Abstract: This document consists of notes on aspects of the specification language Z prepared as a contribution by the FST project to the Z standardisation activity currently being undertaken by the ZIP project.

Distribution: R.D. Arthan
R.B. Jones
J.M. Nicholls PRG
C.T. Sennett RSRE
J. Woodcock RAL
0  Document control

0.1  Contents list

0 Document control .......................................................... 2
  0.1  Contents list .......................................................... 2
  0.2  Document cross references ........................................ 2
  0.3  Changes history ....................................................... 3
  0.4  Changes forecast ..................................................... 3

1  GENERAL

1.1  Scope ................................................................. 3
1.2  Introduction .......................................................... 3

2  PARTIALITY AND EQUALITY

2.1  Partially-defined Expressions ...................................... 4
2.2  equality ................................................................. 6
  2.2.1  formal semantics .................................................. 6
  2.2.2  reference manual ................................................ 6
  2.2.3  recommendation ................................................... 7
2.3  set abstraction and lambda abstraction ............................ 7
2.4  definite description ................................................... 8

3  IMPLICIT GENERIC PARAMETERS .................................. 8

4  LOOSE GENERICS ......................................................... 9

5  SCHEMA OPERATIONS .................................................. 9

5.1  $\Delta$ and $\Xi$ ......................................................... 9
5.2  Theta-terms .......................................................... 9

6  CLOSURE PROPERTIES .................................................. 9

7  CONSERVATIVE EXTENSION ......................................... 10

8  MODULES ............................................................... 10

0.2  Document cross references


0.3 Changes history

Issue, Date 15 June 2012

This is the first issue of this document.

0.4 Changes forecast

The document may be extended as further insights arise from the proof support development work.

1 GENERAL

1.1 Scope

This document provides comments on the definition of the specification language Z. These comments are provided from the perspective of a group concerned with the development of a proof tool for use with Z and intending to apply this tool on applications requiring extensive machine checked formal proofs.

1.2 Introduction

ICL Secure Systems, formerly a part of ICL Defence Systems, has for some time been engaged in work on very high assurance secure systems developments which has involved the preparation of machine checked formal proofs. Because of the lack of suitable proof support for the specification language Z, which has been preferred by our customers, these proofs have been obtained by translation of specifications from Z to HOL, and the construction of proofs using the HOL proof assistant developed by Dr. Mike Gordon and others at the University of Cambridge.

ICL Secure Systems is now engaged in the development of proof support tools in its own account, and expects in due course to provide proof support for the full standard Z specification language by building upon a prototype HOL system recently developed at ICL Winnersh.

ICL Secure Systems therefore has an interest in remaining abreast of developments in the standardisation process for Z, and is able to contribute to this process some insights into how various decisions may impact the feasibility and costs of fully formal reasoning with Z.

The manner in which Z proof support is to be implemented involves activities which are equivalent to reformalising the semantics of Z and deriving proof rules from this formalised semantics. This work might form the basis for formal semantics and proof rules to be included in the standard, or (more probably) would furnish a useful cross check to help identify problems in any semantics or proof rules proposed for inclusion in the standard.

Since no draft standard has yet been made available these comments are made on the language as defined in The Z Reference Manual [4]. The semantics in the reference manual is informal, though quite a number of ‘Laws’ are supplied, which may help to check any interpretation of the informal semantics. Reference has also been made to Understanding Z[3], which contains a formal semantics
for a part of the language described in the reference manual. Some discrepancies are noted between these documents.

In preparing this material I have benefited from discussions with members of the ICL FST project (particularly Rob Arthan), the RSRE SCEIP unit, and Jim Woodcock of Rutherford Appleton Laboratories. The opinions expressed and recommendations made are however my own.

2 PARTIALITY AND EQUALITY

On my reading of the documents cited, the position on partiality and equality is not coherent. On the semantics of equality, for example, the reference manual contains ‘Laws’ which are not consistent with the semantics as described in the text, and neither the text nor the ‘Laws’ agree with the formal semantics.

These choices have significant impact on the convenience with which the specification language may be used, and upon the costs associated with formal proofs.

2.1 Partially-defined Expressions

Both references make it clear that there are intended to be circumstances under which a term may fail to have a value (by distinction with it being unknown what its value is). This is significant in representing a clear departure from classical first order set theory (or higher order logic) in which terms are always deemed to have a value, though we may at times be unable to establish what the value is. The simplest example of a term lacking a value is the case of an application of a partial function to a value outside its domain.

In a strictly classical framework function application could be described using a definite description operator. The axiom characterising this operator gives no information about value of the definite description if the description fails uniquely to identify an individual.

\[
\begin{array}{c}
\vdash [A] \\
\mu^l : \mathcal{P} A \to A \\
\forall x:A \bullet \mu^l \{ x \} = x
\end{array}
\]

Note that here \( \mu \) is shown as a total function. (I confess that this, and the following are loose generics.)

Application may then be defined (taking the liberty of using an identifier consisting only of a decoration):
Note that application is also total.

The above treatment would permit classical first order set theory or higher order logic to be used, and would result in there being expressions whose value could not be determined, but which nevertheless were deemed to have a value (of the appropriate type).

This is not a position which I am advocating, it is described simple to draw the contrast between this position and the various alternatives.

My understanding of the references is that in Z, terms may in some stronger sense fail to denote (this is particularly clear in the formal semantics). A useful reference which appears to treat partiality in a manner similar to that implied by the formal semantics is [1].

If definite description were defined as a function (which it isn’t, but could be), the definition would be along the lines:

Note that here µ² is shown as a partial function.

Application may then be defined:


Having taken the position that some terms may fail to denote, the formal semantics then describes precisely the effect of such undefined terms on the value of the expressions or predicates in which they occur.

According to the formal semantics, term forming constructs are generally strict in the sense that they yield undefined terms when any one of their constituents is undefined. More complex rules are required in the case of constructs involving quantification or abstraction over terms.

2.2 equality

2.2.1 formal semantics

According to the formal semantics, equality and membership are strict in the slightly different sense that if either or both of their arguments is undefined then they yield \textit{false}. This is fine for membership, but not so good for equality.

A consequence of the semantics as described in ‘Understanding Z’ is that the law of reflection does not hold for arbitrary terms, but only for terms which are \textit{defined}. This position also has serious consequences for the ease of definition of partial functions in terms of other partial functions. This is tacitly acknowledged by Spivey through his extensive use in [3] of the special so-called \textit{strong} equality •, which he nevertheless did not include in the language he was defining. Equality as defined does not permit the domain of definition of a partial function to be inherited through equations from the domain of definition of the expressions used to define the functions. This therefore requires the specifier to specify explicity the domain of definition (in order to restrict quantification to this domain), which makes the specification in some cases substantially more complex than a functional implementation would be, and make the specification much more likely to prove (through error in specifying this domain) inconsistent. It also inhibits theorem proving, since rewriting an invocation of a partial function using the defining expression will not be possible unless that expression has been shown in the particular circumstances to be well defined. This is one of several features in the formal semantics which unnecessarily inhibits rewriting of expressions until such time as they have been shown to be defined, which is regrettably since such rewriting may well be the best way of showing that they are defined.

2.2.2 reference manual

In the reference manual this position is modified. Equality and membership are said to be \textit{undetermined} in the case that either argument is undefined. We have not moved to a three valued logic, but we have left the truth value of these predicates undefined under some circumstances.

Among the laws for equality in the reference manual ‘\(x = x\)’ is cited. This is ambiguous insofar as it is not clear whether the ‘\(x\)’ here is an object language variable, or a meta-language variable, and if a meta-language variable whether it is to range over object language variables or object language terms. In the case that it is an object language variable or ranges over object language variables, the compatibility of the law with a putative semantics depends upon whether variables are deemed to range over undefined values or not. The law is consistent with the informal semantics in the reference manual only on the assumption that any logical rule of substitution is restricted to term which have been shown to be \textit{defined}. 
This will not be true under the semantics described in either of the references. The weaker result ‘\([A] \vdash \forall x : A \bullet x = x'\)’, is all that can be obtained. This and related results significantly impact the costs of obtaining formal proofs, since it inhibits rewriting expressions until they have been shown to be defined (something you might like to do to show that they are defined!).

2.2.3 recommendation

I would recommend that both forms of equality relation (= and \(\bullet\)) found in [3] should be included in the language.

2.3 set abstraction and lambda abstraction

The reference manual and the formal semantics disagree on these constructs insofar as the proof rules identified in [4] are not sound in respect of the semantics in [3].

I believe that the semantics implicit in the rules shown in the reference manual are better both from the point of view of ease of writing specifications and from that of proving results about them, and I would recommend acceptance of those rules and a semantics consistent with them.

The discussion can be confined to set abstraction since lambda abstraction is defined in terms of set abstraction (and I agree that it should be, though some have differed, see [2]).

The difference between the two lies in the consequences of the term in the body of the abstraction (after the \(\bullet\)) proving to be undefined somewhere in the range of the abstraction. In the formal semantics this would result in the abstraction being undefined (this was pointed out to me by Rob Arthan). In the reference manual the undefined value is simply ignored. Neither semantics permits an the undefined element to be a member of a set (if it did the law of extensionality would fail).

The difference between the two may be seen by showing the rules which result in the two cases.

The following laws is cited in the reference manual ([4] page 60.

\[
\begin{align*}
\forall S \bullet E = E \vdash y \in \{S \bullet E\} & \iff \exists S \bullet E = y \\
\end{align*}
\]

Provided \(y\) is not a variable in (bound by) \(S\).

To reflect the semantics as expressed in Understanding Z ([3], page 68) this would have to be qualified:

\[
\begin{align*}
\forall S \bullet E = E \vdash y \in \{S \bullet E\} & \iff \exists S \bullet E = y \\
\end{align*}
\]

Where, with the semantics of “=” as given in Understanding Z, \(E = E\) may be read \(E\) is defined.

For lambda abstraction a law of beta-reduction may be derived from the definition of lambda abstractions in terms of set abstractions and the definition of function application.
2.4 definite description

The reference manual and the formal semantics disagree on the semantics of definite description. Of the two accounts I believe the one in the formal semantics to be preferable.

This preferred account treats $\mu$ as if it is a function which can be applied to any set and which yields a value if and only if that set is a singleton. It is arguable that $\mu$ should be treated as a function (and allowed to be applied to arbitrary sets, as it once was) and that the standard notation be regarded as an optional elision (of the curly brackets).

The semantics in the reference manual is not consistent with this view. This is illustrated by the following theorem schema (not cited by Spivey) which I believe to be consistent with the semantics as described in the reference manual (page 61):

\[
\forall S \mid P \bullet E = v
(Provided \ v \ is \ not \ bound \ by \ S.)
\]

According to the formal semantics the following rule would hold:

\[
(0 \in \mu_{x:\mathbb{N}} | true \bullet 0) \Leftrightarrow \{0\} = \{x:\mathbb{N} | true \bullet 0\}
\]

These two rules differ when instantiated to:

\[
(0 \in \mu_{x:\mathbb{N}} | true \bullet 0) \Leftrightarrow (\exists x:\mathbb{N} | true) \wedge \forall x:\mathbb{N} | true \bullet 0 = 0)
\]

where the right hand side is plainly false, and:

\[
(0 \in \mu_{x:\mathbb{N}} | true \bullet 0) \Leftrightarrow \{0\} = \{x:\mathbb{N} | true \bullet 0\}
\]

where the right hand side is true.

3 IMPLICIT GENERIC PARAMETERS

The discussion of how to instantiate omitted generic parameters in [4] (page 80, ignoring the confused statements about infix generics) may be sufficient to enable type checking to be undertaken. However, generic parameters are sets rather than types, and for proof (or simply to understand the meaning of a specification) it is necessary to know exactly which set is to be supplied. Type inference will not help with this.

Some clear rules will have to be articulated about what sets are used where actual generic parameters are omitted.
4 LOOSE GENERICS

I believe that the exclusion of these from the language was mistaken, and should be reversed. They are actually more difficult to exclude than to support. Their exclusion unnecessarily promotes overspecification.

5 SCHEMA OPERATIONS

5.1 $\Delta$ and $\Xi$

$\Delta$ and $\Xi$ as presently defined are the only feature of the language which is non-monotonic (i.e. adding premises causes deletion of conclusions), and therefore represent a problem from the point of view of proof support.

This is because $\Delta STATE$ is deemed to have the ‘conventional’ meaning unless the user chooses to give it some other meaning. This is a freedom which can only cause confusion.

From an implementation point of view it would be better if they were both regarded as operators over schemas (and were not permitted as the first character of identifiers). Failing that the next best thing would be to prohibit their redefinition.

5.2 Theta-terms

The reference manual ([4], page 64) indicates that a theta-term must be used in a context in which all the names in the signature of the schema are in scope, but does not require them to have the same type as that assigned to them in the schema.

This seems to be contrary to the spirit of the language and I would recommend that the types should be required to match.

6 CLOSURE PROPERTIES

Automatic theorem proving is likely to depend heavily upon rewriting with equations. In the case of schema expressions this raises the difficulty that there are no equations of schema expressions in the language.

It is therefore desirable that the language should be extended by admitting equations between schema expressions as predicates.

A further problem then arises. The language is not closed under equality substitutions. For example, if a schema designator in the context of a predicate or declaration were to be replaced by a schema expression (e.g. its definition), the result would not be legal Z, since only schema designators are permitted in this context.

Arbitrary constraints of this kind are likely to prove detrimental to productive proof work by preventing the use of powerful general rewriting facilities even though these are semantically sound. These
constraints could be relaxed exclusively in the context of proof work and retained in specifications, but it would be better if they were reviewed with a view to removing unnecessary constraints from the specification language.

7 CONSERVATIVE EXTENSION

I would recommend that some way of distinguishing conservative from non-conservative extensions be provided, and a way of indicating that a coupling of new generic sets with axiomatic descriptions is conservative (so that generic sets can be constrained in a demonstrably conservative way).

The case for this needs to be stated at greater length than I am able to do at present.

8 MODULES

A standard for the Z language must surely contain some support for modular specifications.

From our present proof support technology base that could be a recipe for nightmares, and I would favour the simplest acceptable modularity features, essentially confined to giving some control over the scope of global names.