Relating Z and First-Order Logic

Andrew Martin[⋆]

Oxford University Software Engineering Centre
Computing Laboratory, Wolfson Building
Parks Road, Oxford OX1 3QD, UK.
apm@comlab.ox.ac.uk

Abstract Despite being widely regarded as a gloss on first-order logic and set theory, Z has not been found to be very supportive of proof. This paper attempts to distinguish between the different philosophies of proof in Z. It discusses some of the issues which must be addressed in creating a proof technology for Z, namely schemas, undefinedness, and what kind of logic to use.

1 Introduction

The Z notation [26] is gaining widespread acceptance as a useful means of specifying software systems. Tool support for Z, though given impetus by the Z standardization activity [20], is quite varied, especially in the area of proof. There appears to be little consensus about what proving properties of Z specifications actually means. Different people seem to understand Z and proof differently.

Z has been widely regarded as a sylized form of classical first-order logic and set theory, with a simple type system (but this is not a universal view; see below). The schema language complicates this relationship considerably. As the semantics of schemas has become more complex, it has been increasingly unclear whether Z needs its own logic, or whether it can be manipulated using a familiar system. In creating proof tools, some authors do not appear to have even asked this question, but have assumed a Z semantics which corresponds exactly with the computational logic of some host system.

Therefore, the language of discourse has become surprisingly clouded. The aim of this paper is to relate the different understandings of the nature of proof in Z, and to explain the issues involved. Automatic translations from one notational system to another—or one logic to another—are of course part of any total logical system for Z. Since Z is a large and rich notation, the verification of those translations becomes an issue itself.

Outline of the paper The following section sets the scene, discussing the notion and purpose of proof in Z. Section 3 defines many terms, according to long-standing logical terminology. Somehow these have not always been followed in the Z community, or have come to mean different things to different people. The

[⋆] Paper written at the University of Southampton.

next section surveys some of the leading approaches to this topic—and points to a more detailed survey elsewhere. Sections 5 and 6 detail the two most basic issues which affect reasoning in Z: schemas, and the issue of undefinedness. Section 7 considers related and future work. Alternative approaches are not based on first-order logic at all. For the future, the key challenge seems to be in finding ways to manage the complexity of Z proofs. The paper finishes with some conclusions.

2 Z, Logical Calculi, and Semantics

Z is quite clearly a logical notation. The central task of writing a Z specification is writing schemas. The main content of a schema is found in its predicate part. This is a collection of sentences of first-order logic. The syntax which Z has settled upon is slightly different from that found in most logical texts, but this is not significant.

Whilst in some philosophical disciplines logical notation may be introduced largely as a means to add precision to statements, in mathematical work, and in particular, in software engineering, it is more usual to add a calculus to the presentation of a logical notation. Logic is not used as a language in its own right, but as a means to facilitate proof activity.

In this regard, Z is unusual, since whilst it is not 'semantics free' it has arisen with a rich language, but little in the way of logic. Spivey [25] presents a denotational semantics, and also elements of a logical calculus for the schema notation. More mundane logical proof rules are assumed and not stated.

This reminds us that there are at least two possible approaches to giving meaning to a piece of formal text: we can give a denotational semantics; a model in some system which is already understood—this is Spivey's main approach (despite the use of Z as the metalanguage). Alternatively, we can give a logical calculus, a system of reasoning which relates terms of the language without interpreting them. Such a logic is a higher-level description, and is, in general more practical.

Of course, many writers of specifications will be concerned neither with the model nor with the logic. They are interested in writing specifications and communicating ideas unambiguously, but not in proving that their specifications are complete, say. Nevertheless, a logical system—however expressed—is an important mental tool in writing specifications. The specifier needs to know whether, for example, writing

$$x \notin \operatorname{dom} f \wedge f' = f \cup \{x \mapsto y\}$$

is equivalent to writing

$$f' = f \oplus \{x \mapsto y\} \quad .$$

The application of the definitions of the various operators is a logical process. Some such questions are easily answered, others will require careful detailed reasoning. The answer depends upon the choice of logical system, and will not usually be answered with reference to the model. Most logics will agree on most answers, but in some cases—particularly where application of partial functions is involved—the answers will vary.

If the Z user *does* wish to undertake more formal reasoning (in calculation of preconditions, for example) then the detailed choice of logical system (whether to support a proof by hand, or by machine) will have a significant effect, potentially on both the ease or difficulty of the proof, and perhaps the outcome.

A number of approaches to proof in Z have been taken as a result of the lack of a 'standard' answer. Some of these are explored below; a comprehensive survey is presented in [18], covering both traditional logical presentations, and computational logics/proof tools for Z.

3 Languages and Logic

For some reason, the language used to discuss Z semantics and logic is frequently at odds with the usual terminology of mathematical logic [19, 8]. This section seeks to establish some definitions.

3.1 Terminology

In this paper, we see the Z notation as a *language* in the logical sense. The study of how Z is to be expressed is a matter of *syntax*. The study of meaning, and the relationship of one Z specification (or predicate) with another—whether via models or logics—is the concern of *semantics*.

We may construct a semantics for Z by associating with each term in the language some object in an underlying structure (typically, a set). A particular semantic construction is a *model* of a particular Z specification (or predicate) if the relationships between terms in the specification also hold in the model (if it makes the predicate *true*). A predicate is *logically valid* if it holds in all models. It is logically valid relative to a specification if it holds in all the models of that specification.

A *logic for Z* is a collection of axioms and rules of inference written using the Z notation. These rules and axioms may be used inductively to define which predicates are *logical consequences* of others. Those which are the logical consequence of the empty set of predicates (or, equivalently, the logical consequence of only the axioms of the logic) are known as *theorems*. Usually, such a logic will allow the use of the definitions in a specification (schemas, generic definitions, etc.), so that a predicate may be a theorem *of a specification*, if it is a logical consequence of the definitions in that specification.

A logic for Z may be proven *sound* in some model by showing that each of its theorems is logically valid. It is traditional to ask whether a logic is also *complete* — i.e. whether every logically valid predicate is also a theorem. A consequence of Gödel's incompleteness theorem is that it is not possible to find a complete logic for Z with respect to any conventional underlying theory of sets.

A different question is to ask whether a logic for Z is complete with respect to the mapping into the underlying set theory: this relies on treating the underlying theory as itself a formal system, rather than a model as above. Such an alternative understanding will be discussed below.

Alternative Views Z has often been seen not as a language, but as a sugared syntax for ZF set theory. In this view, most of the foregoing terminology is not used. Instead, one seeks to exhibit a mapping from Z to an underlying language of sets, and to conduct all one's reasoning in the corresponding theory of sets. Since the core parts of Z are simple set theory, the mapping is sometimes invisible, and one is able to use the normal rules of set theory on Z terms. If a partial inverse for the mapping can be found (it will not be bijective, since the richness of Z syntax is not typically present in the underlying system) then the underlying theory might be said to induce a logic for Z, though this construction has not been explored.

Noteworthy is that this process of translation is *largely indistinguishable* from the process of providing a model for Z, as described above. This may help to explain why the process of Z standardization which began by seeking to provide a logic for Z has lately focussed most of its semantic efforts on the model theory of Z.

In many of the proof tools described in the literature, there is no attempt to describe a logic for Z. Instead, the tool is constructed by 'embedding' Z in some computational logic system. The most successful embeddings use higher-order logic (as described in the HOL tool, or in Isabelle's HOL theory) as the 'host'. The embeddings are on a spectrum from 'shallow' to 'deep', depending on how much of Z's semantic structure is retained in the translation. To produce a deep embedding (i.e. one which retains much of Z's structure) is a non-trivial task. As a result, the question arises of how to validate this translation.

A proof in the underlying theory may be demonstrably sound for models of that theory, but the embedding approach is unable to ask whether this represents a sound proof for Z, since there is nothing against which to judge the translation. The difference between truth and provability clearly borders at some point on the philosophical, and is outside the scope of this paper.

3.2 Theorems

It has long been customary to write conjectures (or theorems) in Z specifications as

$\vdash \mathit{pred}$

the intention being to speculate (or indicate) that *pred* is a logical consequence of the foregoing specification. Various Z paragraphs have been written to the left of the turnstile ($\vdash$)—schemas, declarations and predicates, etc.—the intended meaning being that the foregoing specification, augmented with those paragraphs, has *pred* as a logical consequence, Thus, for example, in Spivey's [26] *BirthdayBook* specification, he demonstrates (without writing $\vdash$) that

$$AddBirthday \vdash known' = known \cup \{name?\}$$

meaning that in the context of this specification, and further, within the scope of *AddBirthday* as a definition, *known* has the property shown.

Spivey avoids use of $\vdash$ in this way, but other authors do not. Since the system of logic to be used in such a proof is usually ill-defined, this is a most remarkable abuse of notation, or else a philosophical oddity even for the most ardent Platonist. In the mentioned texts on logic, and others, the turnstile is used as a metalogical symbol to indicate (conjectured) theoremhood with respect to some particular logical calculus. The calculus is either understood from the context, or its name is supplied as an explicit decoration. For a Z specification, where there is no calculus to be understood from the context, it is far from clear what such a conjecture (or theorem) is to mean.

On the 'alternative view' given above, the calculus is presumably understood to be classical first-order logic and set theory. The variety of available presentations (and as a result, of sets of theorems), especially for the latter, would seem to leave the statement of a theorem in this way dangerously close to meaningless.

The Draft Z Standard [20] has given a meaning for statements of Z conjectural theorems which is entirely independent of any logical calculus. This is of course quite a feat. In fact, the Standard's definition does not touch theoremhood at all, but instead deals with model-theoretic truth. That is, the Standard says that

$$\vdash pred$$

holds in a specification *Spec* exactly when all the models of the specification satisfy the predicate. That is, in the notation of the Standard,

$$\{\!|\ Spec\ |\!\}^{\mathcal{M}} \subseteq \{\!|\ pred\ |\!\}^{\mathcal{M}} \quad .$$

This form of proposition is more traditionally written

$$Spec \models pred \quad .$$

As we have observed, one might expect a logic for Z to be sound—that is, that every theorem of a specification is true in every model of that specification (or, '$\vdash$' $\Rrightarrow$ '$\models$'). In pure first-order logic, the converse property (completeness) also holds, so every logically valid predicate is a theorem. This property does not extend to axiomatic set theory (if it is consistent), so it is not the case that the notions of theoremhood and logical validity can be used interchangeably in Z.

As a result, we must conclude that the unqualified use of statements such as the one which began this subsection is unfortunate, if not misleading.

4 Approaches to Logic and Semantics

A logical presentation is able to give meaning to a metalogical statement like $\vdash$ *pred*. A denotational semantics—giving models for the language—is able to give meaning to statements like $\models pred$. The most tractable way to demonstrate inductively that a logical calculus is consistent is to exhibit a semantic structure and demonstrate that the logic is sound with respect to the semantics. This has been the intention of the Z Standards activity [20], though lately most effort has been spent on the model.

Many texts have given elements of a logical calculus for Z, [25] and [30] being the among the first. Later, Spivey [26] remarks that his Z Reference Manual has not included a full set of inference rules, not least because there is insufficient experience of how to present such rules so that they interact helpfully. Potter et al. [22] devote a chapter to formal reasoning in Z, but for pedagogical reasons, rather than to present a systematic reasoning system for Z. Perhaps Woodcock and Davies [29] give the fullest textbook treatment of reasoning in Z, but again, this is presented as a good means of understanding the complexities and nuances of the notation.

Separate accounts have described logics for Z in a more systematic manner. $\mathcal{W}$ [28] was perhaps the first; it has been followed by $\mathcal{V}$ [7], which corrects a number of shortcomings in the account of $\mathcal{W}$. These logics have been developed to be part of the Z Standardization activity. Henson and Reeves [11] have also produced a logic for Z, which they have named Z_C. Their logic and conclusions mirror the $\mathcal{W}$ family in some respects, but achieve a higher level of rigour and a greater separation of concerns. They argue that some of the innovations of $\mathcal{W}$, such as a novel treatment of substitution, are unnecessary—see below.

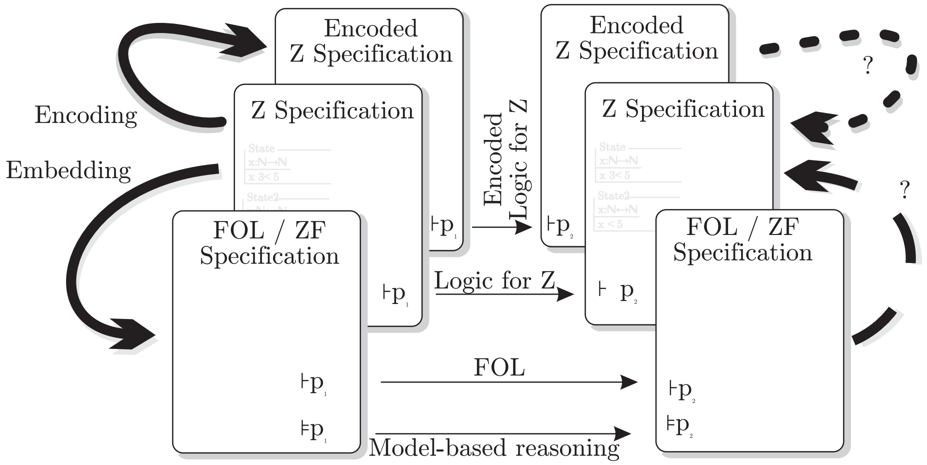

Fig. 1. Some approaches to reasoning

Figure 1 illustrates some of the approaches to giving meaning to, and/or reasoning about, Z specifications. Logics for Z transform one conjecture into another—or discharge the proof obligation to show that the conjecture is a theorem—without leaving the Z notation. In each case, the turnstile symbol $\vdash$ is understood relative to the logic in use, be it $\mathcal{W}$, $\mathcal{V}$, Z_C, etc.

The figure also illustrates the possibility of transforming a Z specification into a model, in, say the language of first-order logic and set theory. It is then possible to re-cast the conjecture in two different ways—as a possible theorem of the formal system of first-order logic and set theory, or as a property of the model. These two activities appear very similar indeed, but are in fact quite different endeavours. Working in the former system will be a formal logical activity; work in the latter is more akin to mainstream mathematics. One is in the domain of proof theory, the other, model theory.

Most proof tools take the first of these approaches. Z/EVES [23] does precisely this; ProofPower [13] and Z-in-Isabelle/HOL [14] use higher-order, rather than first-order, logic. Bowen and Gorden [5] explain that these embeddings of one logic in another may take place at a number of different *depths*, depending upon the extent to which the artefacts of the high-level language (Z) are modelled directly in the host logic. In Z, the principal interest is with the modelling of schemas—see below. A comparison of tools on this basis is included in [18].

It is worthwhile noting that the soundness of these formal logical manipulations is itself guaranteed by an embedding in some model. The soundness (where proven) of each logical approach means that the sets of transformations *increase* when passing down the page—that is, the available logical steps in a logic for Z is a subset of those available by reasoning about a representation of Z in FOL/ZF, etc. As we have observed, properties true in the model may not be provable in the logic, so the inclusion is typically as a *proper* subset.

Having performed logical reasoning in FOL or HOL, it is not always an easy task to transform the resulting terms back into Z. The deeper embeddings accomplish this quite successfully. Z/EVES, in particular, is able to recast virtually every predicate back into a Z form. Z/EVES is one system which might, then, be said to induce a logic for Z, though the complexity of the transformation means that the details of this logic have not been elucidated.

An alternative to embedding the Z notation in another logic is to encode a logical calculus for Z in a logical framework [17, 15], or in a custom implementation such as CADi$\mathbb{Z}$ [27]. The soundness of these encodings is usually called *faithfulness.* An encoding is called *adequate* if it allows precisely the same set of manipulations as the logic it encodes. In practice this is rarely the case, so encodings are shown in the diagram with a transformation arrow the highest on the page; it may be imagined to admit the smallest set of transformations.

The diagram is of course only indicative. Particular logics, encodings, embeddings, and models will not necessarily form a neat hierarchy, because there is no general agreement on the class of Z theorems and derivations.

5 Schemas

Schemas are without doubt the most important and distinctive feature of the Z notation. Semantically, they represent a challenge, and framing suitable inference rules for dealing with them has been a significant research activity, which is reported elsewhere [9, 6]. The difficulty with schemas has been in finding a uniform representation which can represent a schema abstractly, and used to give its properties in each situation in which it may appear.

Consider the following schema, which describes a size-limited phone directory, assuming that MAX has been given some natural number value.

$$S == [n : NAME \ffun PHONE \mid \# n < MAX]$$

S might be used as a declaration, as in the following predicate which says that only international numbers are stored:

$$\forall S \bullet (\forall p : \operatorname{dom} n \bullet p \in INTERNATIONAL)$$

S might be used as an expression, as in the following predicate which says the same thing:

$$\forall s : S \bullet (\forall p : \operatorname{dom} s.n \bullet p \in INTERNATIONAL)$$

S might be used as a predicate, as in

$$\forall n : NAME \ffun PHONE \bullet S$$

which we might interpret as saying that *all* phone directories are limited in size.

The difficulty arises because when S is used as a predicate, the component labelled n behaves like a free variable—it can be captured by a quantifier. Incidentally, MAX cannot be captured by a quantifier; in all instances of S it takes the value it did at the point where S is defined. When S is used as an expression, n is more like a bound variable—though not entirely, as alpha-conversion would affect the type of the expression. Moreover, in Draft Standard Z, a schema-type object can be created at any level of scope, not just at the top level, so the boundness/freeness of variables depends critically on the *context* of the term.

Dealing with these issues led the designers of $\mathcal{W}/\mathcal{V}$ etc. to create an elaborate calculus of bound and free variables, and eventually an object-level notion of substitution. When a schema is used as an expression, it denotes a set of *bindings*, which are associations of labels with values. For example, the schema

$$\begin{array}{|l}
\hline T \\
x, y : \mathbb{N} \\
\hline
x < y \\
\hline
\end{array}$$

has as members bindings such as

$$\llparenthesis\ x == 3, y == 4\ \rrparenthesis, \llparenthesis\ x == 2, y == 6\ \rrparenthesis$$

etc. A binding such as $\llparenthesis\ x == e\ \rrparenthesis$ carries the same semantic content as a traditional substitution $[x/e]$. Instead of writing $P[x/e]$, we might write

$$P \odot \llparenthesis\ x == e\ \rrparenthesis$$

the latter being a wholly Z predicate.

Henson and Reeves [11] have since suggested that this innovation was unnecessary and over-complicates the logic. Their logic avoids needing schema components to be sometimes bound and sometimes free, and consequentially is able to use a more usual notion of substitution.

The $\mathcal{V}$ logic presents a complete calculus for schemas based on this approach to bindings and substitution. Henson and Reeves [9] present a similar calculus, based upon their logic Z_C, retaining a more traditional notion of substitution, and basing their rules on a notion of schema membership which is more liberal. Both pieces of work demonstrate sufficient inference rules to allow schema objects to be eliminated entirely—demonstrating that a logic for the whole of Z can be constructed using ordinary ZF and adding one new construction (the labelled product/schema type/binding).

These calculi also make formal many of the results that have been used by Z practitioners (and schema expansion tools) over many years for the analysis of operators in the schema calculus. For example, if D_1 and D_2 represent schema declaration parts in 'normalized' form (itself typically described informally) then

$$[D_1 \mid P_1] \wedge [D_2 \mid P_2] = [D_1 \sqcup D_2 \mid P_1 \wedge P_2]$$

where $\sqcup$ is some signature-compatible union. A proof of this property is presented by Henson and Reeves [9], though it was also possible using $\mathcal{W}$ [28].

6 Undefinedness

Whilst most well-formed expressions in Z can be given a meaning quite readily, there are two places where a problem arises with potentially undefined terms. One is in the area of function application; the other with improper μ-terms. The latter may be seen as a special case of the former (and vice versa), and so we restrict our attention to function application. (The μ operator is used to select from a set the unique member having a given property. Where no such unique member exists—either through there being no member with the property, or several members with the property—the μ-term is called improper. It is a simple matter to reconstruct this as a discussion of functions and relations.)

Z allows functions to be total or partial. In Z, both functions and relations are sets of pairs. There is no uniform procedure to check whether a given relation is in fact a function, so the language allows relations to be applied to arguments as if they were functions. Problems may arise if a partial function is applied to an argument not in its domain, or if a relation is applied at a point where it is not functional (i.e. it maps its argument to more than one value): what is the function application to denote, and how does this value affect the surrounding terms, or the predicate in which it occurs?

This is not merely an academic enquiry, since it may materially alter the meaning of a specification. Moreover the use of partial functions is very common Z style, so the question potentially affects very many Z specifications. Z has been criticised on these grounds, particularly by the creators of PVS [21]. In PVS all functions are total—the burden is thereby shifted into the type system, which unlike Z's is therefore undecidable.

The tool Z/EVES [23] incorporates a similar check that each function application is well-formed. Its authors report that the majority of 'real' Z specifications examined have failed this test. Many have argued (notably, the late Peter Lupton [16]) that a specification which exploits any interpretation of undefinedness is a bad specification.

6.1 Approaches to Undefinedness

A comprehensive treatment of the possible approaches to undefined terms is given by Arthan [1]. These include the possibility of allowing undefined terms to be reflected in making the predicates in which they arise also undefined. That approach gives rise to a three-valued logic, as used in VDM [12], but it is not generally embraced by Z users (the Cogito project [3] is an exception).

The two most popular approaches are characterised by Arthan as 'UPF' (undefined propositions are false) and 'UED' (undefined expressions denote). In the latter case, all expressions are assumed to denote a value, but it may not be possible to determine which one.

The Draft Z Standard has accommodated the variety of approaches by loosely defining the semantics of function application. Whereas most expressions are defined by an *equation*, so that the meaning of a particular expression is a particular set or relation in the underlying set theory, function application is defined by a *set inclusion*. Thus, the value of an expression involving a relation applied outside is domain, or where it is not functional, is not prescribed.

Different logical systems may resolve this in different ways, and nevertheless be considered compliant (arguably, sound). For example, each type could be given a non-Z error value, and this could be taken as the value of the undefined term. Or, possibly, there could be a different error value for each term, to ensure that undefined terms are not inadvertently made equal. The resolution chosen by the $\mathcal{W}/\mathcal{V}$ family of logics is to determine that undefined expressions denote a Z value of the appropriate type, but not to allow sufficient apparatus to determine which value that is.

6.2 Baumann's Question

The $\mathcal{W}/\mathcal{V}$ resolution described above gives rise to a question posed by Baumann [2].

If every expression denotes a Z value of the appropriate type, then surely

$$\vdash \forall f : X \rightarrow\!\!\!\!\!\!\!+\; Y \bullet \big(\forall x : X \bullet (\exists y : Y \bullet f\ x = y)\big)$$

therefore

$$\vdash \forall f : X \rightarrow\!\!\!\!\!\!\!+\; Y \bullet (\forall x : X \bullet (\exists y : Y \bullet x \mapsto y \in f))$$

therefore

$$\vdash \forall f : X \pfun Y \bullet (\forall x : X \bullet x \in \operatorname{dom} f)$$

therefore

$$\vdash \forall f : X \pfun Y \bullet f \in X \rightarrow Y$$

That is to say, all partial functions in Z are also total functions. This is clearly undesirable; indeed, it is at odds with most users' understanding of Z.

The resolution of this apparent paradox lies in the difference between *truth* (in the model) and *provability* (in the logic). Writing $\models_D$ *pred* to mean that *pred* holds in the model extended so that it has the property 'every term denotes', the following may be true

$$\models_D \forall f : X \pfun Y \bullet \big(\forall x : X \bullet (\exists y : Y \bullet f\ x = y)\big) \quad ,$$

but it does not follow (indeed, it is not the case) that this predicate is a theorem of $\mathcal{W}$ etc. (Its truth is not certain, because the extension of the model may be made in a number of different ways: the model might determine that any application of a function outside its domain results in a 'bottom' value which is outside the type system, or a distinguished value of the appropriate type, or an entirely unspecified member of the type. The predicate above would be true in the latter two cases—provided Y is a basic type, and not some subset—but not in the first.)

6.3 Issues of Methodology

We have already argued that matters of logic are important not only to those who wish to prove properties of their specifications, but to all who try to *understand* Z specifications. In particular, a working knowledge of logic will enable the reader to ask questions about what would have happened if something had been written differently.

The treatment of undefinedness has an impact on how one writes specifications as well as how one reasons about them. An oft-cited example is the specification which declares a (possibly infinite) set and then asserts that its cardinality is fixed.

$$\begin{array}{|l} s : \mathbb{P}\,\mathbb{N} \\ \hline \#s = 4 \end{array}$$

The cardinality function # is partial; its value is defined only on finite sets. The question of whether or not the definition of s is a useful one depends upon one's treatment of undefined terms. If every expression denotes ('UED'), then $\#s$ and 4 can be used interchangeably in the sequel, which is probably almost what the specifier intended—but not quite. On the other hand one interpretation of the position 'UPF' would mean that the very possibility that $\#s$ may be undefined would be enough to make the predicate false, and therefore the whole specification inconsistent (and useless). Another interpretation of 'UPF' would be to rule out those models in which $\#s = 4$ is false (for whatever reason, including undefinedness), leaving a specification which performs as 'intended'. The apparatus for achieving the latter is non-trivial.

Conversely, we can easily exhibit specifications—such as set comprehensions or schemas—where we would like undefined terms to give rise to false predicates, so that the rogue terms are excluded from the set/schema. For example, writing

$$S_4 == \{\ s : \mathbb{P}\,\mathbb{N} \mid \#s = 4\ \}$$

we would hope that S_4 would be the set of all sets of natural numbers having cardinality 4—and no others. On the 'UED' interpretation, we can be sure that the sets of four natural numbers are members of S_4, but we cannot prove that, say, $\mathbb{N}$ is not. In 'UPF', however, S_4 would contain exactly the sets of natural numbers having four elements, and nothing else.

The study of undefinedness has a long history in classical mathematics, with no universal answer, and often no need for one because the 'intuitive' answer is usually the right one (normal mathematical practice not being overly formal). That there should be no ready answer in Z is therefore not surprising.

7 Future and Related Work

7.1 Managing Complexity

Just as undefinedness is a widespread problem, so too is the issue of managing complexity. Z's schema notation provides the specifier with the opportunity to write very deeply-nested specifications in a very compact way. Anyone who has used a schema expansion tool will know how readily an innocuous-looking five-line schema can when fully expanded occupy several printed pages.

A challenge in supporting proof is to find ways to manage this complexity. This issue is almost orthogonal to the logical concerns already raised, but has a profound impact on the tractability of formal proof for anything but the smallest specification.

Moreover, the language is quite rich, and uses a variety of specialised symbols. Again, these are a means to the management of complexity, since they allow complex ideas to be expressed succinctly. They are, however, outside the general abilities of most proof tools today, though they need not be, since on-screen display of symbol fonts is now quite commonplace.

Support for practical reasoning will require solving these two difficulties in tool implementations. The first is ideally a methodological issue as well as an interface one. Despite discovering logics for schemas, no practical method for using schemas to structure proofs as well as specifications has been discovered.

7.2 Other Logics

Whilst this paper has largely described Z in relation to first-order logic, higher-order logic has been found to be a good basis for Z semantics. In particular, it offers a very appropriate type model (provided schema types/bindings are incorporated). [24] has described an isomorphism between Z and HOL (disregarding names in schema types). This provides the theory underlying the Z in Isabelle/HOL work described above.

Others have considered using constructive logics [10]. In this approach, proof becomes a means of program development. In other contexts, second-order logic has been found to be a useful tool; perhaps it could also add expressiveness to Z.

8 Conclusions

Z now has quite a lengthy history, and sometimes the original motivations and assumed theory have become obscured. This paper's purpose is to make some observations about the purpose and role of proof in Z, and its relationship to first-order logic. General understanding of these topics has been divergent, and the hope is to promote some common acceptance and understanding; in essence to provide a framework for a philosophy of proof in Z. Much of the paper has been devoted to documenting issues that are 'part of the folklore' of Z proof but do not seem to have been recorded anywhere—to the annoyance of newcomers to the field.

The original connection between Z and first-order logic was very close. Various developments have served to obscure this relationship. Two pieces of recent work [11, 6] show that the relationship persists, and indeed a first-order logic can be described which, with the single addition of schema *types* can be used to reason about the whole language. Insofar as demonstrating this relationship is non-trivial it also suggests that Z adds some significant structure to the language of first-order logic and set theory. Experience suggests that this structure is useful for specification. It appears that it may also be useful for proof.

References

[1] R. D. Arthan. Undefinedness in Z: Issues for specification and proof, 1996. Presented at CADE-13 Workshop on Mechanization of Partial Functions.

[2] Peter Baumann. Private Communication, April 1994.

[3] Anthony Bloesch, Ed Kazmierczak, Peter Kearney, and Owen Traynor. The Cogito methodology and system. In *Asia–Pacific Software Engineering Conference '94*, pages 345–355, 1994.

[4] J. P. Bowen, A. Fett, and M. G. Hinchey, editors. *ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, 24–26 September 1998*, volume 1493 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[5] J. P. Bowen and M. J. C. Gordon. A shallow embedding of Z in HOL. *Information and Software Technology*, 37(5–6):269–276, 1995.

[6] Stephen Brien and Andrew Martin. A calculus for schemas in Z. *J. Symbolic Computation*, 2000. To appear.

[7] Stephen M. Brien. *A Logic and Model for the Z Standard*. D.Phil. thesis, University of Oxford, 1998.

[8] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[9] M. C. Henson and S. Reeves. A logic for the schema calculus. In Bowen et al. [4], pages 172–191.

[10] Martin Henson and Steve Reeves. New foundations for Z. In Jim Grundy, Martin Schwenke, and Trevor Vickers, editors, *IRW/FMP'98*. Springer-Verlag, 1998.

[11] Martin C. Henson and Steve Reeves. Investigating Z. Technical Report CSM-317, Department of Computer Science, University of Essex, 1998. *Journal of Logic and Computation*, to appear.

[12] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall Intenational, second edition, 1990.

[13] R. B. Jones. ICL ProofPower. *BCS FACS FACTS*, Series III, 1(1):10–13, Winter 1992.

[14] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In *1996 International Conference on Theorem Proving in Higher Order Logic*. Springer-Verlag, 1996.

[15] Ina Kraan and Peter Baumann. Implementing Z in Isabelle. In Jonathan P. Bowen and Michael G. Hinchey, editors, *ZUM'95: The Z Formal Specification Notation*, volume 967 of *LNCS*, pages 355–373. Springer-Verlag, 1995.

[16] Peter J. L. Lupton. Z and undefinedness. Technical Report PRG/91/68, Z Standards Panel / Programming Research Group, 1991.

[17] A. Martin. Encoding W: A logic for Z in 2OBJ. In J. C. P. Woodcock and P. G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods*, volume 670 of *Lecture Notes in Computer Science*, pages 462–481. Formal Methods Europe, Springer-Verlag, 1993.

[18] Andrew Martin. Why effective proof tool support for Z is hard. Technical report 97-34, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072. Australia, November 1997.

[19] Elliott Mendelson. *Introduction to Mathematical Logic*. Mathematics Series. Wadsworth and Brooks/Cole, 1987.

[20] John Nicholls, editor. *Z Notation*. Z Standards Panel, ISO Panel JTC1/SC22/WG19 (Rapporteur Group for Z), 1995. Version 1.2, ISO Committee Draft; CD 13568.

[21] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[22] B. F. Potter, J. E. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall International Series in Computer Science, 2nd edition, 1996.

[23] M. Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, 1997.

[24] T. Santen. On the semantic relation of Z and HOL. In Bowen et al. [4], pages 96–115.

[25] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, January 1988.

[26] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, second edition, 1992.

[27] I. Toyn. Formal reasoning in the Z notation using CADiZ. In *Proc. 2nd Workshop on User Interfaces to Theorem Provers, York*, July 1996.

[28] J. C. P. Woodcock and S. M. Brien. $\mathcal{W}$: A Logic for Z. In *Proceedings 6th Z User Meeting.* Springer-Verlag, 1992.

[29] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement.* Prentice Hall International Series in Computer Science, 1996.

[30] J. C. P. Woodcock and M. Loomes. *Software Engineering Mathematics: Formal Methods Demystified.* Pitman, 1988.

Acknowledgements

If I have any insight in this area, it has been gained in discussions with members of the Z Standards panel. Anthony Hall helped to make stark the choices faced in dealing with undefinedness, and discussions with Martin Henson have greatly helped my understanding and exposition.