The Z of ZETA

Robert Büssow and Wolfgang Grieskamp

July 26, 2000 – for version 1.5 of ZETA

Abstract

The ZETA[1] environment provides tool support for the Z notation [1], implementing the upcoming Z Standard [2]. This document describes the Z of ZETA (called ZETA-Z), inclusive of its lexical representation in LaTeX, and type-setting with ZETA's LaTeX style. The document also describes differences to traditional Spivey-Z[1], and to the Z supported by the ESZ type checker version 2 (a predecessor of the ZETA toolkit), called ESZ2-Z.

Contents

[1]zeta.html

1 Upgrading?

Upgrading from Spivey-Z or ESZ2-Z to ZETA-Z is relative easy, since the type-checker will detect incompatibilities for you. The most common problem is the use of primed schema names, S'. In Standard-Z, the $'$ is part of a name. To refer to the operator $'$ which primes all the names in a schema, it is necessary to put a space (soft, or hard, `~`) between the schema name and the $'$, as in $S\,'$ (`S~'`). You may also put braces around the schema name, such as in $(S)'$, to make the difference better visible in the type-setted specification. The places where this fix is required are usually flagged as type-errors by the type-checker. An other problem is that **let** is not allowed in predicates anymore. It can be replaced by $\exists_1$ since abbreviation (==) are allowed in declarations now.

It should be noted that ZETA-Z has a few restrictions compared to the currently proposed Standard, and – what may be a more serious problem – has some *extensions* which are (currently) not part of the Standard. For a discussion of the restrictions and extensions, see Section 4.5.

Note: If you are upgrading from ESZ or older versions of ZETA, you do *not* need to place the directive

```
%%toolkit "zrm"
```

in the preamble of your document any longer. ZETA 1.5 comes with only one toolkit, which is selected by default. This toolkit is compatible to Spivey-Z and the ZRM (which is also recommended by the Standard), with the restriction that it does *not* support bags any longer (as the Standard does not support them).

2 Novice to LaTeX-Based Z?

Say `\usepackage{zeta}` in the preamble of your LaTeX document to include the ZETA style. There are some style options that can be used, e.g., `\usepackage[oxsz]{zeta}`. Refer to Section 3.3 for a list of options. In a Z document, Z paragraphs can be arbitrarily mixed with explanatory text. ZETA-Z provides all the environments to introduce Z paragraphs as they are known from LaTeX based Z representations. Normal schemata are typeset using the `schema` environment. Line breaks have to be explicitly specified with `\\`.

$$\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
x, y : \mathbb{N} \\
X : \mathbb{P}\,\mathbb{Z} \\ \hline
x \in X \land X = \{\, z : \mathbb{Z} \mid z \leq y \bullet 2 * z \,\} \\ \hline
\end{array}$$

```
\begin{schema}{S}
    x, y : \nat \\
    X : \power \num
\where
    x \in X \land
    X = \{~z : \num | z \leq y @ 2*z~\}
\end{schema}
```

For generic schemata, the formal generic parameter is written directly behind the schema name (this is in contrary to the proposed Standard-Z, where generic parameters are written as `\begin{schema}{G}[X]`[2]:

$$\begin{array}{|l}
\multicolumn{1}{l}{G[X]} \\ \hline
x : X \\ \hline
\end{array}$$

```
\begin{schema}{G[X]}
    x : X
\end{schema}
```

[2]The next version of ZETA might support this notation for compatibility reasons.

Abbreviations, given and free types are put in `zed` environments. In Standard-Z, schema expressions are treated as expressions, therefore "$\widehat{=}$" is not needed anymore. (Nevertheless, it is still supported for compatibility.):

$$\begin{array}{l}
T == [a, b : \mathbb{Z} \mid a > b] \\
EVEN == \{ x : \mathbb{Z} \mid \exists y : \mathbb{Z} \bullet x = 2 * y \} \\
[\,TYPE\,] \\
COLOR ::= red \mid blue \mid yellow
\end{array}$$

```
\begin{zed}
    T == [ a, b : \num | a > b ] \\
    EVEN ==
      \{~x : \num |
         \exists y : \num @ x=2*y~\} \\
    [ TYPE ] \\
    COLOR ::= red | blue | yellow
\end{zed}
```

Axiomatic definitions are written with the `axdef` environment. Formal generic parameters can be specified by an optional environment parameter. For compatibility with Spivey-Z and Standard-Z, generic axiomatic definitions can also be typed with the `gendef` environment.

$$\begin{array}{|l}
limit : \mathbb{N} \\
\hline
limit \geq 4711
\end{array}$$

```
\begin{axdef}
    limit : \nat
\where
    limit \geq 4711
\end{axdef}
```

$$\begin{array}{|l}
\hline\hline [X] \\
f : X \leftrightarrow \mathbb{N} \\
\hline
\end{array}$$

```
\begin{axdef}[X]
    f : X \rel \nat
\end{axdef}
```

$$\begin{array}{|l}
\hline\hline [X] \\
f : X \leftrightarrow \mathbb{N} \\
\hline
\end{array}$$

```
\begin{gendef}[X]
    f : X \rel \nat
\end{gendef}
```

For a listing of the full Z syntax and Z symbols in LaTeX, see Appendix A and Appendix B.

3 The Style: Details

3.1 Tabbing and Alignment

There are two ways of setting indents. You can either use the `\t`n (put n in braces, if $n \geq 10$) to insert a space of n `\zedtabs` or use the `\<` and `\>` markers for relative indentation:

$$\begin{array}{|l}
f : \mathbb{Z} \rightarrowtail\!\!\!\!+ \mathbb{Z} \\
\hline
\forall x : \mathbb{N} \bullet \\
\qquad x \in \operatorname{dom} f \land \\
\qquad x < f\ x
\end{array}$$

```
\begin{axdef}
    f : \num \pfun \num
\where
    \forall x : \nat @ \\ \t1
        x \in \dom f \land \\ \t1
        x < f~x
\end{axdef}
```

$$\begin{array}{|l}
f : \mathbb{Z} \pfun \mathbb{Z} \\
\hline
\forall x : \mathbb{N} \bullet x \in \operatorname{dom} f \land \\
\qquad\qquad x < f\, x
\end{array}$$

```
\begin{axdef}
    f : \num \pfun \num
\where
    \forall x : \nat @ \<
        x \in \dom f \land \\
        x < f~x \>
\end{axdef}
```

Both kind of markups are ignored by the type-checker.

A `zed` environment also introduces a tabular, thus definitions can be aligned:

$$\begin{array}{rcl}
TREE & ::= & Leaf \\
 & | & Node \langle\!\langle Left : Tree, Right : Tree \rangle\!\rangle
\end{array}$$

```
\begin{zed}
    TREE &::=& Leaf \\
         & | & Node
                \ldata Left : Tree,
                       Right : Tree
                \rdata
\end{zed}
```

The environment `syntax` known from Spivey-Z is just an alias for the `zed` environment.

Vertical space between lines can be introduced by using `\also`, `\Also`, and `\ALSO` instead of `\\`. A vertical space of `\jot`, `2\jot`, resp. `4\jot` is inserted, where `\jot=3pt` by default.

3.2 Side by Side

For small schemata and definitions, it is usually desirable to set them side by side. The `zeta` style introduces the `zedgroup` environment that sets paragraphs automatically side by side, if they fit.

$$\begin{array}{|l|}
\hline
A \\
a : \mathbb{N} \\
\hline
\end{array}
\qquad
\begin{array}{|l|}
\hline
B \\
b : \mathbb{N} \\
\hline
b > 4 \\
\hline
\end{array}$$

```
\begin{zedgroup}
\begin{schema}{A}
    a : \nat
\end{schema}
\begin{schema}{B}
    b : \nat
\where
    b > 4
\end{schema}
\end{zedgroup}
```

Line breaks between paragraphs can be forced with `\\`. In order to produce a regular layout, schemata in the same row are set with the same height and there is a limited predefined set of possible paragraph width, i.e. full width, half width, a third, and a quarter. `zedgroup` environments can not be nested. Note that due the implementation of `zedgroup`, errors in the LaTeX code may lead to error messages with inaccurate positions.

For downwards compatibility with other styles, there is also a `sidebyside` environment. You have to specify explicitly with `\nextside` which portions are supposed to be set side by side. If more than two columns are required, the number of columns has to be specified, e.g. `\begin{sidebyside}[3]` for three columns.

A

$a : \mathbb{N}$

B

$b : \mathbb{N}$

$b > 4$

```
\begin{sidebyside}
\begin{schema}{A}
    a : \nat
\end{schema}
\nextside
\begin{schema}{B}
    b : \nat
\where
    b > 4
\end{schema}
\end{sidebyside}
```

3.3 Style Options and Parameters

- `oxsz`: use `oxsz` charset (you need to have an installation of this font, which originates from the fUZZ package). If not given, special Z symbols will be simulated using existing symbols (as in the `oz` style).

The layout of the Z boxes and formulae can be manipulated by various TEX parameters:

- `\zedindent`: left indent of Z paragraphs.
- `\zedbeforeskip`: space before Z paragraphs.
- `\zedafterskip`: space after Z paragraphs.
- `\zedtab`: horizontal space that is introduced by `\t1`
- `\zedleftsep`: horizontal space between left vertical line of schemata and axiomatic definitions and the beginning of declarations and predicates.
- `\zeddeclpartskip`: vertical space between top horizontal line of a schema and the declarations.
- `\zedlinethickness`: thickness of schema and axiomatic definition box's lines.

4 The Language: Details

4.1 Sections

Z sections are specified with the `\zsection` command. Here, the section *BirthdayBook* is defined. The parent of the section is the *library* section; if it is omitted, the standard toolkit is used as the parent (the standard toolkit to be used can be overwritten with the directive `%%toolkit`, see below):

section *BirthdayBook* **parents** *library*

```
\zsection[library]{BirthdayBook}
```

All paragraphs following the `\zsection` command belong to this section, up to next `\zsection` command. This also holds across LATEX inclusion commands (`\input` and `\include`). You can arbitrarily switch between sections, as in the example below, where we first add some declarations to the section *BirthdayBook*, then the *BirthdayBookExec*, and then again to *BirthdayBook*:

```
\zsection[library]{BirthdayBook}
\input{basic}
\zsection[BirthdayBook]{BirthdayBookExec}
\input{exec}
\zsection[library]{BirthdayBook}
\input{addtobasic}
```

If a document does not contain a `\zsection`, then one is implicitly created named after the file name of the document.

4.2 Toolkits

Each Z section has the section *Toolkit* as an implicit parent. In order to change the name of this implicite parent, the type-checker directive `\zsection` is used. This directive needs to placed in the preamble of the document, before the first Z section or any other Z markup, such as in:

```
  \documentclass{article}
  %%toolkit "ExtendedToolkit"
  ...
```

If the string given to `%%toolkit` is empty, not implicite parent will be created. This is useful to define toolkits by your own.

4.3 Operator Templates

Z prefix (e.g. -4), postfix (e.g. $R^{\sim}$), infix (e.g. $47+11$) and nofix (e.g. $\langle 4, 5, a \rangle$) operators can be declared by Standard-Z operator templates. These are given in LaTeX by the commands `\zfunction`, `\zrelation`, resp. `\zgeneric`. The templates are declared with three arguments, (1) the precedence, (2) the associativity, and (3) the template itself. The precedence is omitted for relations and nofix operators. The associativity is only given for infix functions and generics. The template is build up by ordinary identifiers and place holders for single parameters (i.e. `\_`) and comma separated lists of parameters (i.e. `,,`):

function 40 leftassoc $(\_ * \_)$	`\zfunction{40 \leftassoc (\_*\_)} \\`
function 30 leftassoc $(\_ + \_)$	`\zfunction{30 \leftassoc (\_+\_)} \\`
function 90$(_^{\sim})$	`\zfunction{90 (\_\inv)} \\`
function $(\langle ,, \rangle)$	`\zfunction{(\langle ,, \rangle)} \\`
relation $(\_ \subseteq \_)$	`\zrelation{(\_\subseteq\_)} \\`
generic 5 rightassoc $(\_ \longrightarrow \_)$	`\zgeneric{5 \rightassoc (\_\fun\_)}`

Note that operator template commands can appear anywhere in a document, but they must not appear inside of Z environments such as `zed`. The ZETA-style provides the environment `zdirectives` in order to group operator templates; it behaves like `zed` regarding type-setting, but has no special meaning for the type checker.

The directives for introducing operator templates known from Spivey-Z and ESZ2 are still supported, by internally translating them to the according Standard-Z templates. The mapping is as follows:

- `%%inrel word` $\Longrightarrow$ **relation** $(\_\mathit{word}\_)$
- `%%prerel word` $\Longrightarrow$ **relation** $(\_\mathit{word})$
- `%%ingen word` $\Longrightarrow$ **generic** 2 rightassoc $(\_\mathit{word}\_)$

- `%%pregen word` $\Longrightarrow$ **generic** $90(\_word)$
- `%%inop word N` $\Longrightarrow$ **function** $10 + (10 * N)$ leftassoc $(\_word\_)$
- `%%texrel N \tok` $\Longrightarrow$ `\zrelation{9999 (\tok {\_} ... {\_})}`
- `%%texgen N \tok` $\Longrightarrow$ `\zgeneric{9999 (\tok {\_} ... {\_})}`
- `%%texop N \tok` $\Longrightarrow$ `\zfunction{9999 (\tok {\_} ... {\_})}`

The ZeTAstyle offers some auxiliar macros to define styles of new operators:

- `\Zkeyword`: define a keyword.
- `\Zinop`: define an infix operation, such as div.
- `\Zpreop`: define a function or prefix operations, such as dom or pre.
- `\Zinrel`: define an infix relation, such as partition.
- `\Zprerel`: define a prefix relation, such as disjoint.

Use these macros as in:

```
\newcommand{\myinop}{\Zinop{op}}
```

The effect is a selection of an appropriate font and spacing properties of the introduced LaTeXtoken in math.

4.4 Further Standard Z Features and Notes on Upgrading

ZeTA-Z supports the Standard-Z language (beside of a few incompatibilities described in the next Section). Among Z-sections and operator templates, the most important features are the followings:

- unification of value expressions with schema expressions. A schema expression is just an expression which denotes a set of bindings.
- usage of expressions where in traditional Z only schema references are allowed: any expression denoting a set of bindings can be used as a schema reference
- schema decoration $(S\,')$ as an operator
- λ and μ expressions bind tighter then binary relations. In particular, it is possible to write $f = \lambda\, x : \mathbb{N} \bullet E$ instead of the formerly required $f = (\lambda\, x : \mathbb{N} \bullet E)$
- It is possible to provide an empty declaration list in schema text or axiomatic definitions. In particular, one can write:

$$\begin{array}{|l}
[X] \\
\hline
f = \lambda\, x : X \bullet E \\
\hline
\end{array}$$

Users upgrading from Spivey-Z or ESZ-Z to ZeTA-Z (resp. to Standard-Z in general) have to tackle with the following typical incompatibilities (for a comprehensive discussion, see [2]):

- decoration is now part of a name. Thus S' denotes the name S', not the schema S where all variables are decorated with $'$. In order to denote the decoration operators on schemas, one either uses space, $S\ '$, or braces, $(S)'$ (in the later example, S can be in fact an arbitrary expression denoting a schema).
- a μ-expression without a $\bullet$ must always be put in parentheses $((\mu\, E))$

These incompatibilities are straight-forward detected by a type checker. Experiences show that around 90% of the problems when upgrading a Z specification are related to the usage of S', which is commonly applied in expressions such as $(\theta S, \theta S')$ (now to be written as $(\theta S, \theta S\ ')$).

4.5 Extended ZeTa-Z Features and Incompatibilities with the Standard

Standard-Z is still a moving target. ZeTa-Z does not conform completely to it, and adds extensions which are currently not found in the Standard.

The following *restrictions* compared to the Standard are currently present:

1. the LaTeX-lexis of ZeTa-Z is oriented towards Spivey-Z, and does not support the recently proposed features such as multi-token-words. The LaTeX-tokens `{` and `}` are not treated as whitespace, but as normal tokens[3]. The lexis will by synchronized with the Standard in one of the next revisions of ZeTa.
2. it is not possible to redefine the meaning of ΔS and ΞS. In, fact Δ and Ξ are provided as operators on schemas (see below).
3. there are some subtle restriction in operator templates (which shouldn't become visible to users, however). These will be synchronized with the Standard in future revisions.

The following extensions compared to the Standard are currently found in ZeTa-Z:

1. The order how paragraphs appear in a Z section is arbitrary, apart of that the definition-use relation of paragraphs must be acyclic. (A paragraph is in definition-use relation to another paragraph, if it introduces a name which is referred to by the other paragraph.)
2. Δ and Ξ are introduced as expression operators. An explicite definition of a schema named ΔS (ΞS) is not supported.
3. Mutual recursive free types may appear in different paragraphs, and not just one paragraph where they are separated by `&`. It is in general possible to refer to Z names in a free type definition which are themselves defined in dependency of this free type.
4. Global constants can be declared multiple times in the environment of a section. All declarations must be type-compatible.
5. Given and free types may be generic. A generic given type is written as $[T[X]]$. A generic free type is written as $T[X] ::= \ldots$.

Some of these extensions have been proposed to the Z ISO Panel, and some of them might make their way into the final Standard. Yet, be aware that when using these extensions, your specification might become incompatible with Standard Z[4]

[3]This is to be downwards-compatible with ESZ2, to support its LaTeX-style operator templates, `%%texop` etc. The use of these templates is discouraged, and its support will be canceled in the future. Use the `%%macro` directive to simulate similar effects.

[4]The type checker currently doesn't warns you when using ZeTa extensions. This may be fixed in one of the next releases.

4.6 The Denotation Type

A further extension of ZETA-Z is a builtin notation and type for *denotations* (strings). A denotation is written as follows:

$$\begin{array}{|l} d : \mathsf{denotation} \\ \hline d = \text{"}\texttt{Hello World! How do You do?}\text{"} \end{array}$$

```
\begin{axdef}
  d: \denotation
\where
  d = \ZD{Hello World! How do You do?}
\end{axdef}
```

Inside of a denotation `\ZD{...}`, the following escape sequences can be used:

- `\n` – newline
- `\r` – carriage return
- `\t` – tabulator
- `\b` – backspace
- `\f` – formfeed
- `\\` – backslash
- `\{` – braceopen
- `\}` – braceclose

Only two functions are available on denotations: dec d converts a denotation in a sequence of natural numbers, where each number represents the encoding of the given letter, whereas enc s converts a sequence of numbers into a denotation. The encoding is unspecified.

$$\begin{array}{|l} \mathsf{enc} : \mathsf{denotation} \longrightarrow \mathrm{seq}\,\mathbb{N} \\ \mathsf{dec} : \mathrm{seq}\,\mathbb{N} \nrightarrow \mathsf{denotation} \end{array}$$

```
\begin{axdef}
  \denc: \denotation \fun \seq \nat \\
  \ddec: \seq \nat \pfun \denotation
\end{axdef}
```

5 Type-Checker Directives

A type-checker directive starts with `%%` at the beginning of a line and lasts until the end of the line. Beside of the `%%toolkit` directive mentioned in Section 4.2, and the operator template directives described in Section 4.3, the following directives, as known from FUZZ and ESZ2, are supported:

- Text after a double TEX comment followed by a space (`%% ...`) is processed by the type-checker, wheres it is ignored in the type-setted document.
- A simple concept for macro expansion is provided. The directive

  `%%macro` *token n replacement*

  defines the macro *token* with n parameters. The macro is replaced by the *replacement* before the syntax is checked (similar the the C preprocessor). Parameters are referenced with `#1`, `#2`, etc. in the replacement.

  The macro directive can e.g. be used for defining layout tokens or comment commands, such as in

```
%%macro \< 0
%%macro \zcomment 1
```

where the replacement is just empty. The `%%ignore` directive as known from FUZZ is a special case of this usage of macro-directives, and is still supported.

Macros can also be used for introducing new operators, such as the definition of an exclusive or (i.e. `\lxor{x>0}{x\leq 0}`):

```
%%macro \lxor 2 (\lnot (#1 \iff #2))
```

- It is possible to assign to environments the syntactic “role” of an existing Z-environment. This is done by the directive:

  ```
  %%environment new-env-name old-env-name
  ```

  This feature makes it possible to define new Z environments which implement some extra layout functionality (e.g. by generating a margin).

- The `%%line` directive as known from fUZZ is supported. It has the form:

  ```
  %%line "source" $ n
  ```

  The effect is that the next line and all subsequent lines will get locators as if they belong to *source* starting at line n. This is useful if Z is generated from some other input language.

A LaTeX Syntax for Z

This syntax is based on the working draft version 1.5 of the Z standard.

A.1 Specification

```
Specification      ::=  Paragraph ∗ Section∗
Section            ::=  \zsection [ [Parents] ] {NAME} Paragraph∗
Parents            ::=  NAME { , NAME }
```

A.2 Paragraphs

```
Paragraph          ::=  \begin{zed} ZedParagraphs \end{zed}
                    |   \begin{axdef} [ [Formals] ] SchemaDisplayText \end{axdef}
                    |   \begin{gendef} [Formals] SchemaDisplayText \end{gendef}
                    |   \begin{schema}{NAME [ [Formals] ]}
                            SchemaDisplayText
                        \end{schema}
ZedParagraphs      ::=  ZedParagraph
                    |   ZedParagraph Sep ZedParagraphs
ZedParagraph       ::=  [ NAME { , NAME } ]
                    |   EqualDecl
                    |   GenName == Expression
                    |   NAME ::= Branch{ | Branch }
                    |   OperatorTemplate
Branch             ::=  DeclName [ \ldata Expression \rdata ]
Formals            ::=  NAME { , NAME }
Sep                ::=  ;
                    |   \\
```

A.3 Predicates

```
Predicate          ::=  \forall SchemaText @ Predicate
                    |   \exists SchemaText @ Predicate
                    |   \exists_1 SchemaText @ Predicate
                    |   Predicate \\ Predicate
                    |   Predicate ; Predicate
                    |   Predicate \iff Predicate
                    |   Predicate \implies Predicate
                    |   Predicate \lor Predicate
                    |   Predicate \land Predicate
                    |   \lnot Predicate
                    |   Relation
                    |   Expression
                    |   true
                    |   false
                    |   (Predicate)

Relation           ::=  Expression NAME { Expression NAME }
                    |   Expression NAME { Expression NAME } Expression
                    |   NAME Expression { NAME Expression }
```

A.4 Expression

```
Expression     ::=  \forall SchemaText @ Expression
                |   \exists SchemaText @ Expression
                |   \exists_1 SchemaText @ Expression
                |   \lambda SchemaText @ Expression
                |   \mu SchemaText@ Expression
                |   \LET EqualDecl{ ; Equal-Decl } @ Expression
                |   Expression \iff Expression
                |   Expression \implies Expression
                |   Expression \lor Expression
                |   Expression \land Expression
                |   \lnot Expression
                |   \IF Predicate\THEN Expression \ELSE Expression
                |   Expression \semi Expression
                |   Expression \pipe Expression
                |   Expression \hide (Decl-Name { , Decl-Name } )
                |   Expression \project Expression
                |   \pre  Expression
                |   Expression \cross Expression { \cross Expression }
                |   \power Expression
                |   Application
                |   Expression Expression
                |   Expression STROKE
                |   Expression Renaming
                |   Expression.RefName
                |   Expression.NUMBER
                |   \theta Expression { STROKE }
                |   RefName
                |   RefName[ Expression { , Expression } ]
                |   NUMBER
                |   \{ Expression { , Expression } \}
                |   \{ SchemaText  [@ Expression]\}
                |   [ SchemaText ]
                |   \lblot EqualDecl { , EqualDecl } \rblot
                |   ( Expression , Expression { , Expression } )
                |   \mu SchemaText
                |   ( Expression )

Renaming       ::=  [DeclName / DeclName { , DeclName / DeclName } ]
Application    ::=  Expression NAME { Expression NAME }
                |   Expression NAME { Expression NAME } Expression
                |   NAME Expression { NAME Expression }
                |   NAME Expression NAME { Expression NAME }
```

A.5 Declarations

```
SchemaDisplayText   ::=  DeclPart [ \where Predicate ]
SchemaText          ::=  DeclPart [ | Predicate ]
DeclPar             ::=  Declaration { Sep Declaration }
Declaration         ::=  ColonDecl
                     |   EqualDecl
                     |   Expression
ColonDecl           ::=  DeclName { , DeclName } : Expression
EqualDecl           ::=  DeclName == Expression
DeclName            ::=  NAME
                     |   OpName
RefName             ::=  NAME
                     |   ( OpName )
```

A.6 Operator Templates

```
OpName              ::=  \_ NAME { \_ NAME }
                     |   \_ NAME { \_ NAME } \_
                     |   NAME \_ { NAME \_ }
                     |   NAME \_ NAME { \_ NAME }
GenName             ::=  NAME { NAME }

OperatorTemplate    ::=  \zrelation Template
                     |   \zfunction Category-Template
                     |   \zgeneric Category-Template
CategoryTemplate    ::=  Prec  Prefix-Template
                     |   Prec  Postfix-Template
                     |   Prec  Assoc  InfixTemplate
                     |   Nofix-Template
Prec                ::=  NUMBER
Template            ::=  Prefix-Template
                     |   Postfix-Templat
                     |   InfixTemplate
                     |   Nofix-Template
Prefix-Template     ::=  ( NAME { ( \_ | , , ) NAME } \_ )
Post-Template       ::=  ( \_NAME { ( \_ | , , ) NAME } )
Infix-Template      ::=  ( \_NAME { ( \_ | , , ) NAME } \_ )
Nofix-Template      ::=  ( NAME { ( \_ | , , ) NAME } )
```

A.7 Identifier

```
NAME                ::=  \ LETTER LETTER { STROKE }
                     |   Letter {Letter | Digit | \_) } { STROKE }
LETTER              ::=  A | . . . | Z | a | . . . | z
DIGIT               ::=  0 | . . . 9
STROKE              ::=  ' | _Digit | ? | !
NUMBER              ::=  DIGIT { DIGIT }
```

B LaTeX Symbols

			Logic		
¬	`\lnot`	*unary*	pre	`\pre`	*unary*
∧	`\land`	*left*	∨	`\lor`	*left*
⇒	`\implies`	*right*	⇔	`\iff`	*left*
∀	`\forall`	–	∃	`\exists`	–
↾	`\project`	*left*	\	`\hide`	*left*
⨟	`\semi`	*left*	≫	`\pipe`	*left*
if	`\IF`	–	**then**	`\THEN`	–
else	`\ELSE`	–	**let**	`\LET`	–
			Sets		
ℙ	`\power`	*pregen*	𝔽	`\finset`	*pregen*
∅	`\emptyset`	*word*	#	`\#`	*word*
∩	`\cap`	*inop 4*	∪	`\cup`	*inop 4*
\	`\setminus`	*inop 3*			
⊆	`\subseteq`	*inrel*	⊂	`\subset`	*inrel*
=	`=`	*inrel*	≠	`\neq`	*inrel*
∈	`\in`	*inrel*	∉	`\notin`	*inrel*
⋃	`\bigcup`	*word*	⋂	`\bigcap`	*word*
disjoint	`\disjoint`	*prerel*	partition	`\partition`	*inrel*
			Relations and Functions		
↦	`\mapsto`	*inop 1*	×	`\cross`	*inop 1*
dom	`\dom`	*word*	ran	`\ran`	*word*
∘	`\circ`	*inop 4*	⨟	`\comp`	*inop 5*
⊕	`\oplus`	*inop 5*	∼	`\inv`	*postop*
◁	`\dres`	*inop 6*	▷	`\rres`	*inop 6*
⩤	`\ndres`	*inop 6*	⩥	`\nrres`	*inop 6*
+	`\plus`	*postop*	*	`\star`	*postop*
↔	`\rel`	*ingen*	⇸	`\pfun`	*ingen*
→	`\fun`	*ingen*	⤔	`\pinj`	*ingen*
↣	`\inj`	*ingen*	⤀	`\psurj`	*ingen*
↠	`\surj`	*ingen*	⤖	`\bij`	*ingen*
⇻	`\ffun`	*ingen*	⤕	`\finj`	*ingen*
id	`\id`	*word*			
⦇	`\limg`	–	⦈	`\rimg`	–
			Integers		
ℤ	`\num`	*word*	ℕ	`\nat`	*word*
..	`\upto`	*inop 2*			
+	`+`	*inop 3*	−	`-`	*inop 3*
∗	`*`	*inop 4*	div	`\div`	*inop 4*
mod	`\mod`	*inop 4*			
<	`<`	*inrel*	≤	`\leq`	*inrel*
>	`>`	*inrel*	≥	`\geq`	*inrel*
			Sequences		
⟨	`\langle`	–	⟩	`\rangle`	–
seq	`\seq`	*pregen*	iseq	`\iseq`	*pregen*
in	`\inseq`	*inrel*	⁀	`\cat`	*inop 3*
prefix	`\prefix`	*inrel*	suffix	`\suffix`	*inrel*
↾	`\filter`	*word*	↿	`\extract`	*word*

References

[1] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[2] I. Toyn. Innovations in standard Z notation. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98: The Z Formal Specification Notation*, volume 1493 of *Lecture Notes in Computer Science*, pages 193–213. Springer-Verlag, 1998.