count@(k+1) = count@k + 1 \tag{219}$$

Thus *count* is the successor function $suc : One \to N$. We can define a counter which counts to some $k \in N$ as a map $k - count : One \to N$ by

$$n < k \Rightarrow k - count@n = n \tag{220}$$

$$n > k \Rightarrow k - count@n = 0 \tag{221}$$

We assume that corresponding to each positive integer k there is a map $k : One \to One$ which is constant, i.e., for any i and j

$$k@i = k@j. \tag{222}$$

For the natural number time the $i :: N$ can be corresponded with a first order variable which we write as $\mathbf{i}$. The maps *count* and $k - count$ correspond to first order functions of $\mathbf{i}$.

More generally, a map $f@(i)$ corresponds to a sequence of individuals with the range type of f. A valid interpretation of an Algos axiom set which contains state variables is a tuple of individuals in a topos which contains a domain for the state variable types and whose Ω is $\{true, false\}$. For simplicity we can assume that the model is within set theory. The first order interpretation for a model with behaviour need only represent the maps which are not constant as first order functions. By leaving the time variable free we can talk about the effects of external actions which determine the execution paths.

5.1.3. *Action* In axiom sets which embed models with behaviour, maps which change with respect to time are represented as sequences in the state space. Of these maps some represent the effects of actions external to the model and other represent the response of the model to change. The operation consists of evaluating map change with respect to time and taking action to further modify attribute values for state variables. Action within a model is triggered by a change in a map value. The model response is to change other map values.

A simple case of a model which changes as a response to its external environment is a switch, which when turned on increments, and when turned off resets the counter to 0. Let

$$M = (x : N, switch : \{on, off\}) \tag{223}$$

$$if\, switch@k = on\ then\ x@k + 1 := count@k + 1 \tag{224}$$

$$if\, switch@k = off\ then\ x@k + 1 := 0 \tag{225}$$

Note that M represents a system with a projection x whose value type is N and a projection $switch$ whose value type is $\{on, off\}$. The two formulae which define the behaviour of the system are Horn clauses in the single integer variable **k**. It increments when the system is "on" and returns to 0 when the system is "off". Note that the equations above are Horn clauses and so are Algos axioms. The valid interpretations are described by the sequence of settings for *switch*.

When model to be axiomatized is not necessarily a product type the construction above can be modified to represent the states as projection maps. For example, if a type M is to have two "attributes" $x : N$ and $switch : \{on, off\}$ the product

$$(s : M, x : N, switch : \{on, off\}) \tag{226}$$

represents the two attributes as projection maps. The symbols s, x, and $switch$ are all projection maps on the product. The product contains M as a factor. The tuple $< s, s.x, s.switch >$ has type

$$< s, s.x, s.switch >: M \to (s : M, x : N, switch : \{on, off\}). \tag{227}$$

5.1.4. *State machines* Perhaps the most simple state machine is one which turns on when it is off and turns off when it is on. This machine can be defined by

$$switch(x : \{on, off\}) \to \{on, off\} \tag{228}$$

$$x.switch = case(incl_{on} : x := off, incl_{off} : x := on) \tag{229}$$

The run map of this machine is the map

$$switch^* = do\, switch\, forever \tag{230}$$

Only the initial state of the machine matters.

The Algos axiom sets considered here make the restriction on the engineering models and their axiom sets that the mutable maps are projection maps onto data types such as N and $\{on, off\}$ in the model M above. The internal actions generalize the *switch* behaviour to that of state machines. To represent the complexity of composite structure models the approach of using first order variables for time has to be integrated with structural part decomposition, as well as the fact that the functions representing behaviour interact in random ways. Natural number time can be generalized to real number time.

5.2. *An Engineering Modelling Perspective*

This subsection outlines modelling language constructions semantics which are used by engineers to design and analyse complex systems such as automobilies and aircraft. SysML implements these language constructions. These modelling constructions have proven sufficiently precise for engineering analysis with their informal semantics; they also lend themselves to the kind of axiomatic and model theoretic semantics possible in Algos. This subsection outlines these constructions and gives a well-defined axiomatic semantics.

Analysis and design of a system such as an automobile with its operating behaviour requires modelling the operating environment of the system of interest. Often the systems of interest are described as reactive systems in the sense that they react to changes in their environment. Common engineering modelling practice for the design or analysis of reactive systems is to model the system of interest in the context of a model of its operating environment. The composite of the two models is used to analyse and reason about the system behaviour. These systems have sensors which respond to perceived change and effectors which respond to perceived change. Change originating outside the system results from the effect of physical laws and from actions taken by external agents in the environment.

For an engineering model of a reactive system the model axiom set will contain variables whose values change during operation. These variables are called state variables. A valid interpretation of the model maps the state variables to a product of the types of the variables. The behaviour variance of the system is represented by paths in the state space as it evolves in time. The physical laws transform the state space responding to actions of the system. Since the variables in general range over a space-time region the interpretations contain functions defined for the space-time.

As modelling is becoming state of the practice in engineering design and analysis, the model becomes the authoritative information source, not only for the design, but for analysis and verification of the system's capabilities. As a result it is becoming evident that the physical laws which effect behaviour have to become part of the combined system and environment model. The inclusion of physical laws is also necessary for the simulations to have validity. While multiple behavioural constructions are used in computer science and modelling languages only two kinds of behavior are considered here, state machines which represent the behavior of human or machine actors and physical laws which transform the state space.

5.2.1. *Unique Decompositions* From both the viewpoint of reasoning about composite structures and constructing valid interpretations the embedding of a model into a logic-based framework has as requirements that any instance of a composite structure has a unique component decomposition. The kinds of components encountered in composite structure models include, but are more general than the part maps of the molecular examples. The components of an instance of a type X, in addition to parts, include attributes, operations which have arguments, and "state machines" which effect state change. The individual components that represent data values and operations are uniquely associated with an individual component which owns the operation or data storage. Each of the different kind of components are used for a specific kind of model expressiveness. As a consequence each kind of component is embedded in a form specific to the kind of component.

5.2.2. *Attributes* One of the simplifications made is to restrict the mutable maps to projection maps which correspond to state variables in the axioms. A system component a of type X retains its identify and its change is measured by the change of maps whose domain is X and whose range type is a projection map onto a data type. Such a map is called an attribute of a. The attribute is then well-formed for each individual of the type. This restriction enables attributes to be represented as projection maps in Algos and corresponded to state variables in the Horn clause axioms for the model. A model explicitly declares which attributes are mutable. Thus it is the modeler's responsibility to identify mutable attributes. For example a model of an aircraft for flight test might make the complete outer surface to be mutable. Mutable maps are maps whose values change with respect to time. Interaction between systems takes place through the mediation of causal relationships between what we call mutable attributes. These are attributes of an individual which are subject to change either from internal or external causes. The mutability is represented by the change of an object with respect to time or space-time.

5.2.3. *State machines* This form of behaviour construction used in Composite Structure Models is described by state machines. This construction is sufficient to illustrate how behaviour works within the Algos formalism. While in general hierarchical and concurrent state machines are needed to represent Composite Structure Models only non-hierarchical machines are considered here. A state machine is a potentially reusable individual as is an operation. This requires that a state machine has to be identified with the part component to which it belongs. The state machines in the examples are restricted to access (read and write) attributes of the type which the state machine is a component.

5.2.4. *Physical Laws* An engineering modelling technique, well supported by language constructions in SysML, uses equations to model causal or associative change between attributes that result from the physical laws used by the model. The causal relationships are expressed by equations whose variables are bound to attributes in systems or their components. These equations are encapsulated for reuse purposes with variables which are bound to the attributes for a specific application. Approximations of physical laws are often used for analysis including inference. It is becoming increasingly clear to modelling

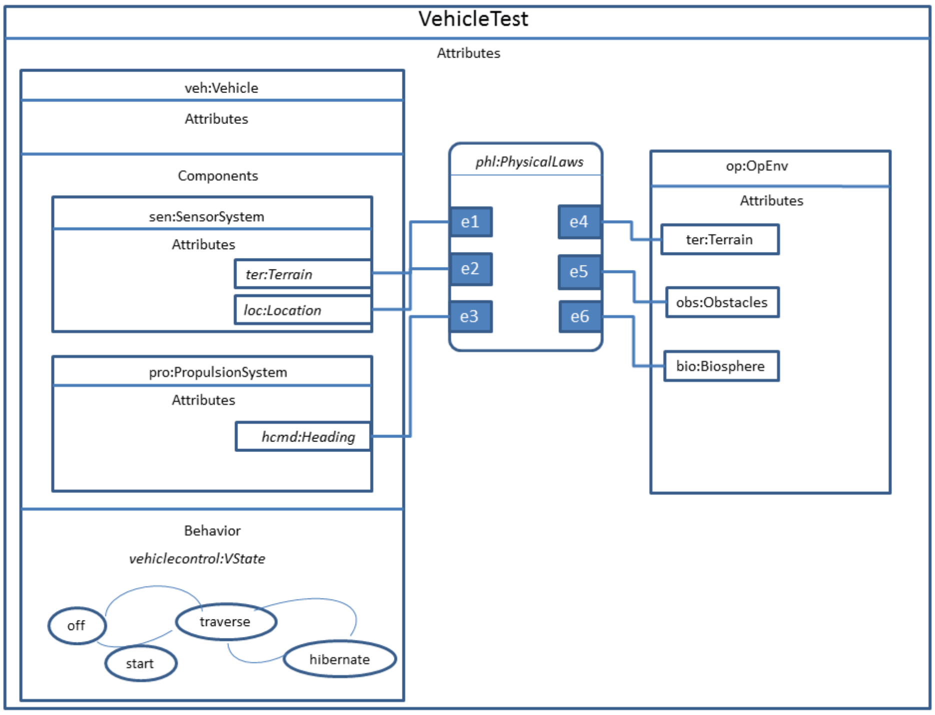

Fig. 4. Vehicle Test Setup

practitioners that these assumptions need to be part of the model as the analysis is contingent on these assumptions. The approach of using equations to represent associative relationships is used in the vehicle example. While the issues of finding equations and their solvability are at the centre of much engineering analysis these complexities will not be dealt with here.

5.3. *A Vehicle Test System*

A Composite Structure Model is illustrated with a model of a vehicle in an operating environment. For this example a number of simplifying assumptions are made to illustrate the semantics of the language constructions. After presenting the $VehicleTest$ model in a graphical syntax based on SysML we give the corresponding linear syntax. The concepts such as parts structure introduced for the molecular example will be used here. This example makes also use of the sum and exponential type constructions. A simple kind of state machines, represented in Algos, are used to specify behaviour. The state machines will imply that the terminal type has a time structure. Additional notation is introduced which is defined within Algos.

The $VehicleTest$ model illustrated in Figure 4 contains an autonomous vehicle operating in a physical environment. The vehicle has sensors, propulsion. The physical environment has attributes for terrain, obstacles, and biosphere conditions. For simplicity we assume that these attributes are immutable. The effects of the vehicle behaviour are mediated by physical laws governing motion and sensor precision. A vehicle executes a predetermined plan to traverse waypoints in its environment. If the sensors detect any anomalies such as obstacles which prevent it from reaching one of its waypoints its plan provides alternative actions. The behaviour of this vehicle is described by a single state machine. The state machine changes its internal state and controls its propulsion in response to changes in the sensor attributes. The operation of the vehicle can be described as transitioning between the following states:

1 off
2 startup - turns on sensors and propulsion and initializes the state
3 traverse - computes next waypoint and gives commands to go there
4 hibernate - shuts down waiting for external command

Both its sensors and effectors are mediated by physical laws. The electro-optical laws degrade sensor precision. The laws of motion for propulsion are mediate the effects of the biosphere. Physical laws can be used to represent the effects of time delay for a sensor to detect an object in the environment and relay the result for further processing to set the heading.

Many engineering examples are an elaboration or refinement of this general pattern. This template has been used for a number of aircraft design and analysis studies (Graves and Bijan 2011), as well as, for underwater vehicles, and for design analysis for a robotic vacuum cleaner.

5.4. *Graphical Syntax*

The graphical in Figure 4 syntax follows SysML closely. In general, for large models, multiple diagrams are used. A particular diagram may hide some of the structure in the interest of intelligibility. However, this model only uses the single diagram. The diagram in Figure 4 contains the signature of the Algos axiom set. The description and discussion below is similar to that which would be used to explain the diagram to an engineer. An informal description of the role and meaning of the components is included. The diagram is a more efficient way to represent the model for human consumption than is the linear syntax that will be given for this model.

5.4.1. *Component Hierarchy* The diagram contains a top level rectangle labelled $Vehicle-Test$. The rectangle has three components, labelled $veh : Vehicle$ and $op : OpEnv$ and a rectangle with rounded corners, labelled $phl : PhysicalLaws$. These blocks have interior structure. Interior rectangles of $veh : Vehicle$ are connected to interior rectangles of $phl : PhysicalLaws$ as are interior rectangles of $op : OpEnv$. The $veh : Vehicle$ rectangle has two non-empty compartments, *components* and $behavior$. The *components* compartment has two rectangles one labeled $s : SensorSystem$ and $p : PropulsionSystem$. Each of these components has attributes. The block labelled $Vehicle$ has interior subdivisions which show component structure. The block has compartments labelled *Attributes*, *Components* and $behavior$. The attribute compartment of $sen : SensorSystem$ has two attributes, $ter : Terrain$ and $loc : Location$.

5.4.2. *Vehicle State Machine* The vehicle *behavior* compartment contains the symbol *vehiclecontrol* which is the name of the state machine which controls the behaviour of the vehicle. The compartment also a graphical representation of the state machine from which we can determine the states and the actions which cause state transition. In the graphical syntax the *vehicle* type has a compartment called *behavior* which contains the expression $vehiclecontrol : VehicleState$ as well as a diagram in the lower part. Within the *behavior* compartment of $Vehicle$ the name *vehiclecontrol* references the state chart in the bottom part of Figure 4. Informally *vehiclecontrol* reads the location sensor attribute, calculates the heading for the next waypoint and updates attribute *hcmd*, or takes an alternative action if the sensors indicate any issues. The current state is updated. The behaviour operation can read and write mutable attributes in the context of an instance of $Vehicle$. The vehicle control operation does not have access to any other attributes. It controls behaviour only on the basis of what its sensors can see. What they see is mediated by their characteristics and the laws of physics.

5.4.3. *Operating Environment* The rectangle $op : OpEnv$ only contains attributes which are $ter : Terrain$, $obs : Obstacle$, and $bio : Biosphere$. For this example we may assume that the attributes op, ter,and *bio* are immutable. However, in generalizations of this example that need not be the case. For example when the vehicle is a ship, then the environment may model a changing sea state. Further, in more complex models the motion of the vehicle may effect the biosphere. Modelling this aspect requires the physical laws to model the effects of motion on the biosphere.

5.4.4. *Physical Laws* The rectangle with rounded corners corresponds to a SysML constraint block. A constraint block encapsulates a set of equations in a finite number of variables. The small rectangles in $PhysicalLaws$ are connected by lines connecting to attributes in the context of the diagram. The block with rounded corners $PhysicalLaws$ contains four small rectangles which we call variables. They are labelled $e1, e2, e3, e4, e5, e6$. These rectangles are connected by lines to the attributes of respectively *ter*, *loc*, and *hcmd* which are all attributes of components of $Vehicle$. Interior to $PhysicalLaws$ are functional equations of the form $f1(e1) = e3$ and $f2(e2) = e4$. The lines represent binding operations so that when the bindings are made. including the small rectangles in $veh : Vehicle$ and $op : OpEnv$ connect to small rectangles within $phl : PhysicalLaws$.

The diagram in Figure 4 contains the signature of the Algos theory to be generated by the model. This signature contains the type and map symbols in the diagram. The map symbols include the attribute symbols. The model may contain initial values for attributes. Some of the attributes may be immutable. The model also contains the equations which relate the attributes of the different types. The vehicle testing set up is represented by a type, $VehicleTest$ as it may have multiple instances. When testing a device or analysing the behaviour of a system either by physical test or by simulation one generally constructs multiple instantiations of the type corresponding to the application domain. Most system analysis and verification is concerned with probabilistic behaviour as calculated from the collection of realized instances of the experiment.

5.5. *Linear Syntax*

The graphical syntax is notable for its use of hierarchical containment structure. The linear syntax contains equivalent information omitting placement and scale, albeit in a less visual form as a graphical version can be generated from the linear syntax. The linear syntax does contain additional assumptions needed for constructing the axiom set from the graphical syntax. These assumptions could be made part of the graphical representation. In the graphical syntax the rectangles have compartments such as *parts*,*attributes*, and *operations*. These compartments contain declarations for the particular types of components. The intent of the declarations is that any instance of a type such as $v : vehicle$ has the components declared in the compartments. Some of these compartments have mutable state. For example, a vehicle might have a temperature gauge which changes in response to the engine operating conditions. Each of the kinds of compartments will use specific conventions to provide an association between the value of the component and the instance of which it is a component.

The linear syntax for a composite structure axiom set which embeds the graphical syntax follows it closely. However, the typing of the maps that occur in the axiom set reflect an axiom set requirement that the axioms provide a unique decomposition for the part maps and enable attributes, operations, and state machines to be uniquely associated with the component part to which they belong. An equivalent linear syntax for $VehicleTest$ starts by introducing notation to correspond to the compartments. While the conventions for each compartment as defined below are different they all define maps which are monic. This means that the part components, as well as the other components defined by the compartments are distinct for each instance of the type $VehicleTest$.

5.5.1. *Parts* Part maps, as occur in the molecular example, occur in the vehicle test example. Note that $VehicleTest$ has a components compartment with three components, $veh : Vehicle$, $op : OpEnv$, and $phl : PhysicalLaws$. $Vehicle$, $OpEnv$, and $PhysicalLaws$ are types. $op : OpEnv$, and $phl : PhysicalLaws$ are the image types. The interior compartments of the three image types are assumed to be interior types of the three types, $Vehicle$, $OpEnv$, and $PhysicalLaws$. The compartments declare operations for the three types which are inherited by the three image types. The assumption for part maps, as well as the other component maps, is that they are monic. This means that, for example, the parts of vehicle are distinct. Without an explicit assumption, such as was made in the molecular example, a vehicle may share parts with another vehicle. The three rectangles in $VehicleTest$ translate into the three axioms

$$veh : Part(VehicleTest, Vehicle) \tag{231}$$
$$op : Part(VehicleTest, OpEnv) \tag{232}$$
$$phl : Part(VehicleTest, PhysicalLaws) \tag{233}$$

The expression $veh : Part(VehicleTest, Vehicle)$ is equivalent to

$$veh : VehicleTest \to Vehicle, veh :: Part \tag{234}$$

Thus the symbol veh is a map with the designated range and domain. As with the molecular example $veh : Vehicle$ is identified with the image $Im(veh)$. The use of $Part$ means that the maps designated as parts are assumed to satisfy the part property axioms. Note that a vehicle test vt has a vehicle part and the vehicle part has a propulsion system. The part path component maps provide a unique decomposition tree for any individual $vt1 :: VehicleTest$. The part paths for $VehicleTest$ are

$$veh, veh.sens, veh.sens, veh.sens, veh.pro, phl, op \tag{235}$$

For any instance $vt1 :: VehicleTest$ the composition provides a tree of instances

$$vt1, vt1.veh, vt1.veh.sens, vt1.veh.pro, vt1.phl, vt1.op \tag{236}$$

The other components such as attributes will be uniquely identified by the part to which they belong. The other kinds of components are represented slightly differently.

5.5.2. *Attributes* Attributes are a kind of component. For a sensor system individual $s1 : SensorSystem$ the map $s1.loc$ is an individual with $s1 :: SensorSystem$. $s1.loc$ represents the location of the sensor individual $s1$. The value type of loc is a variable of type $Location$. $Location$ is assumed to be a 3D vector in some coordinate system. The value of $s1.loc$ which is a 3D vector which may change during the course of operating the system that contains $s1$. Within a model of $VehicleTest$ in which the values of $s1.loc$ change we need to be able to track that the 3D vectors are associated with the individual $vt1.veh.sen.loc$.

The linear syntax for the attributes of $SensorSystem$ uses the notation

$$loc : Attriubte(SensorSystem, Location) \tag{237}$$
$$ter : Attribute(SensorSystem, Terrain) \tag{238}$$

The axiomatic description for attributes satisfies the following properties.

— An attribute of an individual is a possibly mutable value of its domain type
— the attribute and attribute value for an individual can be uniquely identified
— operations involving substitution can be composed with attributes.

The substitution properties are achieved when the attribute maps are projection maps. If $SensorSystem = (x1 : Location, x2 : Terrain)$ we could define the two attributes as the two projection maps. However, $SensorSystem$ is not necessarily the product. The uniqueness and substitution objectives can be achieved for $SensorSystem$ using the product

$$(s : SensorSystem, loc : Location, ter : Terrain) \tag{239}$$

The symbols s, loc, and ter are projection maps on the product. The product contains $SensorSystem$ as a factor. The attribute syntax is equivalent to the following map

$$< s.s.loc, s.ter >: (s : SensorSystem, s.loc : Location, s.ter : Terrain) \to \tag{240}$$
$$(SensorSystem, loc : Location, ter : Terrain) \tag{241}$$

Note that this single tuple for the two attributes of SensorSystem incorporates all of the maps of the attribute compartment.

The map schema ($loc := k$) is defined for any product which contains loc as a projection. Thus for an individual $s1 :: SensorSystem$

$$< s1, s1.loc, s1.ter > (loc := k) =< s1, k, s.ter > . \tag{242}$$

Thus for any individual $s1$ of $SensorSystem$ the triple $< s1, s1.loc, s1.ter >$ uniquely associated two individual values $s1.loc$, and $s1.ter$ with the individual $s1$. As the part path compositions provide a unique component decomposition of any individual $vt1$ of $VehicleTest$ these part components with their attribute value tuples provides a unique representation for the attributes that vary. This decomposition is extended to include the state machines states. The state machine states are also projection maps.

5.6. *Operations*

A type X may have a compartment called *operations*. An operation $t.f$ is a component of X which for an instance of $t :: X$ may have applicative arguments. An operation component represents a capability which belongs to the instance. Since application terms are not necessarily unique to the type X, the elements and the *operations* compartment in linear syntax is

$$\begin{aligned} &f1 : Operations(X, Y1^{Z1}) \\ &..fn : Operations(X, Yn^{Zn}). \end{aligned} \tag{243}$$

The $f1, \ldots, fn$ are represented as a tuple of maps

$$< pr1, pr2.f1, \ldots prn.fn >: X \to (pr1 : X, Y^Z, \ldots Y^Z). \tag{244}$$

Thus, for an individual $a :: X$ where there is only one operation f the

$$a. < id, pr1.f >=< a, a.f : Y^Z > . \tag{245}$$

The operations are used by the state machines which bind them to attribute arguments and may bind the value to an attribute.

5.7. *State Machines*

This section describes the syntax of the map *vehiclecontrol* which represents the state machine. The behaviour of the state machine is described in Section 5.8. Syntactically, the state machine consists of actions which have attributes and a current state as arguments. The state machine describes for these arguments an update for attributes and a transition to a new state. The states of the machine are the type $Vehiclestate$ which has 4 individuals: $off, startup, traverse,$
$hibernate$ which is represented:

$$Vehiclestate = \{off, startup, traverse, hibernate\} \tag{246}$$

In Algos the domain of *vehiclecontrol* are both the product of $Vehicle$ together with the types $VehicleState$, for is the current state, and the attributes of the subsystems

$SensorSystem$ and $Propulsion$. Thus the type of $vehiclecontrol$ is

$$\begin{aligned} vehiclecontrol : (v : Vehicle, v.sens.loc : Location, vehs : VehicleState) \\ \to (v : Vehicle, hcmd : HeadingCmd, veh : VehicleState) \end{aligned} \tag{247}$$

Thus for an individual $vt1 :: VehicleTest$

$$\begin{aligned} vt1.veh.vehiclecontrol. < vt1.veh, vt1.sens.loc, vt1.veh.vehs > \\ =< vt1.veh.pro.hcmd, vt1.veh.vehs > \end{aligned} \tag{248}$$

when executed effects the state transition.

Syntactically, the commands are represented as modifying attribute values, i.e., as substituting values for mutable attributes. The change in these attribute values is caused by external action when a vehicle is placed in an operating environment. The behavior depends on the sensor values. The digram in the lower part of the figure is a state chart that describes the engine run operation. It is referenced in the *behavior* compartment of $Vehicle$. The vehicle control will have a number of condition-action statements with a state transition. For example

$$vehiclecontrol(loc, ter, vehs) = (hcmd := do\, nothing, vehs := off) \tag{249}$$

The engine run operation described by the state chart can also be defined using a case statement. Conditional statements can be defined in terms of the sum type construction which is definable within Algos.

$$\begin{aligned} runengine = case(\\ f(loc, ter) = normal\, do < heading := g(heading), normal > \\ nearobstacle(loc, ter) = true\, do < heading := h(heading), normal > \\ seriousproblem(loc, ter) = true\, do < heading := stop, hibernate > \\ ) \end{aligned} \tag{250}$$

Several constraints are imposed on the behaviour of this reactive system. All interaction of the engine and its testing environment is through attributes of the vehicle. The attributes of the vehicle are mediated by equations which are part of $PhysicalLaw$. The run or control operation for the engine, $hcmd$ transitions vehicle states when ever the value attributes change. The way that behavior is effected is described in the subsection State change with respect to time. The state machines that effect a system's behaviour may use operations that belong to the systems which the state machine belongs to.

5.8. *State Space of Vehicle Test*

A Composite Structure Model such as the vehicle test has a unique parts decomposition and all of the mutable attributes are associated with a part. The mutable attributes of a composite structure model and the state machine variables define the state space of a Composite Structure Model in the sense that these components are the only ones that vary. When constructing valid interpretations (logician's models, engineer's simulations) the evolution of these states characterizes the dynamic behaviour of the model. This is under the assumption that the model is consistent. The dynamic model is a vector valued function of time. This subsection introduces notation to simplify representing the state space.

The composition as dot notation gives a unique decomposition for all of the part components of a vehicle. For an instance $vt1 : VehicleTest$ only the attributes and machine state change with respect to space-time. A model of the dynamics of $VehicleTest$ only needs to represent how these attributes change. The three attribute maps, *veh.sen.ter*, *veh.sen.loc*, and *veh.prop.hcmd*, and the state map *vstate* describe the mutable states with respect to time. The state space of vehicle test is the product of the types that occur as the variable types of the mutable attributes and the state machine types. To understand the structure of the state space, the type

$$(pr1 : Vehicle, pr2 : (pr21 : SensorSystem, pr22 : Location, pr23 : Terrain)) \tag{251}$$

describes the component type needed for the vehicle sensor and its two attributes. Note that loc is defined as $pr22$ and ter is defined as $pr23$ after renaming the projections. Thus,

$$< pr1, < pr21, pr22, pr23 >>=< pr1, < pr21, loc, ter >> \tag{252}$$

and

$$\begin{aligned} &< veh, veh.sen > . < pr1, < pr21, pr22, pr23 >>= \\ &\qquad < veh, < veh.sen, veh.sen.loc, veh.sen.ter >> \end{aligned} \tag{253}$$

For an individual $vt1 :: VehicleTest$

$$\begin{aligned} &< vt1.veh, vt1.veh.sen > . < pr1, pr21, loc, ter >= \\ &\qquad < vt1.veh, < vt1.veh.sen, loc, ter >> \end{aligned} \tag{254}$$

This tuple contains the two variables loc and ter. These variables can be assigned values. For example, let $k1 :: Location$ and $k2 :: Terrain$. The composition

$$\begin{aligned} &< vt1.veh, < vt1.veh.sen, loc, ter >> .(loc := k1).(ter := k2) = \\ &\qquad < vt1.veh, < vt1.veh.sen, k1, k2 >> \end{aligned} \tag{255}$$

The tuple map

$$< veh, < veh.sen, loc, ter >, < veh.pro, hcmd >, < veh, pr3 >> \tag{256}$$

has type

$$\begin{aligned} &(loc : Location, ter : Terrain, hcmd : VCommand, VState) \to \\ &\qquad (Vehicle, (SensorSystem, Location, Terrain), \\ &\qquad (Propulsion, PCommand), pr3 : VehicleState) \end{aligned} \tag{257}$$

The type is the mutable part of the state space of $VehicleTest$. The tuple allows us to identify each attribute and machine state variable with respect to the component to which it belongs.

An initial state for the $VehicleTest$ is an individual $vt1 :: VehicleTest$ and constants $k1, k2, k3$ for the three attributes, and an initial state machine state, say off. This state

$$< vt1.veh, < vt1.veh.sen, k1, k2 >, < vt1.veh.pro, k3 >, < veh, off >> \tag{258}$$

identifies the attributes uniquely and their values. Now we examine how the state space evolves.

5.9. *State Change with respect to Space-Time*

An attribute $f : X \to A$ is mutable if $a.f@i$ is non constant as i varies with respect to time for an individual $a :: X$. A typical example of a mutable attribute is vehicle sensor attribute ter.

$$ter : SensorSystem \to Terrain \tag{259}$$

Then for an individual $vt1 : VehicleTest$ and a time i

$$vt1.veh.sen.ter@i : Terrain \tag{260}$$

We introduce a syntax of initialize an attribute.

$$ter : Engine \to Ter[@t0 = 0] \tag{261}$$

For example the location of a $Vehicle$ is defined within $VehicleTest$ context.

$$veh.sen.loc : Vehicle \to Location \tag{262}$$

The state of an attribute loc of a vehicle v at time t is described by

$$v.veh.sen.loc@t \tag{263}$$

An instantaneous state of a model instance is described by a vector in the state space of $VehicleTest$. A simulation for $VehicleTest$ consist of recording the values of each of these “states” for each time instance, starting with $t0$.

The specific language constructions and axioms used in an applications, as with any Algos application axiom set, reflect ontological assumptions about the application domain and carry the responsibility to provide a means to verify interpretations. The reactive systems with state machines have served effectively to model and simulate complex physical systems. We make the simplifying assumption that only maps designated as attributes change.

5.10. *Semantics*

The assumptions used for the vehicle test system enable its dynamic behaviour to be described by the evolution of the attribute vectors in its state space through time. This characterization can be generalized to arbitrary Composite Structure Models. For simplicity we assume that a Composite Structure Model has only a single state machine and may have physics laws. Then the Composite Structure Model has a state type

$$S = (X1, \dots, Xn, SM, N) \tag{264}$$

where the Xi is an enumeration of the attributes defined by the unique part decomposition, MS is the machine state type, and N is discrete time. Note that $+1$ is a function on N. The projection maps can be viewed as first order functions. The only axioms we have encountered so far in the Vehicle Test Model are those associated with state change. The axioms for state machine are expressed as Horn clauses in $n+2$ variables which we write as $\mathbf{x1}, \ldots, \mathbf{xn}, \mathbf{ms}, \mathbf{i}$.

$$P1(\mathbf{x1}, \ldots, \mathbf{xn}, \mathbf{ms}, \mathbf{i}), \ldots, Pk(\mathbf{x1}, \ldots, \mathbf{xn}, \mathbf{ms}, \mathbf{i}) \Rightarrow Q(\mathbf{x1}, \ldots, \mathbf{xn}, \mathbf{ms}, \mathbf{i+1}) \qquad (265)$$

where Pi are the predicates which tests if the preconditions for the state machine is satisfied and Q is the predicate which tests that the action of the machine has changed the state. The physics laws, which have not been stated for the vehicle test model, are also assumed to have the same form where the P and Q are atoms in the language of the signature of the Algos axiom set embedding the model. These assumptions provide rules which govern the evolution of the state space and provide the definition of an execution semantics for an instance of a Composite Structure Model.

Applications of Composite Structure Models are often concerned with whether a specific design configuration can, under given preconditions, satisfy postconditions. The precondition-postcondition statements can generally be represented the same Horn clause form in the language of the application model. Often the preconditions are provable from the axioms of the theory. If not they are added to the axiom set provided they do not make the axiom set inconsistent. Then one looks to see if adding the consequent $Q(\mathbf{x1}, \ldots, \mathbf{xn}, \mathbf{ms}, \mathbf{i})$ makes the axiom set inconsistent. If not then one can conclude that the assertion follows from the axioms in the theory. Examples of this are described in (Graves and Bijan 2011). Simple examples of inference for the vehicle test model is that for any execution sequence it halts or completes its waypoint traversal. One can define modal operators with respect to whether an execution reaches a particular state space configuration. With some extensions to the state machine it is possible to prove that certain state configurations are eventually obtained.

5.10.1. *Models and Simulation* Engineering analysis for dynamic systems make extensive use of simulation to determine reasonable preconditions under which the system operation can obtain the desired outcome, and to rule out unworkable design solutions. Simulation in the Algos context is a valid interpretation of the axiom set representing the system and an operating environment. Practically, simulation is often used to decide what machine states and state transitions are needed, as well as whether additional sensors are needed.

The connection between simulation and topos-based model theory is made informally here. For a Composite Structure Model axiom set all of the part maps are compositions which originate from a *start* type. In the case of the vehicle system model the start type is $VehicleTest$. The valid interpretations of a composite model are function spaces with a base space consisting the product of the state space including the machine states and time. For a composite model execution (simulation) is defined for an instance. The execution of an instance is a path of state tuple (vector) whose components ordered by time. For vehicle test model these state vectors have the form

$$\begin{aligned} s =< vt1.veh, < vt1.veh.sen, loc, ter >, \\ < vt1.veh.pro, hcmd >, < vt1.veh, pr3 >, t > \end{aligned} \tag{266}$$

where $vt1 :: VehicleTest$ and t is time. For the vehicle test model actions are performed by the vehicle machine *vehiclecontrol* and the physics laws. A vehicle test instance $vt1 : VehicleTest$ is executed from an initial state by successively updates to the mutable attributes and the next machine state.

The vehicle machine can be viewed as an agent whose actions are its state updates and the setting of the heading commands. Using a labelled transition notation this can be written as:

$$s \ ^{machine} \rightarrow s' \tag{267}$$

As an example of an execution of an instance of vehicle test consider the case where the initial machine state if off at $t0$. The machine turns on at $t0 + 1$ and sets *hcmd* for the first waypoint. At the next time instance the terrain sensor *ter* detects an obstacle. If we start at $t0$ with the initial state

$$< vt1.veh, < vt1.veh.vt1.k1, loc, k2 >, < vt1.veh.pro, k3 >, < veh, off >> \tag{268}$$

Then

$$\begin{aligned} < vt1.veh, < vt1.veh., vt1.veh.sen, k2 >, < vt1.veh.pro, k3 >, \\ < vt1.veh, off >> .vehiclecontrol \end{aligned} \tag{269}$$

checks the values of *loc*,*ter* and sets *hcmd* and updates the state to yield

$$\begin{aligned} < vt1.veh, < vt1.veh., vt1.veh.sen, k2 >, < vt1.veh.pro, k3 >, \\ < vt1.veh, off >> .vehiclecontrol \end{aligned} \tag{270}$$

Note that an event is a state which triggers a state machine action, i.e.,

$$s.mstate@t + 1 \neq s.mstate@t \tag{271}$$

where *mstate* is the projection map onto the machine state type.

We assume that the environment actions are performed by the laws of physics. These laws can be represented by state transition equations. These laws can also be viewed as an agent which transforms the state space. These laws can also be written as:

$$s \ ^{laws} \rightarrow s' \tag{272}$$

For simulations the equations bound to the specific state variables are solved and their values are assigned within the next state vector. In general the semantics for a composite structure model can be represented using Labelled Transition Systems (Knight *et al.* 2012).

As usual the states are too fine grained for analysis and reasoning and so one is interested in equivalence classes of states such as are defined by bisimulation relations. Execution for axiom sets exhibit variance not found in static models. Coping with vari-

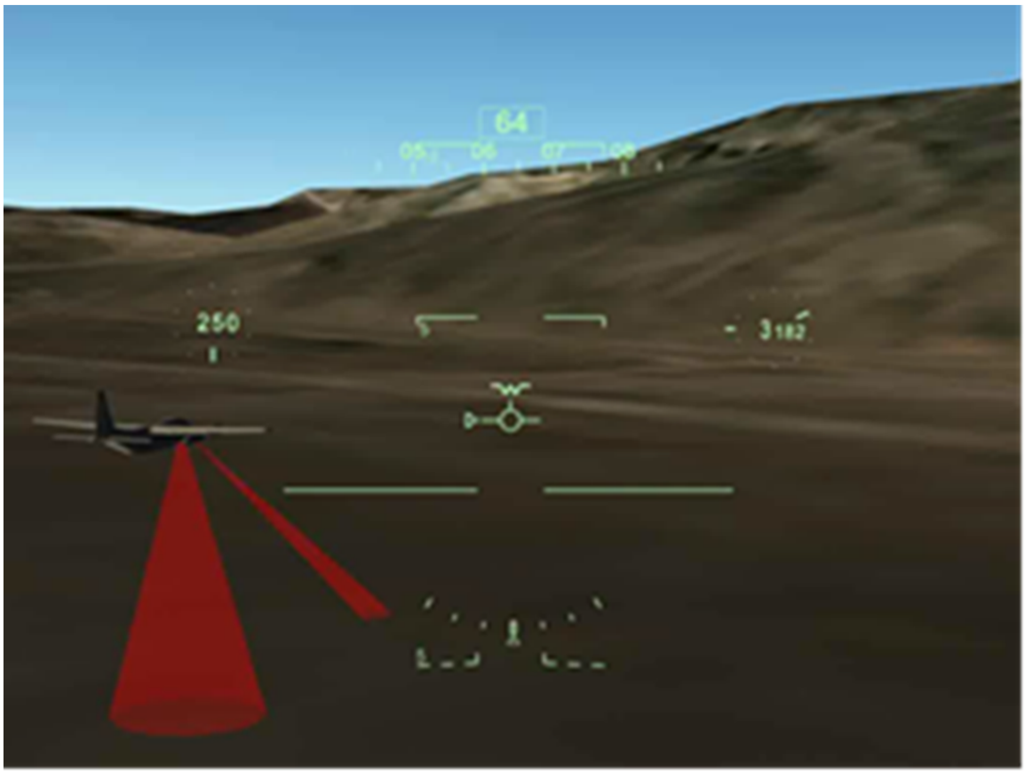

Fig. 5. Terrain Following and Obstacle Avoidance

ance has led to using equivalence relations such as bisimulation to enable study of models to be restricted to manageable classes. These issues go beyond the scope of this paper.

For $VehicleTest$ the execution the only thing that effects its behaviour is whether there is an object in its path. Of course the model is so simplistic that one can not tell much about its behaviour without further refinement and elaboration. For example, the machine has no way of knowing if it has hit an obstacle other than whether it sees the obstacle in its terrain image. A technique common for behavior analysis is to add attributes to the top level, i.e., to $VehicleTest$ that serve as ground truth regarding the distinction between what the vehicle's sensor location value and the true location with respect to the model execution. This is used, for example, to compare the heading command with the vehicle movement as calculated from the equations in the model for motion. These equations may be physical laws or laws constructed from empirical test. Generally executable models such as $VehicleTest$ are equipped not only with additional global sensors, but displays to view progress, as well as mechanisms to record the results of an execution.

5.11. *Notes*

The argument advanced here is the that by embedding models as axiom sets automated inference can be used to solve engineering problems. For example the kind of analysis common in engineering modelling is to determine if an aircraft under specified flight conditions can detect and identify an object on the ground. These examples employ both electro-optical laws and aerodynamic laws for the motion of the aircraft in specific atmospheric conditions. A variant example is to determine that an aircraft can be modified to have an object detection and avoidance subsystem. The object detection and avoidance system detects obstacles in the terrain of an aircraft in sufficient time for the aircraft to avoid them. While full examples are too complex to work out here in detail, this section illustrates how the engineering models can be embedded as axiom sets in Algos for which automated reasoning can be used as part of the analysis.

Figure 5 is a still from a video of a simulation of an aircraft analysis model (Graves *et al.* 2009). The picture is a virtual aircraft flying low in mountainous terrain. The red cones are a radar looking for obstacles in front of the aircraft and the small cone is a radar altimeter. The overlays in green are what the pilot sees on the cockpit display. The terrain is a representation of the earth for a certain part of the world. This model was constructed to help answer whether a certain design modification to the aircraft enables the aircraft to recognize an obstacle, e.g., cell tower or mountain peak not maps in sufficient time to avoid it. The model contains a representation of how the aircraft motion in turbulence at specific altitudes and atmospheric conditions. Based on simulation data we were able to define precise pre and post conditions for the terrain following and obstacle avoidance example and give an informal proof that this capability followed from the design.

The composite structure models of this section cover a broad spectrum of engineering models. These models are used both to describe existing systems and as specifications for new ones. These models regularly employ science models. If one removed the vehicle from the vehicle test model one is left with scientific models. The vehicle component could be replaced with a component to test the scientific theory. One difference between the scientific models and the engineering models is that for scientific models one is likely looking for a model in which all of the interpretations are the same. Such a theory is sometimes called categorical. For engineering models one more frequently encounters the situation where it is recognized that there may be multiple valid interpretations of the model. This possibility only emphasizes the usefulness of employing the precise concept of valid interpretation from logic. As one actually uses the model (theory) to make analytic conclusions the justification of reasoning plays an increasingly important role.

The intent of this section is to validate that putting an axiomatic foundation under practical broad scope modelling languages is feasible and has potential benefits. For some situations some of the constraints used here might be relaxed or changed. However, with these constraints very large complex engineering models have been constructed. At lease informally, design conclusions have been made on the basis of these models. Discussion of the constraint types and their equations is not addressed here.

The specific language construction found or needed in engineering modelling languages have been treated here in a cursory and incomplete way due to lack of space. One example of a language construction which can be added to the current discussion is a type for *ports*. Composite structure models often make use of components called ports. In detailed models connections between components is by connecting a port component of one subsystem to a port component of another system. The type of a port (i.e., its attributes) are used to describe what flows through the port, whether the flow is discrete, continuous, unidirectional, or bidirectional, and possibly other constraints. Port types are readily accommodated within this framework.

5.11.1. *Type Definitions* Another topic not discussed is that of type definitions and type equations. Here the only type definitions and type equations permitted are type definitions which introduce a symbol for an Algos type construction. Other type equations are not allowed. However, a model such as vehicle test can be characterized by the type definition of its "start" type, in this case $VehicleTest$ using the DL existential type constructions. For example, to represent that the type $VehicleTest$ the type definition note that subtype relations such as

$$VehicleTest \sqsubseteq \exists veh.Vehicle \sqcap \exists op.OpEnvironment \tag{273}$$

can be expanded and converted into an equality. Thus, for example for an individual $vt1 :: VehicleTest$ one obtains the existence of a vehicle

$$vt1.veh :: Vehicle. \tag{274}$$

5.11.2. *Generalization and Specialization* The efficiency of engineering model development depends on having a large body of reusable models which can be composed to provide a leveraged starting point. We have suggested that the vehicle test model is a case of a general paradigm that is applicable to many applications. Also physics models tailored for engineering purposes are candidates, as are axiom sets for units and measures. The mechanisms for combining axiom sets generally involve axioms sets that contain map and type variables. Models with variables can be instantiated with closed terms. On the other hand, a model such as vehicle test leave units and measures unspecified which effectively means that these theories contain variables which are bound to produce concrete theories.

6. Conclusions

By viewing engineering models as embedded within axiom sets this work is an application of the axiomatic method. An engineering model, whether it is for manufactured products or biomedicine, is intended to capture the relevant structures which realize the description provided by the engineering model. Embedding an engineering model as an axiom set within a logic-based formalism provides a way to precisely study the structures which model the axioms, in the logician's sense of model. When engineering models are translated into axiom sets, the engineer's simulation becomes the logician's model. While the names of the concepts in engineering and logic are different the concepts are the same. What an engineer calls a model a logician calls an axiom set; what a logician calls a model an engineer calls a simulation. This equivalence of concepts leads to application of well-established methods of logic to engineering. These methods include the precise relationship between derivability and model theoretic truth. The connection between provability and truth underlies the correct application of automated reasoning to engineering problems. Engineering modelling languages which have built on accumulated experience of constructing and using models provide fundamental insight into how to model both structure and behavior. The Algos representation of structure and behavior builds on these language constructions. Fortunately, they are representable within a topos-based framework. Engineering modeling is also a source of interesting problems which have apparently not been considered by logicians. One of many examples is how to build engineering model axiom sets which characterize all of the allowable variants for some manufactured product.

Engineering modelling and its axiomatics are for everyday use in design and analysis. 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. The conclusions reached are derived within the theory. This is a different state of affairs from that described by Jaakko Hintakka (Hintikka 2011). Hintakka asserts that when axiom sets are used for applications in science, reasoning is most often about the theory generated by the axioms rather than within the theory of the axiom set. This assertion does not hold for engineering axiomatics. It is true that one makes use of proofs of mathematical soundness, characterizations of conditions which imply decidability of satisfaction, and other meta-theoretical results. The primary use of a theory generated by an Algos axiom set is precisely to reason within the theory.

In applications the first question is whether the axioms capture the intended systems. Criteria which have been suggested for evaluating the effectiveness of a model for an application are: (1) correctness, no model constraint is wrong, i.e., it does not accord with the ground truth of the domain of application, (2) precision, non-intended models are excluded, and (3) accuracy, negative examples are excluded. In engineering applications the term ground truth corresponds to the relationship between descriptive models and the real world. As modelling is prevalent in engineering there are established practices for validating simulations of a product interacting with its operating environment. Simulation is used extensively as part of aircraft verification for flight performance. The process of verifying that a product satisfies its requirements often depends on verification that an engineering model simulation of the product behaves correctly operating in a model of the operating environment. If one has a validated simulation model in which some statement is false, then the statement does not follow from the theory of the model and so is not true in the theory.

Many engineering problems when viewed within the context of axiom sets translate into questions of consistency of the axiom set. In design development adding a component to a product design may violate constraints of the component being added or the design. The design, as represented as an axiom set, become inconsistent. For example, when adding an electrically powered device to a design, the presence of the device may overload the ability of the system to provide power. On the other hand, the specifications for a device to be added to a product may be inconsistent with the device's use in the product. Many common manufactured product recalls can be attributed to this kind of inconsistency. While the inference involved in engineering problems is often simple, the size and complexity of the problems is beyond what can be done effectively manually. Hence the need for automated analysis and an understanding of its limits.

Much engineering practice is concerned with analysis of the behaviour of a system within its operating environment. This analysis may be performed to make design decisions during the design process. The analysis is often simply to determine product capability. Can a sensor mounted on a moving air vehicle recognize and identify some stationary or moving object under specified conditions. The Algos approach is introduced in Section 4. Under the simplifying assumptions made behaviour is restricted to modifying maps which are attributes are projections onto a type X. In the language of computer science $a.f$ is a variable. The notation $f : Var(Y)$ is sometimes used. Behaviour analysis is often in the form of determining if

$$P(\mathbf{s}) \Rightarrow Q(f(\mathbf{s})) \tag{275}$$

follows from the theory generated by the axiom set. Here $\mathbf{s}$ is a tuple variable, $P(\mathbf{s})$ are preconditions and $Q(f(\mathbf{s}))$ is a post condition applied to the result of an action. The map f has as its domain type the state space. The action of f is defined for time instances by equations which connect the attributes and by actions performed by state machines whose domain includes the attributes of X. This translates into whether the theory derives the implication.

6.1. *Is a Single Formalism Needed?*

From the engineering viewpoint is an inclusive formalism needed? Each application generates its own theory. In engineering practice multiple languages are generally used. For example, 3D CAD models are used for the geometry of a product and specific material models are also used. A theory may include, or use, multiple additional theories of varying degrees on generality. When multiple engineering models are used with out full semantic integration there is a spectre of false conclusions being reached from semantically incompatible theories. In the general case this calls for logic preserving functors between individual application categories. However, with an integrating formalism such translations are feasible.

6.2. *Formalism Choices*

Practically finding a suitable logical formalism and embedding engineering models as axiom sets has proven difficult, as the term constructions needed are richer than what is possible in a Standard Formalism. The arguments regarding the inability of a standard formalism (first order with a single sort) have been born out by experience. The 2-amino acids are a case in point. Formalisms which provide mathematical soundness, set theory expressiveness, and which have been integrated with computationally tractable reasoning, limit the choices. The choices appear to be a language based on set theory, type theory, or topos theory. In the sense that Algos is an algebraic form of set theory, it generates a topos, and its logic is a type theory; it combines the virtues all three of these candidates. Algos includes constructions that allow a structure to have an operation as a component and to represent state and state change. These constructions are definable within Algos/topos logic.

Lawvere's hypothesis that abstract categories can be used to represent mathematics and physics extends to engineering models. The engineering model examples investigated here can be embedded as an axiom set which generates a topos. Experience with building engineering models in a number of application domains has convinced the author that a category theory based formalism works for engineering modelling as well as mathematics and physics. Categories are a natural model theory for engineering models when embedded as axioms in a "categorical" logic. A category provides a directly interpretation for map and type constructions in axiom sets for biomedical modelling and manufactured products. The validation of this hypothesis has practical consequences for engineering.

6.3. *Engineering Topos theory*

As defined in (Lambek and Scott 1980) a topos is a Cartesian closed category which has subobject classification. In topos theory, as opposed to set theory the language constructions are very algebraic which simplifies reasoning. The first order axioms for topos theory developed by Lawvere (Lawvere 1964) use the existential quantifiers. The Algos axioms have been engineered from the first order axioms for a topos to be in the form of Horn clause in a multi-sorted logic. This enables using Algos as a computational logic. The engineering consisted of converting the first order axioms with existential quantifiers to be Horn clauses by use of Skolem functions. The set of Algos axioms generates a topos, called the free topos. The theory generated by an Algos axiom set is also a topos which is a quotient of the free topos, called the syntactic topos. Any Algos theory satisfies the axioms for a Cartesian category with power types and subobject classification which is an alternative characterization of a topos. This formalism was implemented in 1980s (Graves and Blaine 1985; Graves and Blaine 1986).

Algos, building on topos theory, provides a well worked out type system in which properties of directed graphs are first order. The use of topos type theory enables the definition of classes and relations which have an axiomatic semantics. Reasoning which in Description Logics has to be model theoretic can be given in the topos context. Algos and topos theory have considerable expressiveness beyond what has been used for the amino acid example. The full set of topos constructions are used in the representation of composite structures which have embedded operations and state. Additional axioms from topos theory (Graves and Bijan 2011) can represent structures which change with respect to time or other kinds of events.

6.4. *Engineering of modelling languages*

Producing an implementable formalism that engineers and scientists can, and will use, is itself an engineering problem with multiple dimensions. This work builds on insights embodied in engineering modelling languages regarding how to represent structure and behaviour. These languages have been developed to be used over a twenty year period. They reflect the experience of their use. Graphical properties play a major role in describing structures. This argues for a computational logic in which graphical descriptions and graphical constraints can be expressed as axioms. Algos has this property. Axiom sets can be graphically developed in a SysML authoring tool. The result can be exported in a linear form suitable for automated analysis and question answering. Figure 2 with its two diagrams is a SysML model. SysML blocks, associations, and part properties directly map to Algos constructions. Even for such a simple example as amino acids, linear syntax places a significant burden on a human user understanding as opposed to the graphical syntax. The amino acid case suggests useful extensions to modelling languages such as SysML. For example the DL class constructions and variables such as the side chain variable would be useful.Long after the implementation the author realized that SysML could be embedded within Algos. This result provided the basis for a practically usable interface (Graves and Bijan 2011).

6.5. *Background*

Algos development begin with an attempt to represent and reason about applications within a first order logic inference engine and proof checker using Set Theory axioms. This proved a very cumbersome task. Set theory axioms were designed to be talked about rather than be used. Typically a minimum number of language constructions are used. This effort led to the idea using axioms which were non-necessarily minimal, but which were physically interpretable and which could be used effectively by an inference engine. Elementary topos theory provided axioms for many term constructions used in applications, and at least some of the axioms could be used as computation rules. One axiom, the subobject classification axiom did not have a clear computational implementation. Algos came about in the attempt to engineer the topos axioms. The subobject classification axiom was replaced by a stronger axiom which is satisfied by Set Theory (see Section 3.3).

The development of Algos and its implementation was done in the late 1970s and early 1980s while the author was a faculty member at San Jose State University. The motivation was to find a logical formalism with expressiveness comparable to set theory and in which reasoning was computationally tractable. The criteria governing the development of Algos was laid out in Section 3. The formalism is intended to be used by engineers and scientists for everyday work. Algos has set theory constructions while staying within a first order formalism. The constructions needed for a variety of applications representable within set theory are also representable within topos theory. The embedding of an Algos axiom set within a topos establishes the mathematical soundness of Algos. At the time that the formalism was developed and implemented only a linear syntax was available. In the author's subsequent experience practical usability for complex applications requires a graphical syntax. Only in the last ten years has good graphical syntax been developed for engineering modelling languages. Fortunately this syntax with extensions can be used for Algos.

Algos has been used to develop axiom sets for structural descriptions which include biomedicine and manufactured products. The axiom sets provably constrain the models so that all minimal models are isomorphic. The descriptive language and representation techniques used in the amino acid and the vehicle test examples are equally applicable in other application domains. The discussion of parts and connections can be generalized to have multiple kinds of parts and connections. As evidenced by amino acids the representation of structural descriptions as axiom sets needs a much more expressive language than DL or a single sorted Logic Programming framework. The advantage of the topos logic framework is that the axioms for map and type constructions are worked out in an algebraic form suitable for use by a computation system.

References

Ackermann, W., Solvable Cases of the Decision Problem. Studies in Logic and the Foundations of Mathematics. North-Holland, 1954.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., The description logic handbook, Cambridge University Press, 2007.

Bell, J., From absolute to local mathematics, Synthese, Springer, 1986.

Bell, J., The Development of Categorical Logic, Handbook of Philosophical Logic, Volume 12, Springer, 2005.

Berardi, D., Calvanese, D., and De Giacomoa, G., 2005.

Dumontier, M., Describing chemical functional groups in OWL-DL for the classification of chemical compounds. OWLED, 2007.

Coquand,T., Huet, G., The calculus of constructions, Information and Computation 76(2/3) (1988), 95-120.

van Emden, E., Kowalski, R., The semantics of predicate logic as a programming language. Journal of ACM, Vol. 23, 733742, 1976.

Knight, S., Mardare, R., Panangaden, P.: Combining Epistemic Logic and Hennessy-Milner Logic. ;In Logic and Program Semantics(2012)219-243

Graves, H., Blaine, L., Algorithm Transformation and Verification in Algos, Third International Workshop on Software Specification and Design. IEEE Computer Society Press. August 1985.

Graves, H., Blaine, L., The Algos Computational System, Proceeding of the European Conference on Computer Algebra, Springer, 1985.

Graves, H., Horrocks, I., Application of OWL 1.1 to Systems Engineering, OWL Experiences and Directions April Workshop, 2008.

Graves, H., Representing Product Designs Using a Description Graph Extension to OWL 2. OWL Experiences and Directions October Workshop, 2008.

Graves, H., Guest, S., Vermette, J., and Bijan, Y., Air Vehicle Model-Based Design and Simulation Pilot, Simulation Interoperability Workshop (SIW). Spring 2009.

Graves, H., Bijan, Y., Using formal methods with SysML in aerospace design and engineering, Annals of Mathematics and Artificial Intelligence, Springer 2011.

Graves, H., Integrating Reasoning with SysML, INCOSE International Symposium (IS 2012), 2012.

Harrison, J., and Thery, L.: Extending the HOL theorem prover with a computer algebra system to reason about the reals, in Higher Order Logic Theorem Proving and its Applications: 6th International Workshop, pp. 174-184, Lecture Notes in Computer Science 780, Springer (1993).

Hintikka, J. "What is the axiomatic method?." Synthese 183.1, 69-85 (2011).

Krdzavac, N., Bock, C., Reasoning in Manufacturing Part-Part Examples with OWL 2, National Institute of Standards and Technology, NISTIR 7535 2008.

OWL 2 Web Ontology Language, W3C Working Draft 11 June 2009.

Hastings, J., Dumontier, M., Hull, D., Horridge, M,. Representing chemicals using OWL, description graphs and rules, 2010.

Horrocks, I. Kutz, O., and Sattler, U., The Even More Irresistible SROIQ, in Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57-67, American Association of Artificial Intelligence Press, 2006.

Kuske, S., , Towards an integrated graph-based semantics for UML, Software and System Modeling, 2009.

Lambek, J., Scott, P. J., Aspects of Higher Order Categorical Logic, Mathematical Applications of Category Theory, American Mathematical Society, 1980.

Lambek, J., Scott, P. J., Introduction to higher-order categorical logic, Cambridge University Press, 1986.

Lawvere, F. W., An elementary theory of the category of sets Proceedings of the National Academy of Sciences, 1964.

Magka, D., Motik, B., Horrocks, I., Modelling Structured Domains Using Description Graphs and Logic Programming, DL2012.

Martin-Lof, P., constructive mathematics and computer programming. Logic, Methodology and Philosophy of Science, 1982.

Motik, B., Cuenca Grau B., Sattler, U., Structured objects in OWL: Representation and reasoning. Proceeding of the 17th international conference on World Wide Web, 2008.

Suppes, P., Representation and Invariance of Scientific Structures. Stanford, CA: CSLI Publications, 2002.

OMG Systems Modeling Language (OMG SysML), V1.2, 2010.

Zarba, C., Lecture notes on Decision Procedures, 2007.

Vitae

Dr. Henson Graves is a principal of Algos Associates, a consulting firm. He is a Lockheed Martin Senior Fellow (Emeritus) and a San Jose State University Professor Emeritus in Mathematics and Computer Science. He has industry experience in the development of complex aerospace systems. Technical areas include automatic code generation, modelling and simulation based design, and design analysis. He has published papers in mathematics, computer science, product engineering, ontology, and Description Logic. Dr. Graves is the INCOSE Model Based System Engineering (MBSE) Ontology Action Team lead. His current technical interests are in application of mathematical formalisms to modelling and simulation in engineering and product development.