Formal Architectural Modelling

Roger Bishop Jones

Abstract

Some notes on methods for formal verification of system architecture in the context of Model Based Systems Engineering.

Created 2020-12-26

Last Change Date: 2021-01-10

http://www.rbjones.com/rbjpub/pp/doc/t057.pdf

Contents

1 Introduction

This PDF is hyperlinked to facilitate navigation around the document.[1]

These are notes I put together while looking at the ideas presented in [2, 3], particularly on the idea of "model theoretic" approaches to giving mathematical precision to architecture definitions. I have had some difficulty in getting a clear understanding of what is intended. Probably partly because of knowing too little of important background context, particularly in relation to the relevant standards work of ISO/IEC WG42 and a fuller acquaintance with UML, SysML and Common Logic. Such as I have in in section 2

The second part, in section 3, is a presentation of how I think about system architecture and the problem of reasoning mathematically about whether some formally defined architecture would suffice to build a system meeting formally specified requirements. The formal parts of this account is undertaken in Higher Order Logic (HOL) using the ProofPower-HOL dialect of Church's Simple Theory of Types, supported by the ProofPower toolset.

This approch to the development of secure and safety critical systems was used by ICL in the decade around 1990, at first using the Cambridge HOL proof tool[5] and later the ICL ProofPower tools. A simplified example of the application of these methods to secure systems and discussion of some of the methodological issues may be found in [8].

Work in this vein was provoked by the US DoD, who had been convinced by their academic consultants at the time that the best way to ensure that the computer systems handling highly classified materials were secure would be to mathematically prove that they are, and that, because of the great complexity of such systems, no proof could be trusted unless machine verified. I understand that the USA advised the British government that if we were to continue to have access to their confidential intelligence, then our computer systems would have to operate to similar standards, and a program of work was then sponsored by GCHQ/CESG to establish a UK capability to develop and formally verify secure computing systems. Similar considerations (though less exacting) were later applied to safety critical defence procurements in the UK and embodied in DEF STAN 00-55.

The main problem with this idea is that, even with the best available computer support, formal mathematical proof is very labour intensive. As soon as the systems get non-trivial this is an insuperable obstacle. Since then proof technology has advanced, and the ProofPower tools which I still use, now fall short of the state of the art except possibly in certain areas. Computer verified proofs have now been undertaken for some of the most difficult mathematical results such as the four colour theorem, but I believe are still not practical for other than relatively simple computer systems. This will probably remain the case until we have serious machine intelligence, which I think is still over the horizon.

There was a book [10] published around this area by Donald Mackenzie, then I think a professor of sociology at Edinburgh University. He sent a research associate to interview me, and the transcript of the interview can found online [9].

I personally am an enthusiast for the mechanisation of formal mathematics and its application to engineering design, but I am sangine about its feasibility pro-tem. I do find that when I am able to treat formally a topic which had previously had a less rigorous treatment, lots of valuable detail emerges which was previously obscure. Nevertheless, it is not clear to me that such methods could appear in a general standard for architectural modelling, though I might be less sceptical if I new more about the relevant context, particularly SysML and Common Logic. Some of the work we

[1] Links within the document are coloured blue, external URLs are coloured red. If you read the document in Acrobat Reader on a mac, command left-arrow is the back key. You can get a back arrow on the toolbar by: right click on toolbar -> page navigation tools -> previous view

undertook with ProofPower does support the formal treatment of graphical notations. The ClawZ project, undertaken by Lemma1 under contract to QinetiQ delivered several iterations of a program which translated simulink diagrams into formal specifications in the Z language, which could then be processed by ProofPower[2].

[2]Project documentation, much of it formal specifications in ProofPower-Z, is available at https://www.lemma-one.com/clawz_docs/clawz_docs.html.

2 Architectural Design using Model Theory

Though I appreciate that an understanding of models contributes to a working competence in the use of formal notations, I have so far been unable to get a grip on the proposed use of model theory in the presentation of mathematical models of system architecture.

I do have a working understanding some of the ways in which formal notations can be used in systems architecture and design, and would if necessary be able to give less formal prose descriptions of the same mathematical models, in neither case would I expect to see any explicit mention of model theory.

I think for me to properly engage with these ideas I would need to see something like the ADAS example worked through both by the methods which I understand (so that I do actually understand the problem being modelled) and also using the model theoretic methods advocated. That's probably not going to happen!

I was puzzled by the insistence on *relational* structures, which I would expect to make complex modelling more difficult and less easy to understand. Some more detailed technical observations in that area follow.

2.1 Relational Structures

I'm puzzled why the constraint to *relational* structures. This would simplify metatheory, but at the expense of making the specifications more complex and opaque. Thus the relation: $a^2 + b^2 = c^2$, since it has four instances of function application would have to be rendered using the corresponding relation symbols and existential quantifications:

$$\exists w\ x\ y\ z \bullet\ Re(a,\ 2,\ w) \wedge Re(b,\ 2,\ x)\ \ \wedge Re(c,\ 2,\ y) \wedge Ra(w,\ x,\ z) \wedge z = y$$

In which Re is the relationship of exponentiation ($Re(x, y, z) \Leftrightarrow x^y = z$) and Ra that of addition ($Ra(x, y, z) \Leftrightarrow x + y = z$).

Commenting specifically on the interpretation given in [3] IIIB, there are some difficulties. Sentence (1), assuming that P_1 and P_2 are intended to be relations, is not a syntactically valid sentence of the predicate calculus, since it applies the relationship '=' to two atomic formulae. It would be valid if '$\Leftrightarrow$' were used instead of '=' or if P_1 and P_2 were function names rather than relation names, but then one would not have a strictly *relational* structure.

For a model of the concept of orthogonality, I should myself be looking for something talking about vectors, rather than a relationship between scalars. In a vector space the notion of orthogonality is definable thus:

$$orthogonal\ (x,y) \Leftrightarrow x.y = 0$$

Here, though "orthogonal" is itself a relationship, the language we have used involves functions, and so goes beyond the first order predicate logic into the first order functional calculus.

Even for hard core model theory as a theoretical discipline, the constraint to relational signatures does not seem to be thought a worthwhile simplification of the theory, and, for example, in Hodges[6] we are immediately given a definition of structure which includes operations (functions) as well as relations, relational structures receiving only a passing mention.

In IIIC, SYMBOL and REFERENT are said to be related by a model theoretic *interpretation map*. In [6] interpretation is of one (model theoretic) structure in another, and so neither is syntactic and the two are similar kinds of abstract object. Is it intended that both SYMBOL and REFERENT are model theoretic structures, or is some other notion of interpretation intended?

3 A Formal Model of Architectural Specification and Verification

3.1 Architecture and Ontology

In what follows I present an approach to architectural modelling and verification, in which the primary aim is to cast some light on how mathematical models can be used to define an architecture in a sufficiently precise way as to permit the correctness of the architecture (as a way of realising a system meeting a clearly defined requirement) to be mathematically proven.

In many of the established methods for formal development, the behaviours of the system are modelled formally, but there is no single mathematical entity which represents the system as a whole. Consequently, it is not possible to state in the formal notation the propositions of greatest interest. The implementation method may be said to be *by refinement*, and the correctness of refinement will then be established by proving a number of separate propositions (verification conditions) which are syntactically generated from the script of the formal specification. There are many variations on this theme.

If a sufficiently expressive formal notation is used (or even with appropriate use of informal mathematics), it is possible to have a single type of entity providing models for the class of systems under consideration, to formulate the requirements as a property of such entities, and to clearly state and mathematically prove the correctness of the model corresponding to some proposed design or implementation of a system meeting the requirement.

This is an advantage arising from thinking of a mathematical or formal model as an abstract entity, rather than as a body of text. The method, which is exemplified below in an abstract treatment of architectural modelling (and in many more substantial exemplars elsewhere), assumes such an ontologically complete formal treatment.

For the sake of generality nothing is assumed, in the following model, about the nature of the system being modelled, except insofar as it is presumed that the system will be complex and its behaviour will effectively be determined by that of its various subsystems or components and by the manner in which these have been combined into a whole. The problem of realising the required behaviour may therefore be approached by identification of a collection of parts, description of how these parts are to be assembled into the whole, and stipulating the requirements on those parts which suffice to ensure that the resulting whole behaves in a manner consistent with the requirements for the system as a whole.

No assumptions are made about the nature of the requirements, either for the whole system, or for the subsystems or components. We therefore consider these requirements to be properties (i.e. BOOLean valued functions) of mathematical entities which are considered as models of potential candidates. (This is convenient because ProofPower supports a Higher Order Logic, but a typed set theory would be equally expressive and in that context requirements would be represented as a set.) The definition of these properties is part of the system requirements analysis (at the top level) and the architectural design process (at the next level). The kinds of abstract object which are candidates are determined in the process of formalising the requirement. In order that the requirement can be thus defined, it must be the case that we are talking of the system model as being a single abstract entity, rather than, say, a collection of formal definitions.

An important advantage of this method of mathematical modelling, by contrast with methods such as those advocated for use with the specification languages Z and VDM, is that the set of acceptable implementation models is explicit. If acceptable implementations are obtainable by a process of refinement, then the meaning of the specification is dependent on exactly what refinement process is deemed acceptable, and there may be questions about what kinds of refinement will preserve critical properties.

3.2 Model Theory and Higher Order Logic

I'm not intending here a scholarly account of models of higher order logic, but rather an informal discussion at level appropriate for someone trying to understand how ProofPower-HOL works for systems design[3].

The specifications which follow are formalised in ProofPower-HOL, a polymorphic variant of Church's Simple Theory of Types [1], a language and logic closely based on that of the Cambridge HOL system designed and implemented by Gordon et.al.[5] (originally for use in the verification of digital hardware). An account of the semantics of this formal logic is "model theoretic" insofar as it describes the models of the theory. This is quite a bit different to the model theory of first order logic.

It is worthwhile to have an elementary understanding of this to fully understand specifications in Higher Order Logic, but the main flavour of writing specifications, or constructing and reasoning about "mathematical models" in this system is rather closer to that of writing a program (in a somewhat exotic programming language) than of doing model theory. By that I mean, that a specification in this system feels like (with good reason) a series of definitions, and intuitively one is thinking of the nature of the particular objects one is defining rather than considering any structures which may be formed with them.

Partly this strictly definitional idiom is motivated by the desire to ensure that the specification is consistent, and hence that the specification as a whole does indeed have a model. That requirement is fulfilled by a style of formal mathematics similar to that advocated by Frege [4] and Russell [11, 12]. The idea here is most clearly articulated by Gottlöb Frege's ('logicist') dictum that:

> Mathematics = Logic + Definitions

Since Frege, philosophers (other than hard core logicists) have generally considered this to require a notion of logic somewhat stronger than they were willing to accept, but a more general acceptance has nevertheless accrued to the variant:

> Mathematics = Set-theory + Definitions

To which one may add that Higher Order Logic with infinity and choice, is a sufficient alternative to set theory to cope with a large part of mathematics (including more than enough for systems engineering).

Higher Order Logic, construed under the 'standard' semantics, is *quasi-categorical*, i.e. it has only one model (up-to-isomorphism) at any cardinality. A typical specification will involve a large vocabulary of interdefined names, and the general ethos is more similar to that of programming languages, in which names, for data structures, functions or procedures, are *defined* rather than axiomatically constrained. This contrasts with the more algebraic flavour typical of model theory, in which the signature of the models is constrained by a set of axioms en bloc.

The rationale for this is twofold. First, this does make specifications easier to understand. Secondly, it enables consistency of the specification to be ensured. Normally a discipline of specification by *conservative extension* is adopted. This means that, each additional 'axiom' is introduced to constrain one or more new constants, and it is established that the extension to the theory thus obtained does not eliminate any of the previous models of the theory, but just extends the signature with new values. This is done by proving prior to the extension (often but not always automatically) that there already

[3] For a full and formal account of the semantics of ProofPower-HOL see Arthan[13, 14], also available at https://www.lemma-one.com/ProofPower/specs/specs.html

exist values which might be given to the new names which satisfy the axiom used to introduce them. The axiom which introduces the new names need not uniquely determine their values however. This allows for what I like to call 'loose specifications', but which have sometimes been called 'partial definitions' to the derision of Hodges in [7] who considers this nonsensical. Notwithstanding that opinion, these conservative but incomplete constraints are registered in the theory as 'definitions', whereas, if a non conservative extension is undertaken (which is very rare and tends to be deprecated) that is shown baldly as an axiom.

The primitive logic is very small and has a minimal signature, but most specifications will make use of a library of theories defining a range of mathematical entities or data types with appropriate relations, functions and operations over them, and even a small specification such as the one below will have a fairly rich signature just because of the formal context in which it occurs. However, specifications, including those in the libary, are structured into a heirarchy of HOL theories, using that latter term in this context as a concept not dissimilar to a module in a programming language. One may therefore think of each separate HOL theory as being specified in a fixed linguistic context and think of the models of that theory separately as structures whose signature corresponds to the various names introduced in that HOL theory.

Such theories can be separately listed, and the signature and the constraints imposed by the theory on the values of the names in the signature can be read off the theory listing. Of course, these are not relational sinatures or structures. This may be seen for the specification undertaken below which is listed in appendix A. The signature of the theory may be read from the earlier parts of the theory listing, notably, the type abbreviations, type definitions and constant names shown with their type. The constraints, determining which structures with that signature are models of the theory, are shown either as *definitions* in the case that they have been shown to be conservative, or otherwise as *axioms*.

3.3 The Formal Model

The following simple formal model is provided using the specification language ProofPower-HOL, an implementation of Higher Order Logic supported by the ProofPower proof tool. This is not an example of architecture specification, but a model of the process, in which architectural models are mentioned, and reasoned about, but not exhibited or exemplified.

SML

```
open_theory "hol";
force_new_theory "arch";
set_pc "hol1";
```

3.3.1 System Requirements

We place no constraint on the type used to model the system. The type used must yield sufficiently precise models for it to be definite whether or not any instance of that type does or does not model a system meeting the requirement. The type will therefore be determined during the requirements phase of the project. In the following ProofPower-HOL model this type will be represented by the type variable *'sys*.

A statement of requirement is a property of systems or subsystems. This is reflected in the following *type abbreviation*, in which the type abbreviation REQ is parameterised by the type of system under consideration.

SML
```
declare_type_abbrev("REQ",
      ["'sys"], ⌜:'sys → BOOL⌝);
```

It may be desirable to distinguish between critical or key requirements, each of which will be a predicate or propositional function, in which case the system requirements would be modelled as a pair of such predicates. For present purposes we will address only the critical requirements, which are the ones which may warrant detailed formal verification.

3.3.2 Architecture

An architecture is the highest level of system design, exhibiting the top-level structure of a system implemented to meet a previously defined system requirement.

An architecture shows how systems of the required kind can be constructed. It does so by identifying a number of components, stipulating the requirements which each of those components must satisfy, and providing a method for fitting together the components to forn a system of the required kind. We assume that the subsystems or components of an architecture are of uniform type and are given names so that we can ensure that the correct requirement is applied to each component of the structure, and that the component is fitted in the right place into the whole. Indexed collections are modelled using a set of names and a function from names to values, as seen in the following type abbreviation.

SML
```
declare_type_abbrev("IC",
      ["'el"], ⌜:(STRING SET × (STRING → 'el))⌝);
```

The "fitting together" is modelled by a construction function which takes as an argument models of all the subsystems and delivers a model for the system which would result from using those components. The following type abbreviation defines a type for such constructions, as a function from an indexed set of (models of) components to (a model of) the system built from from those components in the prescribed way. The type abbreviation here shows that the type of such a constructor depends on the type of components (which we are supposing uniform, which can always be ensured by using a disjoint union if necessary) and the type of the resulting system.

SML
```
declare_type_abbrev("CST", ["'sys", "'comp"],
                          ⌜:'comp IC → 'sys⌝);
```

An architecture is then the combination of such a construction with the specifications of the components necessary for the resulting system to meet its requirement. The component specifications are also supplied as an indexed set, the indexes (names) corresponding to those used in the indexed set of components required by the construction function.

SML
```
declare_type_abbrev("ARCH", ["'sys", "'comp"],
                          ⌜:('comp) REQ IC × ('sys,'comp)CST⌝);
```

The architecture is "correct" if applying the construction to components which meet their requirement will always result in a system which satisfies the system requirement. Note that this is a higher order function and that in Church's variant of Higher Order Logic, function application is rendered by juxtaposition and does not require brackets round a single argument. Application of the following

function to a system requirement yields a property of architectures, viz. the property of being a correct way of building a system meeting the requirement.

HOL Constant

$$\$\textbf{\textit{correct}} : ('sys)REQ \to ('sys,'comp)ARCH \to BOOL$$

$$\forall\ arch\ req \bullet\ correct\ req\ arch \Leftrightarrow$$
$$\quad \forall comps{:}'comp\ IC \bullet$$
$$\qquad (\forall name{:}STRING \bullet\ name \in Fst\ (Fst\ arch) \Rightarrow name \in Fst\ comps$$
$$\qquad \wedge\ (Snd\ (Fst\ arch)\ name)(Snd\ comps\ name))$$
$$\qquad \Rightarrow req\ (Snd\ arch\ comps)$$

3.3.3 Composition

This process is not exclusive to the top level, but can be iterated to give ever more detailed design. At any stage the sequence of design steps undertaken can be gathered together to yield a single architecture for the system as a whole which reflects all the detail thus far elaborated. If each step in that process is correct, then the resulting composition will also be correct. We would therefore expect to be able to define in terms of this model the effect of composing two design steps where one designs a component or subsystem required by a previous design step, and then to be able to prove that if the two design steps are correct then so will be the result of the composition.

To reason about this kind of iterated composition it will be necessary to assume that sucessive stages in design composition yield systems of the same type (otherwise the result of a single composition would not have uniformly typed components).

HOL Constant

$$\$\textbf{\textit{compose}} : ('sys,'comp)ARCH \to STRING \to ('comp,'comp)ARCH \to ('sys,'comp)ARCH$$

$$\forall\ (sarch{:}('sys,'comp)ARCH)\ n\ (carch{:}('comp,'comp)ARCH) \bullet\ compose\ sarch\ n\ carch =$$
$$\ let\ ((snames,\ sprops),\ scst) = sarch$$
$$\ in\ let\ ((cnames,\ cprops),\ ccst) = carch$$
$$\ in\ let\ newnames = (snames \backslash \{n\}) \cup cnames$$
$$\ in\ let\ newspecs = (newnames,\ \lambda s \bullet\ if\ s \in cnames\ then\ cprops\ s\ else\ sprops\ s)$$
$$\ in\ let\ newcst = (\lambda cis{:}'comp\ IC \bullet$$
$$\qquad let\ ssn = ccst\ (cnames,\ Snd\ cis)$$
$$\qquad in\ scst\ (snames,\ \lambda m \bullet\ if\ m = n\ then\ ssn\ else\ (Snd\ cis)\ m))$$
$$\ in\quad (newspecs,\ newcst)$$

The composition method meets the following condition:

$$composition\_correct\_thm = \vdash \forall req\ sa\ ca\ n \bullet$$
$$\qquad (Fst\ (Fst\ sa)) \cap (Fst\ (Fst\ ca)) = \{\}$$
$$\qquad \wedge\ correct\ req\ sa$$
$$\qquad \wedge\ n \in (Fst\ (Fst\ sa))$$
$$\qquad \wedge\ correct\ (Snd\ (Fst\ sa)\ n)\ ca$$
$$\qquad \Rightarrow correct\ req\ (compose\ sa\ n\ ca)$$

This has been proven, and the proof script is part of the source of this document, but the printing has been suppressed. This document is a literate script from which the formal materials can be extracted and run through ProofPower. The resulting theory can then be listed, and is shown in appemdix A.

A The Theory arch

A.1 Parents

hol

A.2 Constants

correct $'sys\ REQ \to ('sys, 'comp)\ ARCH\ REQ$
compose $('sys, 'comp)\ ARCH \to STRING \to ('comp, 'comp)\ ARCH \to ('sys, 'comp)\ ARCH$

A.3 Type Abbreviations

$'sys$ ***REQ*** $'sys\ REQ$
$'el$ ***IC*** $'el\ IC$
$('sys, 'comp)$ ***CST*** $('sys, 'comp)\ CST$
$('sys, 'comp)$ ***ARCH*** $('sys, 'comp)\ ARCH$

A.4 Definitions

correct

$$\begin{array}{l}\vdash \forall\ arch\ req \\ \bullet\ correct\ req\ arch \\ \quad \Leftrightarrow (\forall\ comps \\ \quad \bullet\ (\forall\ name \\ \quad\quad \bullet\ name \in Fst\ (Fst\ arch) \\ \quad\quad\quad \Rightarrow name \in Fst\ comps \\ \quad\quad\quad\quad \wedge\ Snd\ (Fst\ arch)\ name\ (Snd\ comps\ name)) \\ \quad\quad \Rightarrow req\ (Snd\ arch\ comps))\end{array}$$

compose

$$\begin{array}{l}\vdash \forall\ sarch\ n\ carch \\ \bullet\ compose\ sarch\ n\ carch \\ \quad = (let\ ((snames, sprops), scst) = sarch \\ \quad\ in\ let\ ((cnames, cprops), ccst) = carch \\ \quad\quad in\ let\ newnames = snames \setminus \{n\} \cup cnames \\ \quad\quad\ in\ let\ newspecs \\ \quad\quad\quad = (newnames, \\ \quad\quad\quad\ (\lambda\ s \\ \quad\quad\quad\ \bullet\ if\ s \in cnames \\ \quad\quad\quad\quad then\ cprops\ s \\ \quad\quad\quad\quad else\ sprops\ s)) \\ \quad\quad in\ let\ newcst\ cis \\ \quad\quad\quad = (let\ ssn = ccst\ (cnames, Snd\ cis) \\ \quad\quad\quad\ in\ scst \\ \quad\quad\quad\ (snames, \\ \quad\quad\quad\quad (\lambda\ m \\ \quad\quad\quad\quad\ \bullet\ if\ m = n\end{array}$$

$$
\begin{array}{l}
\qquad\qquad \mathit{then}\ \mathit{ssn} \\
\qquad\qquad \mathit{else}\ \mathit{Snd}\ \mathit{cis}\ m))) \\
\mathit{in}\ (\mathit{newspecs},\ \mathit{newcst}))
\end{array}
$$

A.5 Theorems

composition_correct_thm

$$
\begin{array}{l}
\vdash \forall\ \mathit{req}\ \mathit{sa}\ \mathit{ca}\ n \\
\quad \bullet\ \mathit{Fst}\ (\mathit{Fst}\ \mathit{sa}) \cap \mathit{Fst}\ (\mathit{Fst}\ \mathit{ca}) = \{\} \\
\qquad\quad \wedge\ \mathit{correct}\ \mathit{req}\ \mathit{sa} \\
\qquad\quad \wedge\ n \in \mathit{Fst}\ (\mathit{Fst}\ \mathit{sa}) \\
\qquad\quad \wedge\ \mathit{correct}\ (\mathit{Snd}\ (\mathit{Fst}\ \mathit{sa})\ n)\ \mathit{ca} \\
\qquad \Rightarrow \mathit{correct}\ \mathit{req}\ (\mathit{compose}\ \mathit{sa}\ n\ \mathit{ca})
\end{array}
$$

B Bibliography

[1] Alonzo Church. A formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 1940.

[2] C. E. Dickerson. Towards a logical and scientific foundation for system concepts, principles, and terminology, 2008.

[3] Charles E. Dickerson, Michael K. Wilkinson, Eugenie Hunsicker, Siyuan Ji, Mole Li, Yves Bernard, Graham Bleakley, and Peter Denno. Architecture Definition in Complex System Design Using Model Theory, 2020. arXiv eprint 1909.06809.

[4] Gottlob Frege. *The Foundations of Arithmetic*. Basil Blackwell. 1980.

[5] Michael J.C. Gordon and Tom F. Melham, editors. *Introduction to HOL*. Cambridge University Press. 1993.

[6] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press. 1997. ISBN 0-521-58713-1.

[7] Wilfrid Hodges. Model Theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2020 edition, 2020. https://plato.stanford.edu/archives/win2020/entries/model-theory/.

[8] Roger Bishop Jones. Methods and Tools for the Verification of Critical Properties, 1992. Also (an updated version) at http://lemma-one.com/ProofPower/doc/wrk050.pdf.

[9] Roger Bishop Jones. An Interview with Roger Jones. *RBJones.com*, 2010. http://www.rbjones.com/rbjpub/www/papers/p006.pdf.

[10] Donald MacKenzie. *Mechanising Proof - Computing, Risk and Trust*. MIT Press. 2001. ISBN 0-262-133937-8.

[11] Bertrand Russell. *The Principles of Mathematics*. George Allen & Unwin. 1903.

[12] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press. 1910-1913.

[13] ds/fmu/ied/spc001. *HOL Formalised: Language and Overview*. R.D. Arthan, Lemma 1 Ltd., http://www.lemma-one.com.

[14] ds/fmu/ied/spc002. *HOL Formalised: Semantics*. R.D. Arthan, Lemma 1 Ltd., http://www.lemma-one.com.

C Index of Formal Names