A guide to the `zed` style option

Mike Spivey

December 1990

1 Introduction

This document is a guide to the version of the `zed` style option for LaTeX dated 11th December 1990. This new version of the style option is fully compatible with the `fuzz` style option distributed with the my *f*UZZ type-checker for Z, but uses two fonts from the AMS in place of the special Z font distributed with *f*UZZ. Some of the symbols have been cobbled together by combining two or more characters, but the results are good enough for rough drafts. The style option requires the 'old' AMS fonts, and will not at present work with SliTeX or the Schöpf-Mittelbach font selection scheme.

The rest of this guide is mostly extracted from the manual for *f*UZZ, and it assumes a basic knowledge of LaTeX. I have not removed some information about how the *f*UZZ type-checker treats various constructs, in case you later want to type-check a document you have formatted with the `zed` style option. For information about the *f*UZZ package and how to order it, see the end of this guide.

2 Loading the `zed` style option

Every LaTeX document should begin with a `\documentstyle` command. If the document contains a Z specification, this command should include the style option `zed`. For example:

```
\documentstyle[12pt,zed]{article}
```

Including `zed` as a style option loads macros from the file `zed.sty` and also loads four fonts of extra mathematical symbols called `msxm9`, `msym9`, `msxm10`, and `msym10`. Your LaTeX installation must have these fonts for the `zed` style option to work; if it doesn't, or they are in the wrong place to be found by TeX, then you will get an error message like this:

```
! Font \ninsxm=msxm9 not loadable ...
```

The `zed` style option can be used with any of the standard LaTeX styles, and it can appear either before or after the type-size option if one is used. It can be combined with most of the standard style options, but it should not be combined with `fleqn`, because `zed` already makes provision for setting mathematics flush left. At present, `zed` does not work with SliTeX.

3 Making boxes

To print a schema, use the `schema` environment. Here is an example, showing first the input, then the output from LaTeX:

```
\begin{schema}{PhoneDB}
    known: \power NAME \\ phone: NAME \pfun PHONE
\where
    known = \dom phone
\end{schema}
```

PhoneDB

$known : \mathbb{P}\, NAME$
$phone : NAME \nrightarrow PHONE$

$known = \mathrm{dom}\, phone$

The name of the schema appears as an argument to the environment, and the horizontal dividing line between declarations and predicates is indicated by `\where`. Successive lines in the declaration and predicate parts are separated by the command `\\`. In this example, the Z symbols '$\mathbb{P}$', '$\nrightarrow$' and 'dom' have been entered as the commands `\power`, `\pfun` and `\dom`: for a complete list of these commands, see Section **??** below.

Like the `displaymath` environment of LaTeX, the `schema` environment (and the others we shall come to in a moment) can appear in the middle of a paragraph, and ordinarily should have no blank lines either before or after it. Blank lines before the environment are ignored, but blank lines afterwards cause the following text to begin a new paragraph.

For a schema without a predicate part, the command `\where` is simply omitted, as in the following example:

```
\begin{schema}{Document[CHAR]}
    left, right: \seq CHAR
\end{schema}
```

Document[*CHAR*]

$left, right : \mathrm{seq}\, CHAR$

This example also shows how to set schemas with generic parameters.

For axiomatic descriptions, the `axdef` environment is used. Here is an example:

```
\begin{axdef}
    limit: \nat
\where
    limit \leq 65535
\end{axdef}
```

$$\begin{array}{|l}
\mathit{limit} : \mathbb{N} \\
\hline
\mathit{limit} \leq 65535
\end{array}$$

In both kinds of box, predicates and declarations can be split between lines before or after infixed symbols, as shown in the following example:

```
\begin{axdef}
    policy: \power_1 RESOURCE \fun RESOURCE
\where
    \forall S: \power_1 RESOURCE @ \\
\t1         policy(S) \in S
\end{axdef}
```

$$\begin{array}{|l}
\mathit{policy} : \mathbb{P}_1\ \mathit{RESOURCE} \rightarrow \mathit{RESOURCE} \\
\hline
\forall S : \mathbb{P}_1\ \mathit{RESOURCE} \bullet \\
\qquad \mathit{policy}(S) \in S
\end{array}$$

The strange hint `\t1` in this example makes the corresponding line in the output have one helping of indentation. As things get more nested, you can say `\t2`, `\t3`, and so on. But if you should ever get beyond `t9`, you'll need to use braces around the argument: `\t{10}`, and you'd better look for some way to simplify your specification!

This system of tab stops is a little crude, but it is easier to use than the alternatives, and usually gives acceptable results. The `\t`*n* commands are completely ignored by the type-checker, so you are free to use them as you like to improve the look of your specification. The size of 'helping' you get with `\t`*n* is a style parameter `\zedindent`, and the default is 2em.

For generic definitions, there's the `gendef` environment: for example,

```
\begin{gendef}[X,Y]
    first: X \cross Y \fun X
\where
    \forall x: X; y: Y @ \\
\t1     first(x,y) = x
\end{gendef}
```

$$
\begin{array}{|l}
\hline\hline
[X, Y] \\
\mathit{first} : X \times Y \rightarrow X \\
\hline
\forall x : X;\ y : Y \bullet \\
\qquad \mathit{first}(x, y) = x \\
\hline
\end{array}
$$

In this environment, the formal generic parmeters are an optional argument. Omitting this argument results in a box with a solid double bar at the top, which can be used for uniquely defining non-generic constants.

If a schema or other box contains more than one predicate below the line, it often looks better to add a small vertical space between them. This can be done with the command `\also`:

```
\begin{schema}{AddPhone}
    \Delta PhoneDB \\ name?: NAME \\ number?: PHONE
\where
    name? \notin known
\also
    phone' = phone \oplus \{name? \mapsto number?\}
\end{schema}
```

$$
\begin{array}{|l}
\hline
\mathit{AddPhone} \\
\Delta \mathit{PhoneDB} \\
\mathit{name}? : \mathit{NAME} \\
\mathit{number}? : \mathit{PHONE} \\
\hline
\mathit{name}? \notin \mathit{known} \\[1ex]
\mathit{phone}' = \mathit{phone} \oplus \{\mathit{name}? \mapsto \mathit{number}?\} \\
\hline
\end{array}
$$

Some Z paragraphs do not appear in boxes, and for these the `zed` environment is used:

```
\begin{zed}
    [NAME, DATE]
\also
    REPORT ::= ok | unknown \ldata NAME \rdata
\also
    \exists n: NAME @ \\
\t1     birthday(n) \in December.
\end{zed}
```

$$[\mathit{NAME}, \mathit{DATE}]$$

$$\mathit{REPORT} ::= \mathit{ok} \mid \mathit{unknown}\langle\!\langle \mathit{NAME} \rangle\!\rangle$$

$$
\begin{array}{l}
\exists n : \mathit{NAME} \bullet \\
\qquad \mathit{birthday}(n) \in \mathit{December}.
\end{array}
$$

This environment should be used for basic type definitions, constraints, abbreviation definitions, free type definitions, and the horizontal form of schema definitions. As the example illustrates, a full stop or comma is allowed just before the closing `\end` command of any of the Z environments, if that suits your taste (or is forced on you by a publisher's house rules). This punctuation is ignored by the type-checker.

For large free type definitions, the `syntax` environment provides a useful alternative to the `zed` environment, as the following example suggests:

```
\begin{syntax}
    OP & ::= & plus | minus | times | divide
\also
    EXP & ::= & const \ldata \nat \rdata \\
        & | & binop \ldata OP \cross EXP \cross EXP \rdata
\end{syntax}
```

$$
\begin{array}{lll}
OP & ::= & plus \mid minus \mid times \mid divide \\[1ex]
EXP & ::= & const\langle\!\langle \mathbb{N} \rangle\!\rangle \\
 & \mid & binop\langle\!\langle OP \times EXP \times EXP \rangle\!\rangle
\end{array}
$$

Just as in the `eqnarray` environment of LaTeX, the fields are separated by `&` characters, and these are ignored by the type-checker.

4 Inside the boxes

The first thing to notice about the text inside the boxes is that multi-character identifiers look better than they do with ordinary LaTeX: instead of $specifications$, you get *specifications*. The letters are not spread apart, and ligatures like *fi* are used. This is achieved by an adjustment to the way TeX treats letters in mathematical formulas, and no special commands are needed in the input file. Embedded underline characters can be set with the `\_` command, which is also used for dummy arguments of operators: `not\_known` gives *not_known*, and `\_ + \_` gives $\_ + \_$.

The various special symbols of the Z language and library have mnemonic names. Some of these names are the same as in ordinary LaTeX, and some symbols have new names more suggestive of their meaning in Z. The spaces inserted around the symbols have been adjusted to make them look better in Z specifications.

A few symbols have two names, reflecting two different uses for the symbol in Z. The symbol $⨾$ is called `\semi` when it is used as an operation on schemas, and `\comp` when it is used for composition of relations. The symbol $\setminus$ is called `\hide` as the hiding operator of the schema calculus, and `\setminus` for the set difference operator. The symbol $\upharpoonright$ is called `project` as the schema projection operator, and `\filter` for filtering of sequences. The spaces around the schema operations are a little larger, and the type-checker recognizes each name only in the appropriate context.

For most symbols, two attributes are of interest: the syntactic class (In-Fun, ...) assigned to it by the type-checker, and the kind of symbol LaTeX generates from it. The first of these affects the parsing of an expression containing the symbol, and the second affects the way spaces will be inserted when the expression is printed. In the description below, 'thin', 'medium' and 'thick' spaces are the same as those produced by the LaTeX commands `\,` and `\:` and `\;` respectively.

Here are the mnemonics for the basic elements of the Z language:

$\mathbb{P}$	`\power`	λ	`\lambda`
$\times$	`\cross`	μ	`\mu`
$\in$	`\in`	Δ	`\Delta`
$\mid$	`|` or `\mid`	Ξ	`\Xi`
$\bullet$	`@` or `\spot`	$\widehat{=}$	`\defs`
θ	`\theta`		

The operators of propositional logic and the schema calculus are as follows. Many of these names are already defined by LaTeX, but the spacing is often adjusted to make them look better in Z specifications.

$\neg$	`\lnot`	$\exists$	`\exists`
$\land$	`\land`	$\exists_1$	`\exists_1`
$\lor$	`\lor`	$\setminus$	`\hide`
$\Rightarrow$	`\implies`	$\upharpoonright$	`\project`
$\Leftrightarrow$	`\iff`	pre	`\pre`
$\forall$	`\forall`	$\fatsemi$	`\semi`

Here are the various sorts of fancy brackets:

$\{\ldots\}$	`\{ ... \}`	$\langle\!\langle\ldots\rangle\!\rangle$	`\ldata ... \rdata`
$\langle\ldots\rangle$	`\langle ... \rangle`	$\ldots(\!\vert\ldots\vert\!)$	`... \limg ... \rimg`
$[\![\ldots]\!]$	`\lbag ... \rbag`		

Those are all the symbols 'built-in' to the Z language; now for the symbols defined as part of the mathematical tool-kit. First come the symbols which are not defined as infix operators, etc.:

	`\empty`	$\mathbb{N}$	`\nat`
$\bigcup$	`\bigcup`	$\mathbb{Z}$	`\num`
$\bigcap$	`\bigcap`	$\mathbb{N}_1$	`\nat_1`
dom	`\dom`	$\#$	`\#`
ran	`\ran`	$\frown/$	`\dcat`

Here are the infix function symbols; they are defined in LaTeX as binary operators, so medium spaces are inserted automatically. The type-checker recognizes them as of class In-Fun. Each symbol is shown with its priority:

$\mapsto$	`\mapsto`	1	mod	`\mod`	4
$..$	`\upto`	2	$\cap$	`\cap`	4
$+$	`+`	3	$\comp$	`\comp`	4
$-$	`-`	3	$\circ$	`\circ`	4
$\cup$	`\cup`	3	$\upharpoonright$	`\filter`	4
$\setminus$	`\setminus`	3	$\oplus$	`\oplus`	5
$\frown$	`\cat`	3	$\lhd$	`\dres`	6
$\uplus$	`\uplus`	3	$\rhd$	`\rres`	6
$*$	`*`	4	$⩤$	`\ndres`	6
div	`\div`	4	$⩥$	`\nrres`	6

The postfix function symbols (class Post-Fun) all produce superscripts:

$^\sim$	`\inv`	*	`\star`
$^+$	`\plus`	^{n}	`\bsup n \esup`

As an example, `R \star` is printed as R^*. For iteration, the commands `\bsup ... \esup` should be used: for example, `R \bsup n \esup` is printed as R^n. The type-checker regards this formula as equivalent to *iter* $n\,R$, as explained on page 112 of the ZRM.

The infix relation symbols (class In-Rel) are defined in LaTeX as relations, so thick spaces are inserted around them automatically:

$\neq$	`\neq`	$>$	`>`
$\notin$	`\notin`	$\leq$	`\leq`
$\subseteq$	`\subseteq`	$\geq$	`\geq`
$\subset$	`\subset`	partition	`\partition`
$<$	`<`	in	`\inbag`

There is only one prefix relation symbol (class Pre-Rel). It separates itself from an argument with a thick space:

disjoint	`\disjoint`

The infix generic symbols are seen by LaTeX as relation symbols, so they are surrounded by thick spaces. Of course, the type-checker itself assigns them class In-Gen:

$\leftrightarrow$	`\rel`	$⤀$	`\psurj`
$⇸$	`\pfun`	$\twoheadrightarrow$	`\surj`
$\rightarrow$	`\fun`	$⤖$	`\bij`
$⤔$	`\pinj`	$⇻$	`\ffun`
$\rightarrowtail$	`\inj`	$⤕$	`\finj`

Prefix generic symbols are assigned class Pre-Gen by the type-checker; in LaTeX, they are defined as operator symbols, so that a thin space is inserted between the symbol and a following generic parameter:

$\mathbb{P}_1$	`\power_1`	seq	`\seq`
id	`\id`	seq_1	`\seq_1`
$\mathbb{F}$	`\finset`	iseq	`\iseq`
$\mathbb{F}_1$	`\finset_1`	bag	`\bag`

5 Fine points

In math mode, which is used for type-setting the contents of Z boxes, TEX ignores all space characters in the input file. The spaces which appear between elements of a mathematical formula are determined by TEX itself, working from information about the symbols in the formula. Although this information has been adjusted in the `zed` style option to make Z texts look as balanced as possible, there are one or two situations in which TEX needs a little help.

Special care is needed when function application is indicated by juxtaposing two identifiers, as in the expression *rev words*. This expression should be typed as `rev~words`. Typing just `rev words` results in the output *revwords*, since TEX ignores the space separating the two identifiers. In a formula, the character `~` inserts the same amount of space as the LATEX `\,` command, but it looks better in the input file. The type-checker completely ignores both the `~` character and the LATEX spacing commands, except that it issues a warning if it finds that a needed one is missing, for example, between two identifiers. It is not necessary to separate symbols like `\dom` and `\ran` from their arguments with a `~`, because TEX inserts the right amount of space automatically. For example, the input `\dom f` produces $\mathrm{dom}\, f$.

It is good style also to insert small spaces inside the braces of a set comprehension, as in this example:

```
\{~x: \nat | x \leq 10 @ x * x~\}
```

$$\{\, x : \mathbb{N} \mid x \leq 10 \bullet x * x \,\}$$

This helps to distinguish it visually from a set display, which should not have the space:

```
\{1, 2, 3\}
```

$$\{1, 2, 3\}$$

The space symbol `~` is ignored by the type-checker, so this is purely a matter of appearance. It also looks better if you add small spaces inside the square brackets of 'horizontal' schema texts.

TEX also needs help when a binary operator appears at the end of a line, as in the following example:

```
\begin{zed}
    directory' = directory \cup {} \\
\t3                 \{new\_name? \mapsto new\_number?\}
\end{zed}
```

$directory' = directory \cup$
$\quad\quad \{new\_name? \mapsto new\_number?\}$

TEX will not recognize `\cup` as a binary operator and insert the correct space unless it is surrounded by two operands, so the dummy operand `{}` has been inserted: this is ignored by the type-checker. This problem affects only binary operators; relation signs do not need to be surrounded by arguments to be recognized by TEX.

6 Bits and pieces

Specification documents often contain mathematical text which does not form part of the specification proper. This section describes some environments for setting various kinds of mathematics; they are provided for convenience, and they are all ignored by the type-checker. Besides these environments for making displays, run-in mathematics can be set with the usual `math` environment, or with the commands `$ ... $` or `\( ... \)`. All the Z symbols listed in Section **??** can be used with these commands.

The simplest display environment is provided by the commands `\[ ... \]`. This form acts just like `\begin{zed} ... \end{zed}`, except that the contents are ignored by the type-checker. Here is an example:

```
\[
    \exists PhoneDB @ \\
\t1     known = \empty
\]
```

$\exists PhoneDB \bullet$
$\quad known =$

These commands generalize the standard LATEX commands with the same name, because the displayed material can be several lines. Note, however, that the contents are set as text style rather than display style mathematics.

A schema box with no name is generated by the `schema*` environment:

```
\begin{schema*}
        x, y: \nat
\where
        x > y
\end{schema*}
```

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

This form is often useful for showing the result of expanding a complex schema-expression.

Another kind of mathematical display is provided by the `argue` environment. This is like the `zed` environment, but the separation between lines is increased a little, and page breaks may occur between lines. The intended use is for arguments like this:

```
\begin{argue}
    S \dres (T \dres R) \\
\t1     = \id S \comp \id T \comp R \\
\t1     = \id (S \cap T) \comp R & law about $\id$ \\
\t1     = (S \cap T) \dres R.
\end{argue}
```

$$\begin{aligned}
&S \lhd (T \lhd R) \\
&\quad = \mathrm{id}\, S \mathbin{;} \mathrm{id}\, T \mathbin{;} R \\
&\quad = \mathrm{id}(S \cap T) \mathbin{;} R && [\text{law about id}] \\
&\quad = (S \cap T) \lhd R.
\end{aligned}$$

When the left-hand side is long, I find this style better than the LaTeX `eqnarray` style, which wastes a lot of space. The second field on each line is optional. Again, the `argue` environment is ignored by the type-checker.

Finally, there is the `infrule` environment, used for inference rules:

```
\begin{infrule}
    \Gamma \shows P
\derive[x \notin freevars(\Gamma)]
    \Gamma \shows \forall x @ P
\end{infrule}
```

$$\frac{\Gamma \vdash P}{\Gamma \vdash \forall x \bullet P} \quad [\, x \notin \mathit{freevars}(\Gamma) \,]$$

The horizontal line is generated by `\derive`; the optional argument is a side-condition of the rule.

7 Style parameters

A few style parameters affect the way Z text is set out; they can be changed at any time if your taste doesn't match mine.

`\zedindent` The indentation for mathematical text. By default, this is the same as `\leftmargini`, the indentation used for list environments.

`\zedleftsep` The space between the vertical line on the left of schemas, etc., and the maths inside. The default is 1em.

`\zedtab` The unit of indentation used by `\t`. The default is 2em.

`\zedbar` The length of the horizontal bar in the middle of a schema. The default is 6em.

`\zedskip` The vertical space inserted by `\also`. By default, this is the same as that inserted by `\medskip`.

8 The fuzz package

The *f*UZZ package consists of two parts – a style option compatible with the `zed` style option described here, and an analysis and checking program. Using *f*UZZ together with LaTeX, you can:

- Input Z specifications as ordinary ASCII files.
- Process them for laser printing or photo-typesetting.
- Check them for conformance with the Z language rules.
- Produce a listing showing the schemas in the specification with components and their types.

The *f*UZZ analysis program works on the same ASCII file as LaTeX; it extracts the formal text and checks it for conformance with the rules of the Z language, producing clear error messages. Analysis of a 1300-line specification takes about 7 seconds on a SUN 3/75.

The *f*UZZ distribution contains the LaTeX style option, a special font of Z symbols, object code for the analysis program, a library containing the standard mathematical tool-kit, and some example specifications. To use *f*UZZ, you will need to have LaTeX installed on your machine, but everything else you need is included. *f*UZZ is currently available under 'no-nonsense' licence conditions for the IBM PC and other DOS machines, and for the SUN 3 and SUN 4 under SUN UNIX. The PC version can also be used on the PS/2. We are willing to produce versions for other machines according to demand.

Ordering information

You can order the *f*UZZ package either by cutting out the coupon below and sending it with your payment, or by sending an official order – we will send an invoice. Please send all orders to the address below. Technical enquiries can be sent to Mike Spivey at the same address, or by E-mail to `mike@uk.ac.oxford.prg` .

*f*UZZ **package: order form**

To: Mrs. A. Spivey, 34, Westlands Grove, Stockton Lane,
York, YO3 0EF, England.

Name: ______________________________

Address: ______________________________

____

____

Telephone: ______________________________

Please send [ ] copies of the *f*UZZ package for the following machines:

[ ]	SUN 3 version: Cartridge tape	£300
[ ]	SUN 4 version: 3.5in disk	£275
[ ]	SUN 4 version: Cartridge tape	£300
[ ]	IBM PC version: 5.25in disk	£200
[ ]	IBM PC version: 3.5in disk	£200

I enclose a cheque for £[ ], payable to Dr. J. M. Spivey.

Signed: