Preparing Z specifications using troff mark-up

/Tutorial guides

This tutorial won't teach you how to write Z, but if you already have some idea on that, then it will show you how to mark-up that Z so that cadiz will be able to process it. Specific examples are given here; for general definitions, see the reference manual.

1. Contents of this page

2. Delimiting formal paragraphs

The formal paragraphs of a Z specification are delimited from the informal paragraphs by being enclosed within troff-like requests. Those paragraphs that have an outline are delimited by `.ZS` and `.ZE`, with the middle line of a paragraph box marked by `.ZM`. These can be thought of as the start, middle and end requests. They must appear at the start of a line, with the `.` having no space after it.

```
.ZS
...
.ZM
...
.ZE
```

Those paragraphs that do not have an outline are delimited by `.ZH` and `.ZE`, where `.ZH` can be thought of as standing for horizontal, since non-outlined paragraphs are typically one liners.

```
.ZH
...
.ZE
```

The paragraph delimiters are illustrated in the example specification below.

3. Informal text

All the facilities of troff and its preprocessors may be exploited for typesetting informal material between Z paragraphs. However, formal and informal text must be separate: cadiz's parser will fail if any informal text appears in a formal paragraph, and troff will fail if a formal paragraph appears within a larger construct, such as a keep.

4. Directives to cadiz

cadiz ignores the informal text in a specification, recognising only the formal Z paragraphs and the directives that affect the tool's behaviour. These directives are all marked-up as individual lines, and look like troff requests starting with `.Z` and followed by a distinguishing letter and any arguments. For example,

```
.Zo sinc U+2286
```

directs cadiz to map all occurrences of `sinc` to the UCS character 2286.

Most directives must be written between formal paragraphs, not within them. There are directives for typesetting indexes, turning off typesetting, generating conjectures, and so on. A full list of directives is in the reference manual.

5. Typesetting troff mark-up

The troff typesetting program is run by cadiz on your behalf: your troff mark-up has to be processed by cadiz before troff can typeset it. This processing includes prettyprinting of the formal text. If your mark-up includes text to be seen by troff pre-processors, such as eqn and tbl, it should be passed through those pre-processors before being given to cadiz, as in the following example.

```
tbl filename.t | eqn >filename.z
cadiz filename.z
```

6. An example specification

The following example specification is intended to illustrate the troff mark-up that cadiz can recognise and the typesetting of formal text performed by cadiz. It is not intended to be indicative of good style but simply to provide examples of troff mark-up and typesetting. The specification is of a bicycle gearing system.

6.1. Section heading

section *gearsystem* parents *standard_toolkit*

```
.ZH
section gearsystem parents standard\_toolkit
.ZE
```

6.2. Free types paragraph

$$gearing ::= Single \mid Hub\langle\langle \mathbb{N} \rangle\rangle \mid Derailleurs\langle\langle \mathbb{N} \times \mathbb{N} \rangle\rangle$$

```
.ZH
gearing ::= Single | Hub << nn >> |
    Derailleurs << nn cp nn >>
.ZE
```

6.3. Horizontal definition paragraph

$$Wellgeared == [gears : gearing \mid gears \in ran\ Derailleurs]$$

```
.ZH
Wellgeared == [ gears : gearing | gears mem ran Derailleurs ]
.ZE
```

6.4. Generic axiomatic description paragraph

$$
\begin{array}{|l}
[X, Y] \\
\hline
First : X \times Y \to X \\
Second : X \times Y \to Y \\
\hline
\forall\, x : X;\ y : Y \bullet \\
\quad First\ (x, y) = x \land \\
\quad Second\ (x, y) = y
\end{array}
$$

```
.ZS [X,Y]
First : X cp Y fx X
Second : X cp Y fx Y
.ZM
forall x : X; y:Y dot
        First (x,y) = x and
        Second (x,y) = y
.ZE
```

6.5. Schema definition paragraph

$$
\begin{array}{|l}
\hline
\textit{Gear} \\
\textit{Wellgeared} \\
\textit{ingear} : \mathbb{N} \times \mathbb{N} \\
\hline
\exists\, r, s : \mathbb{N} \mid \textit{gears} = \textit{Derailleurs}(r, s) \bullet \\
\qquad \textit{First ingear} < r \land \textit{Second ingear} < s \\
\hline
\end{array}
$$

```
.ZS Gear
Wellgeared
ingear : nn cp nn
.ZM
exists r, s : nn | gears = Derailleurs (r,s) dot
       First ingear < r and Second ingear < s
.ZE
```

6.6. Generic schema definition paragraph

$$
\begin{array}{|l}
\hline
\textit{Bicycle}\ [G] \\
\textit{gears} : \mathbb{P}\, G \\
\hline
\end{array}
$$

```
.ZS Bicycle [G]
gears:ps G
.ZE
```

6.7. Axiomatic description paragraph

$$
\begin{array}{|l}
numgears : gearing \rightarrow \mathbb{N} \\
\hline
\forall\, g : gearing \bullet \\
\quad (g = Single \Rightarrow numgears\ g = 1) \land \\
\quad (\exists\, x : \mathbb{N} \bullet g = Hub\ x \Rightarrow numgears\ g = x) \land \\
\quad (\exists\, r, s : \mathbb{N} \bullet g = Derailleurs\ (r, s) \Rightarrow numgears\ g = r * s)
\end{array}
$$

```
.ZS
numgears : gearing fx nn
.ZM
forall g : gearing dot
        (g = Single imp numgears g = 1) and
        (exists x : nn dot g = Hub x imp numgears g = x) and
        (exists r, s :nn dot g = Derailleurs (r, s) imp numgears g = r*s)
.ZE
```

6.8. Conjecture paragraph

Lemma1 ==

$$\vdash ?\ numgears(Derailleurs(3, 7)) = 21$$

```
.ZH
```

```
thrm numgears (Derailleurs (3,7)) = 21
.ZE Lemma1
```

The argument to `.ZE` is the name of this conjecture. A conjecture need not have a name. There is a tutorial on proving conjectures using cadiz.

7. Mathematics within paragraphs

The mark-up of specific mathematical symbols may be found by referring to the following tables in the reference manual,

- standard core Z notation
- non-standard extensions
- toolkit symbols

Spaces, tabs and newlines in the mark-up are retained as *SPACE* characters in the UCS sequence. cadiz prettyprints all formal text, so extra spaces can be included to layout the mark-up neatly without affecting the typeset document.

It can be helpful to bear in mind that the troff mark-up is first converted to a sequence of UCS characters, and that it is that sequence which cadiz lexes and parses. When cadiz rejects your mark-up and you need to work out why, it is worth considering the sequence of UCS characters to which cadiz converts your

mark-up. The prep tool may be helpful, as it does the same translation to UCS characters as cadiz but then just displays them in a window.

Mathematical symbols may be referred to by their troff codenames, e.g. `\(te` is acceptable mark-up for ∃, though you might prefer to use the name `exists`. When you want to introduce a name for a new mathematical symbol, you will need to provide a mark-up directive to tell cadiz how to convert it to a sequence of UCS characters. Here are some examples of mark-up directives for symbols that are single UCS characters.

```
.Zo Delta U+0394
.Zo arithmos U-0001D538
.Zo rel U+2194
.Zo fss U-0001D53D
.Zo clseq U+3009
```

Notice that four-character encodings are used for symbols on the basic multilingual plane of UCS, and eight-character encodings are used for other symbols. A similar directive exists for multiple character symbols.

```
.Zw dcat cat/
.Zw rtc nearrow*swarrow
```

The conversion of each name includes *SPACE* before and after it, except when the name appears in another `.Zw` mark-up directive.

Subscripts and superscripts can be marked-up using the arrow characters named `searrow`, `nwarrow`, `nearrow` and `swarrow`. Single-character subscripts can alternatively be marked-up using `^c`.

The convention for operation schema names involving Greek Δ and Ξ characters can be marked-up using `DELTA` and `XI` prefixes as part of the word, as well as using `\(*D` and `\(*C`. For example, `DELTABirthdayBook` and `\(*DBirthdayBook`.

IT 22-Jan-2002