A Z Specification of a Z Preprocessor

/example Z specifications

1. Introduction

A specification written in standard Z [?] is comprised of a sequence of sections [?], each of which has a name and includes the paragraphs of other sections known as its parents. For compatibility with traditional Z [?], a sequence of paragraphs is accepted as comprising the sections of the mathematical toolkit and an anonymous section containing those paragraphs.

This document is a standard Z specification of a preprocessor for Z specifications. The preprocessor's job is to dispense with the non-standardised notion of file, in which sections are stored in a file system, and to permute sections into a definition before use order as assumed by the Z standard.

This specification assumes that a file contains a sequence of paragraphs: the preprocessor needs to distinguish formal and informal paragraphs, and to identify section headers, but it does not need to parse the Z text within formal paragraphs. There can be several sections within a single file, in which case it can be useful for the file to have several names (links in UNIX terminology).

Any references to parent sections that have not yet been read are presumed to be in files of the same name, and are read from there. Within a file, any formal paragraphs that are not preceded by a section header are treated as if

there had been a section header whose name is that of the file and which has *standard_toolkit* as parent. This is similar to the treatment of anonymous sections in the Z standard. A file's name need not be the same as any of the sections it contains, in which case that name is useless from the point of view of finding parent sections, but it is useful as a starting point for a whole specification.

2. Specification

This specification makes use of the standard mathematical toolkit.

section *preprocessor* parents *standard_toolkit*

2.1. Data types

Strings are encoded in ASCII (to be compatible with the treatment of string literal expressions by version 3.13 of the CADiℤ tool [**?**], which has checked this specification).

$$String ::= string\langle\langle seq(0 \mathinner{..} 127)\rangle\rangle$$

Names (of files and sections) are represented by strings. (The form of names would be irrelevant to this specification but for "*standard_toolkit*".)

$$Name ::= name\langle\langle String\rangle\rangle$$

Only certain kinds of paragraphs need be distinguished. Informal text between formal paragraphs is retained for possible display in the same order between the formal paragraphs. Section headers are treated like paragraphs in this specification.

$$\begin{aligned} Paragraph ::= & \ Informal\langle\langle String\rangle\rangle \\ | & \ Section\_header\langle\langle [name : Name;\ parent\_set : \mathbb{F}\ Name]\rangle\rangle \\ | & \ Formal\langle\langle String\rangle\rangle \end{aligned}$$

The file system is modelled as a function from pathnames (formed of directory and file names) to sequences of paragraphs. This avoids having to specify the parsing of files of text. Section headers can have been distinguished by the section keyword.

$$Directory == Name$$

$$File\_system ::= file\_system\langle\langle Directory \times Name \pfun seq\ Paragraph\rangle\rangle$$

Sections are represented as sequences of paragraphs in which an explicit section header begins each section.

$$\begin{aligned} & Section == \\ & \quad \{ps : seq_1\ Paragraph \\ & \qquad | \ head\ ps \in ran\ Section\_header \\ & \qquad \land\ ran(tail\ ps) \cap ran\ Section\_header = \varnothing\} \end{aligned}$$

2.2. Environment

This specification operates in an environment comprising: the file system *fs*; the current working directory name *cwd*; the name of the directory containing the toolkit sections *toolkit_dir*; and an environment variable *SECTIONPATH* giving the names of other directories from which sections may be read. The environment is modelled as the global state of the specification. Its value is not changed by the specification.

$$
\begin{array}{l}
fs : File\_system \\
cwd, toolkit\_dir : Directory \\
SECTIONPATH : seq\ Directory
\end{array}
$$

2.3. Functions

The function *section_to_name* is given a section and returns the name of that section. The name returned is that in the section header that is the section's first paragraph.

$$
\begin{array}{l}
section\_to\_name == \\
\quad \lambda\, s : Section \bullet ((Section\_header^{\sim})\ (head\ s)).name
\end{array}
$$

The function *sections_to_parents* is given a set of sections and returns the set containing the names of the parents referenced by those sections.

$$
\begin{array}{l}
sections\_to\_parents == \\
\quad \lambda\, ss : \mathbb{F}\, Section \bullet \\
\quad\quad \bigcup \{ s : ss \bullet ((Section\_header^{\sim})\ (head\ s)).parent\_set \}
\end{array}
$$

The function *filename_to_paras* is given a search path of directory names and a file name and returns the sequence of paragraphs contained in the first file found with that name in the path of directories to be searched. If no file with that name is found, an empty sequence of paragraphs is returned (and an error should be reported by an implementation).

$$
\begin{array}{l}
filename\_to\_paras : seq\ Directory \times Name \pfun seq\ Paragraph \\
\hline
\forall\, n : Name \bullet \\
\quad filename\_to\_paras\ (\langle\rangle, n) = \langle\rangle \\
\forall\, d : Directory;\ path : seq\ Directory;\ n : Name \bullet \\
\quad filename\_to\_paras\ (\langle d \rangle \frown path, n) = \\
\quad\quad \mathrm{if} (d, n) \in dom((file\_system^{\sim})\ fs) \\
\quad\quad \mathrm{then} (file\_system^{\sim})\ fs\ (d, n) \\
\quad\quad \mathrm{else}\ filename\_to\_paras\ (path, n)
\end{array}
$$

The function *filename_to_paragraphs* is given a filename and returns the sequence of paragraphs contained in the first file found with that name in the path of directories to be searched. The current working directory is always searched first, then whatever directories are explicitly listed in the SECTIONPATH environment variable, and finally the directory of toolkits.

$$
\begin{array}{l}
filename\_to\_paragraphs == \\
\quad \lambda\, n : Name \bullet \\
\quad\quad filename\_to\_paras\ (\langle cwd \rangle \frown SECTIONPATH \frown \langle toolkit\_dir \rangle, n)
\end{array}
$$

The function *add_header* reads the named file and prefixes its sequence of paragraphs with a section header if the file starts with an anonymous section. If the anonymous section has any formal paragraphs, it is named after the file, otherwise it is given a different name in case the first named section has that name.

$$
\begin{array}{l}
add\_header == \\
\quad \lambda\, n : Name \bullet \\
\quad\quad \mathrm{let}\ ps == filename\_to\_paragraphs\ n \bullet \\
\quad\quad\quad (\mu\, pref, suff : seq\ Paragraph \mid pref \frown suff = ps \land \\
\quad\quad\quad\quad ran\ pref \cap ran\ Section\_header = \varnothing \land \\
\quad\quad\quad\quad (suff = \varnothing \lor head\ suff \in ran\ Section\_header) \bullet \\
\quad\quad\quad\quad\quad \mathrm{if}\ pref = \varnothing\ \mathrm{then} \langle\rangle \\
\quad\quad\quad\quad\quad \mathrm{else\ if}\ ran\ pref \cap ran\ Formal \neq \varnothing\ \mathrm{then} \\
\quad\quad\quad\quad\quad\quad \langle Section\_header \langle\!|\ name == n, \\
\quad\quad\quad\quad\quad\quad parent\_set == \{name\ (string\ ”standard\_toolkit”)\} \\
\quad\quad\quad\quad\quad \mathrm{else} \\
\quad\quad\quad\quad\quad\quad \langle Section\_header \langle\!|\ name == \\
\quad\quad\quad\quad\quad\quad\quad name(string((string^{\sim})\ ((name^{\sim})\ n) \frown ”informa \\
\quad\quad\quad\quad\quad\quad parent\_set == \{\}\ |\!\rangle\rangle \\
\quad\quad\quad\quad\quad \frown ps
\end{array}
$$

The function *filename_to_sections* reads the named file and partitions its sequence

of paragraphs into a sequence of sections.

$$
\begin{array}{l}
filename\_to\_sections == \\
\quad \lambda\, n : Name \bullet \\
\quad\quad (\mu\, ss : seq\, Section \mid \frown\!/\, ss = add\_header\ n)
\end{array}
$$

The function *read_spec* is given a set of names of files to be read and a set of sections already read from files. It returns the set of sections containing those already read, those read from the named files, and those read from files named as ancestors of other sections in this set. A file is read only if the named parent has not already been found in previous files and is not present anywhere in the current file; the parent section could be defined later in the current file, in which case any file with the name of the parent is not read. The sections should all have different names (otherwise an implementation should report an error); this specification merges sections that are identical.

$$
\begin{array}{l}
read\_spec : \mathbb{F}\, Name \times \mathbb{F}\, Section \to \mathbb{F}\, Section \\
\hline
\forall\, ss : \mathbb{F}\, Section \bullet \\
\quad read\_spec\ (\varnothing, ss) = ss \\
\forall\, ns : \mathbb{F}\, Name;\ ss : \mathbb{F}\, Section \bullet \\
\quad read\_spec\ (ns, ss) = \\
\quad\quad \mu\, ss_2 == \bigcup \{ n : ns \bullet ran(filename\_to\_sections\ n) \} \\
\quad\quad\quad \mid \# ss_2 = \#(section\_to\_name(\!|\ ss_2\ |\!)) \bullet \\
\quad\quad\quad\quad read\_spec\ (sections\_to\_parents\ ss_2 \backslash \\
\quad\quad\quad\quad\quad\quad section\_to\_name(\!|\ ss\ |\!), \\
\quad\quad\quad\quad\quad ss \cup ss_2)
\end{array}
$$

The function *order_sections* is given a set of sections and returns those sections in a sequence ordered so that every section appears before it is referenced as a parent. The function is partial because of the possibility of cycles in the parents relation, about which an implementation should report errors.

$$
\begin{array}{l}
order\_sections == \\
\quad \{ss : \mathbb{F}\, Section;\ ss_2 : seq\ Section \\
\qquad \mid ran\ ss_2 = ss \\
\qquad \land\ (\forall\, ss_3 : seq\ Section \mid ss_3\ prefix\ ss_2 \bullet \\
\qquad\quad \{section\_to\_name(last\ ss_3)\} \cap \\
\qquad\quad sections\_to\_parents\ (ran(front\ ss_3)) = \varnothing) \\
\qquad \bullet\ (ss, ss_2)\}
\end{array}
$$

The function *preprocessor* specifies the entire tool. It takes the name of a file, and returns the ordered sequence of sections from that file and the files of ancestral sections.

$$
\begin{array}{l}
preprocessor == \\
\quad \lambda\, n : Name \bullet order\_sections\ (read\_spec\ (\{n\}, \{\}))
\end{array}
$$

3. Further work

1. The consistency of this specification has not been formally proven.
2. A task that is best done by the preprocessor, but has not been specified here, is the extraction from operator template paragraphs of a mapping

from words to tokens for each section, to be consumed by the Z parser, as implicitly required by the Z standard.

The preprocessor has been implemented and is in use within CADiZ. Some small bugs were found in this spec—did you spot them? Cadiz reads any cumulus (-l) file first, and passes the names of the sections so obtained to zpp, which omits them from its output. The preprocessor permutes operator template paragraphs to the beginnings of their sections, so that operators can be used before being introduced in the typeset presentation. (This fails to allow an operator word to be used in multiple operators.) It also moves all glyph directives to before the operator templates. Cadiz checks the paragraphs in the resulting order, but permutes them back into the original order for typesetting; file and line directives are inserted by the preprocessor to enable this. Any file or line directives in the preprocessor's input are ignored. Quiet and reckless directives are recognised, and these modes are recorded as attributes of paragraphs, so that the mode can be set appropriately after permutation.

Acknowledgements

Sam Valentine advised on the use of Z in this specification.