ISO Standard Z

/What is CADiZ?

This section of the CADiℤ manual compares the notation of ISO Standard Z with that of Spivey's Z Reference Manual, 2nd edition (referred to below as ZRM). It is derived from a paper presented at ZUM98.

1. Contents of this page

2. Background

This paper compares the notation of the final Committee Draft (CD) of the ISO standard *Z Notation* with that of Spivey's *The Z Notation: A Reference Manual*. The standard is referred to below as *the final CD* and as *ISO Standard Z*; Spivey's book is referred to below as *ZRM*.

ZRM was a huge step forward at the time of its first publication. Many syntax and typechecking tools for Z have been based on it, and it has become a *de facto* standard. It is an excellent work that has certainly served the Z community well, but it is not perfect, there being several issues that it did not adequately address. Such inadequacies are inevitably resolved by different users in different ways, until a standard is again agreed. An aim of the final CD is to provide adequate resolutions of the problems, so that diverse dialects can be avoided.

3. Improvements

This section presents improvements in ISO Standard Z that address inadequacies in ZRM.

3.1. Sections

Specifications are rarely written in terms of the Z core language. Even pedagogic examples usually refer to the definitions of the mathematical toolkit. Real specifications are constructed from libraries or toolkits of operations relevant to particular application domains. It should be possible to reuse toolkits by reference, without having to duplicate them into every specification that uses them. This is an issue that ZRM ignores.

ISO Standard Z provides probably the simplest possible solution to the toolkit reuse problem, in the form of its *section* notation. A Z section contains a sequence of paragraphs, just like a ZRM specification. It also has a header which gives this section its name and lists the names of those other sections that are parents of this one.

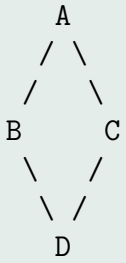
```
section myspec parents toolkit
```

Such a section, in combination with its ancestor sections, is what comprises a specification in ISO Standard Z. At any point in a specification, the environment of global declarations comprises those already declared in the current section and all declarations of all sections named as parents of the current one. For example, consider a specification comprising four sections from which just the headers are as follows.

```
section A
section B parents A
```

```
section C parents A
section D parents B, C
```

The parents relation can be depicted as a directed acyclic graph. (Arrowheads needed!)



Within sections `B` and `C`, the declarations of section `A` may be used. Within section `D`, the declarations of sections `A`, `B` and `C` may be used.

Global redeclaration is not permitted, either within a section or across related sections. Unlike other proposals for module-like facilities, a section cannot hide any of its paragraphs—it is not an encapsulation mechanism. For backwards compatibility, a single sequence of paragraphs with no section header is accepted as a single section with `toolkit` as its sole parent.

The final CD gives the semantics of sections. The mechanics of exactly how parents are brought together is left to the builders of tools, an obvious possibility being to use a mapping between section names and file names.

3.2. Mutually recursive free types

ZRM presents a free type as an abbreviation for a given type followed by axiomatic constraints to ensure that its constants are elements of the given type, that its constructors are injections producing members of the given type, that the elements and the values returned by the injections are all distinct from each other, and that all values of the type are either elements or returned by an injection. Unfortunately, it does not consider mutually-recursive free types.

ISO Standard Z permits mutually-recursive free types to be written, separated by & symbols within a single paragraph. Mutually-recursive free types are especially useful in describing the syntax of languages, as illustrated by the following fragment.

$$
\begin{array}{l}
dec ::= Dec\langle\langle name \times exp \rangle\rangle \\
\& \\
exp ::= Let\langle\langle seqdec \times exp \rangle\rangle \\
\mid Num\langle\langle \mathbb{N} \rangle\rangle
\end{array}
$$

In this example, a declaration *dec* involves an expression *exp*, and an *exp* can involve local declarations. A larger example would be mutual recursion between predicates and expressions in Z.

The final CD presents a transformation of mutually-recursive free types to given types and axiomatic constraints. Its definition of the membership, totality, injectivity and disjointness axioms follow the same pattern as before, the only change being in the induction axiom which involves all of the mutually-recursive

types simultaneously.

3.3. Operator templates

An operator is a name with special lexical status, for example an infix operator appears between its operands. ZRM notation allows the use of various operators. A use of an operator has to be preceded by its definition. Its definition ought to be preceded by an introduction of a template for the operator, to indicate that the name will be defined and used in, for example, infix position. Without the information provided by the template, it is not possible to parse the operator's definition and uses. Although ZRM says what form of templates it permits operators to have, no notation is specified for the introduction of operator templates. It presumes that all the operators defined in its toolkit are already known to the reader and hence recognisable, and leaves each tool to implement its own distinct notation for templates.

ISO Standard Z has a notation called an operator template paragraph that serves to introduce new operators. Here are some examples from the toolkit.

relation $(\_ \neq \_)$
function 30 leftassoc $(\_ \cup \_)$
generic 5 rightassoc $(\_ < \_)$
function 90 $(\_^{\sim})$
relation $(\mathit{disjoint}\ \_)$

Each of these lines begins with an operator's category and ends with a paren-

thesized pattern. In between, as appropriate for the operator, is precedence and associativity information. An operator's category determines how applications of the operator are parsed: an application of a relation operator is parsed as a relational predicate; an application of a function operator is parsed as an application expression; an application of a generic operator is parsed as a generic instantiation expression. When applications of operators are nested, so that one operator application appears as an operand in another operator application, the intended nesting can be made explicit using parentheses. Alternatively, if no parentheses are used, the precedence and associativity information determines how the applications are nested.

ISO Standard Z notation permits a wider variety of operators to be introduced than ZRM notation. The following table summarises what is permitted.

	ZRM	*ISO Standard Z*
Category	*relation*, *function*, *generic*	*relation*, *function*, *generi*
Precedence	*infix functions* 1..6, *others fixed by syntax*	*functions and generics* 0.
Associativity	*left or right*, *fixed by syntax*	*left or right*, *user-defined*
Arity	1..2	1..*n*, *operands and symbo*

The generalization of the available precedences and associativities has led to some backwards incompatibilities, as listed below. The generalization to arbitrary arity is subject to the restriction that operands and symbols must alternate, meaning that two operands cannot be consecutive without an intervening symbol, and two symbols cannot be consecutive without an intervening operand. An example of the latter restriction would be the consecutive symbols `else if` (within the obvious operator), whereas writing them as a single symbol `elsif` would be per-

mitted. ZRM is more restrictive than suggested by the above table: relations and generics cannot be postfix, functions cannot be declared to be prefix (they just are by default), and there are no nofix operators. ISO Standard Z permits more than suggested by the table: it permits sequence arguments distinct from normal value arguments, as in the following examples of sequence extension brackets and bag extension brackets.

$$\text{function } (\langle \, , \, \rangle)$$
$$\text{function } (\llbracket \, , \, \rrbracket)$$

ISO Standard Z defines relational image brackets and sequence brackets in the toolkit, whereas ZRM had to make special cases for them in the core syntax (page 145).

Chaining of relations in ISO Standard Z is exactly as permitted by ZRM, i.e. only infix binary relations may be chained; a chain may not commence with a prefix relation, nor end with a postfix relation, nor can tertiary or higher relations appear in a chain.

Having introduced the template of an operator, the notation for defining and using an operator is essentially the same as in ZRM. The definition has as its left-hand side the pattern without the parentheses. Applications of an operator are written with expressions in place of _ operand markers, and comma-separated lists of zero-or-more expressions in place of , , operand markers. References to an operator without applying it to any arguments are written as the pattern enclosed in parentheses.

3.4. Conjectures

ZRM presents many laws, but without formalising their syntax as part of Z. Their presentation is pseudo-formal, none of their variables being declared. Proof tools typically provide a common syntax of sequents, suitable for expressing not only laws, but also conjectures, theorems, goals, lemmas and axioms. Of these, conjectures are the starting point for proofs, and are sometimes hand-written within specifications. Unfortunately, different proof tools use different syntaxes for sequents, so it is not appropriate to standardize their syntax. However, standardizing a simpler syntax specifically for conjectures is possible and worthwhile, as this allows them to be written within specifications in a form that potentially eases their interchange between tools and allows them to be subjected to typechecking.

ISO Standard Z's notation for a (generic) conjecture involves a $\vdash?$ symbol followed by a single predicate. The following examples formalize a couple of laws from ZRM.

$$\vdash?\ \forall\, a : \mathbb{Z} \bullet a..a = \{a\}$$

$$[X] \vdash?\ \forall\, x, y : X \bullet x \neq y \Rightarrow y \neq x$$

This simple syntax is chosen as it is likely to conform to the syntax of a sequent, or at least be translatable to a sequent, whatever proof tool is used. A conjecture is valid if its predicate can be shown to be implied by the properties of the specification, without itself contributing to those properties. The following conjecture is an example of an invalid one, but the specification of which it is a

part remains well-formed nevertheless.

$$\vdash ?\ 42 \in \{1, 2, 3\}$$

3.5. Binding extensions and tuple selections

The Z core language provides both labelled and unlabelled product types, called schema types and Cartesian product types respectively. One would expect to find notations for construction and selection operations on values of each of these types, but ZRM offers only a selection operation for values of schema type, and only a construction operation for values of Cartesian product type. ISO Standard Z also offers notations for the other two operations. The following table summarises the notations.

		ZRM		*ISO Standard Z*
	Constructors	*Selectors*	*Constructors*	*Selectors*
Tuples	(x, y, z)		(x, y, z)	$triple.3$
Bindings		$binding.name$	$\langle\!\langle\ x == 1, y == 42\ \rangle\!\rangle$	$binding.name$

ZRM introduces a notation for explaining bindings (ZRM pages 26 and 62), but does not permit use of this as Z notation. ZRM notation is used largely for producing abstract specifications of systems, where the emphasis is on the use of schemas and constraints on them rather than particular bindings, those being more specific and concrete. However, Z can be used in other ways and in other contexts, for example, the author has used it in reasoning about a relational database, where the rows of a table were modelled by the bindings of a schema. ISO Standard Z's notation for the construction of bindings from explicit component

values is called a binding extension expression. The syntax of a binding extension expression conforms to the template

$$\langle\!|\ i_1 == e_1, ..., i_n == e_n\ |\!\rangle$$

where the subscripts distinguish different names i and their associated expressions e, and $n \geq 0$. Bindings can arise in ZRM either by theta expressions or as members of schemas. It is particularly useful to have binding extension notation during proofs, where showing the truth of predicates such as $\theta S = \theta S'$ involves consideration of the underlying binding values. An example of such a proof appears in the next subsection.

ISO Standard Z's notation for the selection of components from tuples is called a tuple selection expression. The syntax of a tuple selection expression conforms to the template

$$e.b$$

where the expression e denotes a tuple and b is a base ten number literal in the range of the arity of the tuple; the components are numbered from 1. These conditions are verifiable by typechecking. ZRM notation provides *first* and *second* selectors for pairs in the toolkit, but no selectors for larger tuples. ISO Standard Z retains *first* and *second* in the toolkit for backwards compatibility.

3.6. Schemas as expressions

An expression has a value of a particular type. A schema has a value—it is a set of bindings—but in ZRM the syntax does not permit a schema to be written as an expression. Instead, ZRM has a separate category of schema expressions, and any expression involving a schema must refer to a separately defined named schema. This distinction between schema expressions and other expressions also prohibits an expression whose type is that of a set of bindings from being used within a schema expression.

This syntactic restriction is an obstacle to formal reasoning. The replacement of a name by its defining expression is a typical substitution-of-equals-for-equals logical inference, but that is precluded by the syntactic restriction. Without that particular inference rule, it is not clear how to replace a reference to a schema by the mathematics of its definition, and hence to reason further. More generally, all formulae arising from logical inferences should be expressible in the concrete syntax, and so irregularities in the concrete syntax should be eradicated.

ISO Standard Z has merged the syntactic category of schema expressions into that of expressions. So an arbitrary schema expression may appear as an inclusion declaration, a predicate, an operand to θ, or as an expression, i.e. wherever a schema reference could appear in ZRM notation. The type system ensures that a schema is used only where an expression whose type is that of a set of bindings is permissible, and that only an expression whose type is that of a set of bindings is used where a schema is required. This change to the syntax is the major cause of the backwards incompatibilities listed below. The following example illustrates use of schema expressions as inclusion declarations and as operands to θ.

$$\begin{array}{|l} S \\ \hline x : \mathbb{Z} \\ y : \mathbb{N} \\ \hline \end{array}$$

$$\begin{array}{|l} \Delta Sx \\ \hline S;\ (S)' \\ \hline \theta(S \setminus (x)) = \theta(S \setminus (x))' \\ \hline \end{array}$$

The ΔSx schema can be interpreted as defining a change to the state represented by schema S in which only the x component's value can change. The conjecture that the value of component y is left unchanged by ΔSx can now be stated and proved, in which can be seen schema expressions being used as predicates.

$$\vdash ?\ \forall S;\ (S)' \bullet \Delta Sx \Rightarrow y = y'$$

Expansion of ΔSx

$$\vdash ?\ \forall S;\ (S)' \bullet [S;\ (S)' \mid \theta(S \setminus (x)) = \theta(S \setminus (x))'] \Rightarrow y = y'$$

Expansion of thetas

$$\vdash ?\ \forall S;\ (S)' \bullet [S;\ (S)' \mid \langle\!|\ y == y\ |\!\rangle = \langle\!|\ y == y'\ |\!\rangle] \Rightarrow y = y'$$

Absorption of binding extensions

$$\vdash ?\ \forall S;\ (S)' \bullet [S;\ (S)' \mid y = y'] \Rightarrow y = y'$$

Absorption of schema predicate

$$\vdash ?\ \forall S;\ (S)' \bullet y = y' \Rightarrow y = y'$$

Absorption of implication

$$\vdash ?\ \forall S;\ (S)' \bullet true$$

Absorption of universal quantification

$$\vdash ?\ true$$

3.7. Empty schemas

An empty schema is a schema with no declarations. One can arise in ZRM notation *via* the hiding of all declarations from a schema.

$$Schema \mathrel{\widehat=} [x, y : \mathbb{Z} \mid x \neq y] \setminus (x, y)$$

This should simplify to the following equivalent paragraph, but ZRM does not permit this to be written.

$$Schema \mathrel{\widehat=} [|\ \exists x, y : \mathbb{Z} \bullet x \neq y]$$

ISO Standard Z does permit the list of declarations in a schema text to be empty. Empty schemas are surprisingly useful. If the type of S is that of an empty schema, then S must have one of only two possible values, according to whether the constraint in the schema is *true* or *false*.

$$S \in \mathbb{P}[] \Rightarrow S = [|\ true] \lor S = [|\ false]$$

A use of this schema S as a predicate is equivalent to $\theta S \in S$. In the following, the truth of this predicate is investigated for the two values of S. Here, the $\equiv$ symbol is used to denote an equivalence transformation.

$$\begin{aligned}
&\theta[|\ true] \in [|\ true] \\
&\equiv \theta[|\ true] \in \{(\!|\,|\!)\} \\
&\equiv (\!|\,|\!) \in \{(\!|\,|\!)\} \\
&\equiv true
\end{aligned}$$

$$\begin{aligned}
&\theta[|\ false] \in [|\ false] \\
&\equiv \theta[|\ false] \in \{\} \\
&\equiv (\!|\,|\!) \in \{\} \\
&\equiv false
\end{aligned}$$

The type of an empty schema can be seen from the above to be isomorphic to the Booleans. The following definitions would thus make sense, though they do not appear within ISO Standard Z's toolkit.

$$\begin{aligned}
&False == [|\ false] \\
&True == [|\ true] \\
&Boolean == \{False, True\}
\end{aligned}$$

Z has never provided a *Boolean* type because when writing abstract specifications it is bad style: it is better to use relations and test for membership of those relations. However, Z can also be used in other circumstances. For example, the author was once involved in a project that involved translation of another notation to Z, where that other notation did not share Z's syntactic distinction between predicates and expressions. Wherever a predicate p was used as a Boolean-valued expression, empty schemas enabled the straightforward translation to $[\![p]\!]$. Another example is refinement toward code, where the code will use a Boolean data type. The definition of *Boolean* given above is superior to the occasionally used free type $Boolean ::= False \mid True$ because the definition as schemas allows use as predicates.

To summarise: ISO Standard Z adds notation for empty schemas; Booleans are rarely appropriate for use in abstract specifications and are not defined by ISO Standard Z, but if needed, the definition given above is recommended.

3.8. Local constant declarations

ZRM's restriction on the use of == notation to the global level is an unnecessary irregularity. ISO Standard Z has removed this restriction, allowing use of == in schema texts. A declaration $i == e$ is equivalent to the declaration $i : \{e\}$.

If all declarations in a schema text are of the == form, then unique satisfiability is immediately guaranteed. When used in quantified predicates or definite description (or let) expressions, this provides a neat notation for expressing sub-

stitutions.

$$\exists\, x == 42;\ y == 1998 \bullet p$$
$$\mu\, x == 42;\ y == 1998 \bullet e$$
$$\mathrm{let}\, x == 42;\ y == 1998 \bullet e$$

This generalization makes let notation redundant, but it is retained for backwards compatibility with ZRM (though only the expression form is retained, as explained below).

3.9. Axiom-parts as predicates

ZRM permits newline between outermost conjuncts in an Axiom-part. ISO Standard Z removes the unnecessary irregularity that distinguishes Axiom-parts from other predicates by permitting newline between any predicates to mean conjunction, and giving newline a very low precedence, so that any such new uses of newline must be parenthesized.

3.10. Soft newlines

Newlines serve two different purposes in Z: so-called hard newlines separate declarations and conjuncts; so-called soft newlines merely break up long formulae onto multiple lines without themselves having any semantic significance. In ZRM notation, newlines are soft if they are adjacent to an infix operator. In ISO Standard Z, newlines are also soft if they follow a prefix operator or precede a postfix

operator. This improvement recognises the other circumstances where the next line must contain a continuation of the same formula.

3.11. Toolkit

The major inadequacy in ZRM's toolkit is the omission of a definition of the numeric operations. This omission is resolved in ISO Standard Z as follows. The integers $\mathbb{Z}$ have been replaced as the basis for numeric operations by the set arithmos $\mathbb{A}$ denoting an unrestricted concept of number. $\mathbb{A}$ is introduced in the prelude section, where it provides a basis for the semantics of natural number literals in the Z core language. Other kinds of numbers can be defined as members of $\mathbb{A}$, such as reals and rationals, but those are not included in the ISO Standard toolkit.

With the introduction of the sections notation, any misconception that the mathematical toolkit is the only toolkit can be readily dismissed. Moreover, the informal section structure of ZRM's toolkit is reflected in formal sections within ISO Standard Z's toolkit, namely *set_toolkit*, *number_toolkit*, and *sequence_toolkit*, with *standard_toolkit* being a section that has those three as parents without contributing any additional operations.

section *standard_toolkit* parents *set_toolkit*, *number_toolkit*, *sequence_toolkit*

The absence of a section of bag operations within *standard_toolkit* arises from a reassessment of which operations have been frequently reused and therefore

deserve to be in *standard_toolkit*. The operations dealing with finiteness have been redefined so that they do not refer to numbers, thus allowing *set_toolkit* and *number_toolkit* to be cited individually when the whole *standard_toolkit* is not needed.

4. Incompatibilities

This section lists backwards incompatibilities between ZRM and ISO Standard Z arising from the improvements discussed in the previous section. For each incompatibility is given an explanation of *what* it is, a rationale for *why* it exists, and some notes on *how* instances of it can be detected and rectified.

4.1. Singleton sets

4.1.0.1. What? The notation $\{i\}$ where i is the name of a schema is parsed differently: ISO Standard Z parses it as a singleton set extension, whereas ZRM parses it as a set comprehension. (In ZRM, the set extension (set display) containing a single schema reference is written $\{(i)\}$.)

4.1.0.2. Why? ISO Standard Z permits any schema-valued expression to be written wherever ZRM permits only a schema reference, so the potential ambiguity between singleton set extensions and set comprehensions is broadened to

expressions matching the pattern $\{e\}$. Since e can contain parentheses, the ambiguity cannot be resolved in the way ZRM resolves it. (Where e is not a schema name, both parse $\{e\}$ as a set extension.)

4.1.0.3. How? The type of the ZRM set comprehension $\{i\}$ is that of a set of bindings, whereas the type of the ISO Standard Z set extension $\{i\}$ is that of a set of sets of bindings, so a typechecking tool will detect and report instances of this incompatibility. The ISO Standard Z coercion to a set comprehension is to write $\{i \mid true\}$.

4.2. Schema definition symbol

4.2.0.4. What? The $\widehat{=}$ symbol is not recognised in ISO Standard Z.

4.2.0.5. Why? Schema definitions and abbreviation definitions have been unified, and the separate name-spaces for schema names and variable names have been merged, hence only one of the == and $\widehat{=}$ symbols is needed. The current syntax retains ==, making $\widehat{=}$ obsolete.

4.2.0.6. How? As $\widehat{=}$ is not used in ISO Standard Z, all cases will be reported by a syntax checking tool. Each must be replaced by ==.

4.3. Decorated references to schemas

4.3.0.7. What? Any decoration on a reference to a schema must be separated from the schema name.

4.3.0.8. Why? The merging of schemas and expressions has included the merging of the name-spaces of schema names and other names, so a schema can now be defined with a decoration within its name. For each expression comprising a name with a decoration, there are two possible intentions and hence interpretations: either the name refers to a schema declaration in the environment and the decoration is to be applied to the components of that schema, or the decorated name refers to a schema declaration in the environment whose name is itself decorated. ISO Standard Z must be able to express either intention, whereas ZRM-compliant specifications have only the former possibility.

4.3.0.9. How? ISO Standard Z distinguishes the two intentions by the presence or absence of separation between the name and decoration. That separation can be either white space or parentheses around the name. For example, consider the parentheses in the following.

$$S == [x : \mathbb{N}]$$

$$S' == [x : \mathbb{N}]$$

$$T == (S)' \land S'$$

The expression $(S)'$ is a reference to schema S with its components decorated, giving $[x' : \mathbb{N}]$, whereas the expression S' is a reference to the schema S'. $(S)'$ could equally have been written $S\ '$. Separation is needed to get the ZRM interpretation, but ZRM-compliant specifications might not have that separation. If there is no white space, then a type-checking tool will report that the decorated schema name is not declared. (A typechecker cannot sort this out, as it does not know that the specification is in ZRM notation, both interpretations being reasonable in ISO Standard Z.)

4.4. Decorated references to generic schemas

4.4.0.10. What? The decoration and instantiation on a reference to a generic schema must be reversed, for example, $S'[\mathbb{N}]$ must be changed to $S[\mathbb{N}]'$.

4.4.0.11. Why? ZRM notation treats both the decoration and the instantiation as part of a schema reference, whereas ISO Standard Z notation treats the decoration as an expression. As only the reference can be instantiated, not an expression, the instantiation must be done first.

4.4.0.12. How? A syntax checking tool will recognise a decoration expression, and will then be able to recognise an instantiation list (that being distinct from a schema construction expression, and can appear only in different contexts to generic parameter lists), but will be unable to recognise their juxtaposition, so

a syntax error is guaranteed. The decoration and the instantiation must be reversed.

4.5. let on predicates

4.5.0.13. What? The let notation introduced in ZRM (second edition) cannot be used as a predicate in ISO Standard Z.

4.5.0.14. Why? In ZRM notation, in a context where a predicate is expected, a let with a schema name after its • could be parsed as either a let expression used as a predicate or as a let predicate with a schema name used as a predicate, but both have the same meaning. In ISO Standard Z, any schema expression can be used after the •, and so there can be free variables in that part, leading to different meanings depending on whether the let is taken to be an expression or a predicate. So ISO Standard Z cannot have both let expressions and let predicates. The expression form is the one retained, as it allows some uses of μ to be avoided, μ being disliked if only because it is a Greek letter.

4.5.0.15. How? A syntax checking tool will detect some uses of let on predicates, but might mistake some uses of let on relational predicates as let on the leading expression. Each use of let on a predicate should be replaced by $\exists$ (or by $\exists_1$ or $\forall$ since all mean the same given that the quantified declarations are all $==$ declarations).

4.6. Renaming

4.6.0.16. What? The square-bracketted renaming notation on theta expressions, that was introduced in 2nd edition ZRM, is parsed differently in ISO Standard Z.

4.6.0.17. Why? ISO Standard Z's merging of the syntaxes of schemas and expressions has resulted in a single renaming production that permits renaming of any expression, with the type constraint that the expression be a schema. The renaming production has lower precedence than that of θ, so $\theta S[new/old, ...]$ is parsed as $(\theta S)[new/old, ...]$.

4.6.0.18. How? Since θS is a binding not a schema and renaming is permitted only of a schema not a binding, the renaming of θS will always be detected as a type error. The ZRM notation $\theta S[new/old]$ denotes the binding $\langle\!|\ old == new\ |\!\rangle$. (It is interesting to note that the effect of the notation is not to rename the name on the left of the == but rather to substitute for the value on the right.) That same binding can be built in ISO Standard Z using the notation let $old == new \bullet \theta S$ (which with the addition of surrounding parentheses would be valid ZRM notation too).

4.7. Underlined infix relations

4.7.0.19. What? The underlining notation for infix relational operators, introduced in ZRM second edition, cannot be used in ISO Standard Z.

4.7.0.20. Why? In ZRM notation, there is no way of introducing new operator notation, so instead each use of an identifier as an infix relation can be underlined to make clear that it is being used as an infix symbol. In ISO Standard Z, operator template paragraphs provide a way of introducing new operator notation, so there is no need to mark uses of it as such. Moreover, the underlining notation has not caught on, and it does not help with operators other than infix relations.

4.7.0.21. How? A syntax checking tool will detect all uses of underlining notation. Each underlined infix relational operator should be declared in an earlier operator template paragraph.

4.8. Operator precedences

4.8.0.22. What? The following table enumerates the relative precedences of the predicate and expression notations in ZRM and ISO Standard Z, from lowest at the top to highest at the bottom, revealing some differences. Schema expression notations of ZRM are omitted, as they appear in separate contexts.

ZRM	*ISO Standard Z*
	newline
$\bullet$	$\bullet$
$\mid$	$\mid$
;	;
:	: ==
$\Leftrightarrow$	$\Leftrightarrow$
$\Rightarrow$	$\Rightarrow$
$\vee$	$\vee$
$\wedge$	$\wedge$
$\neg$	$\neg$
pre	
prefix and infix relations	*relational predicates*
if then else	*if then else*
	$\gg$
	$\circ$
	9
	pre
infix generics	*Operator...*
$\times$	*...templates...*
infix functions	*...with...*
$\mathbb{P}$	*...* $\times$ $\mathbb{P}$ *etc...*
prefix generics	*...at...*
$(- \_)$	*...same...*
$(\_ (\mid \_ \mid))$	*...precedence.*
juxtaposed function application	*juxtaposed function application*
postfix functions	

The relative precedences of ZRM's schema calculus operations are, from lowest to highest, $\gg$, $\overset{o}{9}$, $\setminus$, $\upharpoonright$, $\Leftrightarrow$, $\Rightarrow$, $\vee$, $\wedge$, *pre*, $\neg$.

4.8.0.23. Why? Operator templates cause several rows of the table to appear to be different, but in fact the only change in relative precedence caused by them is that between *juxtaposed function applications* and *postfix functions*. The merging of schema expressions with expressions is the cause of most of the differences. Many schema expressions use the same operators as quantified or logical predicates. In ISO Standard Z, one of these schemas used as a predicate is equivalent to the corresponding predicate involving the operand schemas used as predicates. By using the same precedences, that ambiguity can be resolved arbitrarily. The remaining schema operators $\gg$, $\overset{o}{9}$, $\setminus$, $\upharpoonright$, *pre*) are given precedences adjacent to those of other expression-forming (functional) operators, and hence bind more tightly than they do in ZRM. The ZRM column omits some operators (namely *newline*, $==$, *decoration*, and *renaming*) because ZRM notation permits their use in only restricted contexts.

4.8.0.24. How? Type errors are likely but not guaranteed.

4.9. Semicolon between predicates

4.9.0.25. What? Predicates can no longer be conjoined by semicolons.

4.9.0.26. Why? Semicolon has been used so seldom between predicates that it is better not to allow it than to leave readers pondering when it is used. Moreover, *newline* is now permitted between any pair of predicates, not only outermost conjuncts in an Axiom-part, so there is still a lower-precedence alternative to $\wedge$.

4.9.0.27. How? A syntax checking tool can distinguish declarations from predicates by context—where a declaration is acceptable, any predicate must be preceded by |—and so all cases of semicolons between predicates can be detected. Each must be replaced by $\wedge$ or *newline*.

4.10. Theta expressions

4.10.0.28. What? ISO Standard Z requires the types of components in the operand schema to be the same as the types of the same names in the current environment, unlike ZRM.

4.10.0.29. Why? Mismatching types is likely to be indicative of a mistake, and, in those cases where it isn't, binding extensions provide an alternative notation.

4.10.0.30. How? A typechecker will detect and report all such problems.

4.11. Defunct toolkit operations

4.11.0.31. What? The toolkit no longer provides *prefix*, *suffix*, *in*, or operations on bags, except *items*, which now appears in *sequence_toolkit*, where it appears with an extra generic parameter to describe the type of the domain of its argument.

4.11.0.32. Why? Operations that are not reused frequently do not deserve to remain in the toolkit.

4.11.0.33. How? A typechecking tool will detect uses of these operations as undefined names. The operations can still be provided, but in another Z section, separate from the toolkit.

4.12. Lexis of words

4.12.0.34. What? The lexis of words has changed, resulting in, for example, $\lambda\, x$ being lexed as a single word.

4.12.0.35. Why? ISO Standard Z's description of the Z lexis is at two levels: Z characters and Z tokens. The class of Z characters encompasses all UNICODE characters, both λ and x being viewed as letters. Some existing mark-ups do not conform to the two-level view, defining instead mark-up for whole Z tokens. For

example, the widely used mark-up $\backslash dom$ should really be dom, and $\frown\!\!/$ should be formed from the mark-ups of the two characters. This issue is unlikely to be a problem in practice, since existing tools will be able to claim conformance at the syntactic level regardless of mark-up.

4.12.0.36. How? Ideally, white space should be inserted in the $\lambda\, x$ example to avoid a syntax error.

5. Subtle changes

This section discusses some subtle changes in the interpretation of certain notations that nevertheless leave the semantics of ZRM notation unchanged.

5.1. Quantified expressions

ZRM notation's requirement that a schema quantification should quantify only names that are declared within the schema after the $\bullet$ is relaxed in ISO Standard Z, for consistency with the scope rules of quantified predicates. For example, the expression $\forall\, x : \mathbb{A} \bullet [y : \mathbb{A}]$ is illegal in ZRM but legal in ISO Standard Z.

5.2. Preconditions

ZRM notation has *pre* predicates and *pre* schema expressions. ISO Standard Z has only *pre* expressions. ZRM *pre* predicates are parsed as expressions, and those expressions are treated as schema predicates, giving a backwards-compatible effect.

5.3. Schema instantiation

References to generic schemas no longer have to be given explicit instantiations, so long as the instantiations can be determined from the context. On the other hand, a reference to a generic schema in a theta expression is now permitted to have an explicit instantiation.

5.4. Precedence of lambda and mu

ZRM notation requires all λ and μ expressions to be parenthesized. ISO Standard Z gives them precedences, so that parentheses can often be omitted. Parentheses are still required in the case of a μ expression whose $\bullet$ part is omitted.

6. Conclusions

Several innovations in ISO Standard Z relative to ZRM have been presented. These are at the cost of some backwards incompatibilities. Those incompatibilities have been explained and justified, and advice has been given on how to detect and resolve instances of them in existing Z specifications. It is hoped that the innovations presented resolve satisfactorily the known inadequacies in the notation of ZRM, and that this paper will assist their adoption into practice. Inevitably, the final CD will not be the last word on Z—the Z panel has already discussed some other issues, without coming to agreements yet.

IT 18-Jan-2002