`mweb`: proof script management utilities

W. Wong

University of Cambridge, Computer Laboratory
New Museums Site, Pembroke Street
Cambridge, CB2 3QG, England.

February 1994

This document describes version 3.0 of `mweb`.

Contents

Acknowledgement

Thanks to B. T. Graham who spotted several bugs in earlier version of the programs and extended the `mweb` suite and wrote the manual section for the extension, and to P. Curzon who proof-read this manual.

1 A tutorial introduction

1.1 What is `mweb`?

`mweb` is a suite of utility programs for documenting and managing HOL proof scripts. It is inspired by the idea of *literate programming* of D. Knuth[1]. The core of this idea is to treat programs as *works of literature*. When programming in this method, one creates a master file containing both the document and the program text. This master file is processed by utility programs to generate a program file and a document file separately. The program file is then processed by a compiler or interpreter whilst the document file is processed by a text formatter to produce a human readable document. The programs developed by Knuth which embodied this idea are known as the `WEB` system. It has been used to develop one of the most popular text formatters TEX.[1] The main advantage of keeping both the program source and the document in a single file is that it is much easier to keep them both up-to-date in respect to each other and to produce a more accurate and significantly better document.

`mweb` is a suite of such utility programs principally for developing proof scripts in HOL, and essentially for programming in ML. It is simple to use. It has a number of special features, such as multiple languages, including tag files, special operation modes and user-definable parameter values. Probably the most important feature of `mweb` is its highly programmable input syntax and output format. This allows `mweb` to be easily customised to work with different specification languages, programming languages and text formatters, ant to suit different applications, requirements and individual styles. Section 6 describes such an extension to the basic `mweb` suite.

The basic `mweb` utility consists of two programs: `mtangle` and `mweave`. `mtangle` processes the master file to generate an ML file which can then be loaded into HOL to perform the proof. `mweave` processes the master file to generate a LATEX file which can then be formatted using LATEX to produce a document. `mweave` also accepts extra files, known as the include files or *tag* files. These files usually contains the results from running the script through HOL. This process is illustrated in Figure 1. Both utilities share a common core, thus parameters, command line options, etc. described in Section 2–5 apply to both.

1.2 How to use `mweb`?

In the following tutorial example, we assume that the text formatter is LATEX and the proof script is in HOL88, i.e., in classic ML. The procedures can easily be generalised to use other formatter and programming languages. The basic

[1] The whole program of TEX has been published as a book [2] which contains the program text and detail documentation in one volume.

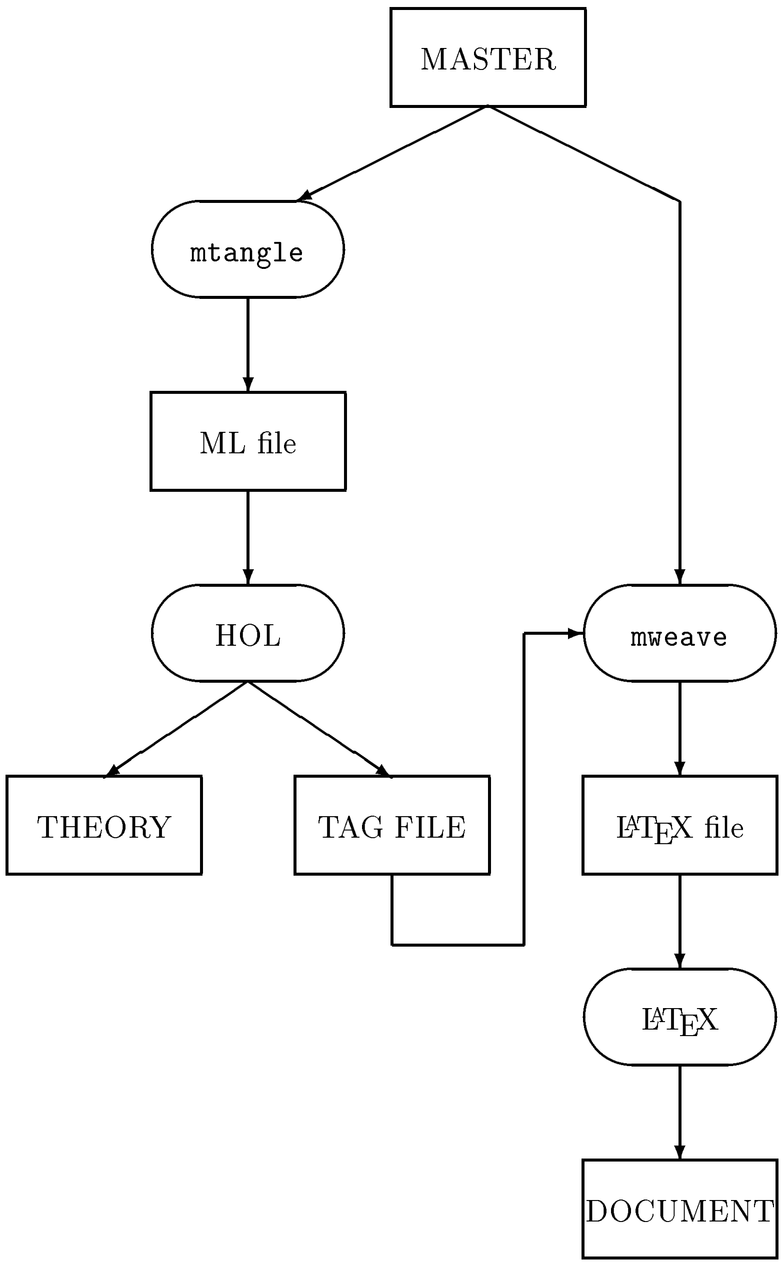

Figure 1: Operation of `mweb`

procedures of developing and managing proof script using `mweb` are:

1. preparing a master file, say `foo.m`;
2. running `mtangle` to generate an ML file `foo.ml`;
3. loading `foo.ml` into HOL to perform the proof;
4. preparing a tag file using the result from HOL if there is any tag command in the master file;
5. running `mweave` to generate a LaTeX file `foo.tex`;
6. running LaTeX to produce a document.

These procedures can be managed automatically using a `Makefile` with the utility `make`. The remainder of this section describes a simple example step by step. Suppose we have a master file called `lmem.m` which contains the following lines.

```
@D "mac_before_tag" = "\\begin{equation}\n"
@D "mac_after_tag" = "\\end{equation}\n"
@D "mac_begin_env" = "\\begin{comment}"
@D "mac_changed" = " \n"
@D "mac_end_env" = "\\end{comment}\n"
@D "ml_only" = "true"

@*{The theory {\tt lmem}}
This theory contains definition of the constant \CONST{LMEM} and a theorem.
@-
@M
@N
new_theory`lmem`;;
load_library`latex-hol`;;
@-

@ {Definition of \protect\CONST{LMEM}}
The constant \CONST{LMEM} is a predicate asserting the membership of an
element in a list.
@t{LMEM_DEF}
@N
let LMEM_DEF = new_list_rec_definition(`LMEM_DEF`,
 "(LMEM x [] = F) /\
  (LMEM x (CONS (h:*) t) = (x = h) \/ (LMEM x t))");;
@-

@ {A theorem}
The theorem \mlname{NULL_NOT_LMEM} states that nothing is an element
of an empty list.
@t{NULL_NOT_LMEM}
```

```
32 @N
33 let NULL_NOT_LMEM = prove_thm(`NULL_NOT_LMEM`,
34     "!(x:*) l. NULL l ==> ~(LMEM x l)",
35     GEN_TAC THEN LIST_INDUCT_TAC THEN REWRITE_TAC[NULL;LMEM_DEF]);;
36 @-
37
38 @M
39 @N
40 latex_thm_tag := `@t `;;
41 latex_thm_end := `@e `;;
42 latex_all_theorems_to (`lmem.tag`)
43  (\s. (theorem `-` s) ? (definition `-` s))
44  [`LMEM_DEF`; `NULL_NOT_LMEM`];;
45 @-
46
47 @M
48 @N
49 close_theory();;
50 @-
```

This file creates a theory named `lmem`. A constant definition defines a HOL constant LMEM. A theorem about this constant is proved and stored in the theory. In the master file listed above, there are a number of `mweb` commands which tell the utility programs how to process the file. Commands start with a `@` character at the beginning of a line. This is followed by a command character. A master file is divided into sections. For example, the command `@*` in line 9 begins a major section. In addition to the `@*` command, there are three other commands which begin a section. Two of these has been used in this file. They are `@␣` which begins an ordinary section as in lines 18 and 28, and `@M` which begins an ML-only section as in lines 12, 38 and 47. The command `@-` terminates a section. There are six sections in this master file.

Each section consists of three parts: the document part, the other-language part and the native-language part. All parts are optional. The document part contains text lines which describe the programs that appear in the other two parts. These lines are simply copied to the LaTeX file. Any LaTeX macros can appear in this part. For example, lines 19–21 are the documentation for the constant definition in that section. No other-language part has been used in this file, so its usage will be explained in a later section. The native-language part contains HOL text or ML programs. It starts with the command `@N` as in lines 13, 22, 32, 39 and 48. Lines in this part will be copied to the ML file, and they may also appear in the LaTeX file. Whether lines in this part appear in the LaTeX file and how they appear in the final document can be controlled by the user using various means. For simplicity, this master file has been processed in a way that no program lines appear in the final document. Any HOL and ML commands can appear in the native-language part. The ML file, `lmem.ml`, generated by `mtangle` from this master file is listed below.

```
%- 5.2 -%
new_theory`lmem`;;
load_library`latex-hol`;;

%- 16.3 {LMEM\_DEF} -%
let LMEM_DEF = new_list_rec_definition(`LMEM_DEF`,
 "(LMEM x [] = F) /\
  (LMEM x (CONS (h:*) t) = (x = h) \/ (LMEM x t))");;

%- 28.4 {NULL\_NOT\_LMEM} -%
let NULL_NOT_LMEM = prove_thm(`NULL_NOT_LMEM`,
    "!(x:*) l. NULL l ==> ~(LMEM x l)",
    GEN_TAC THEN LIST_INDUCT_TAC THEN REWRITE_TAC[NULL;LMEM_DEF]);;

%- 35.5 -%
latex_thm_tag := `@t `;;
latex_thm_end := `@e `;;
latex_all_theorems_to (`lmem.tag`)
 (\s. (theorem `-` s) ? (definition `-` s))
 [`LMEM_DEF`; `NULL_NOT_LMEM`];;

%- 44.6 -%
close_theory();;
```

As can be seen in the above listing, the ML program from each section in the master file is preceded by a comment line. It contains the line number, the section number which is automatically generated by `mtangle` and the name of the section if it is present. This helps to trace the source of problems if failure occurs when the file is loaded into HOL. However, one usually works directly with the master file while developing a proof in an interactive HOL session.

There are still a couple of commands in the master file that have not been explained. First is the command `@D` which appeared in lines 2 to 7. This is the parameter definition command. `mweb` has been developed in a way in which the actual input command and output format can be programmed by the user. This is done by redefining the values of a set of parameters using the `@D` command. The precise meanings of the parameters will be described in Section 5. Another command appearing in lines 21 and 31 in the master file is `@t`, known as the tag command. The string enclosed in a pair of braces following the tag command is the name of the tag. This command instructs `mweave` to find a text block in a tag file which has the same name, and to output this text block in place of the tag command to the LaTeX file. These text blocks usually contain theorems generated by HOL. In fact, any text and macros understood by LaTeX can appear in these blocks. In the second last section of the master file, the function `latex_all_theorems_to` in the `latex-hol` library is called to generate the tag file which is listed below.

```
@t LMEM_DEF
\THM (\FORALL x\DOT \CONST{LMEM} \,x \,\NIL  = \CONST{F}) \AND
     (\FORALL x \,h \,t\DOT \CONST{LMEM} \,x \,(\CONST{CONS} \,h \,t) =
                                (x = h) \OR  \CONST{LMEM} \,x \,t)
@e
@t NULL_NOT_LMEM
\THM \FORALL x \,l\DOT \CONST{NULL} \,l \IMP  \NOT \CONST{LMEM} \,x \,l
@e
```

This file contains two tagged blocks. Each of them is a theorem in LaTeX format. They will be output to the LaTeX file in place of the tag commands in Line 21 and 31 when the following system command is executed:

```
mweave -Ilmem.tag lmem.m lmem.tex
```

This command produces a LaTeX file named `lmem.tex`. In a simple application which consists of only a single master, all LaTeX preamble and post-amble materials can be put into the master file. More often, applications consists of more than a single master file. In this case, the preamble and post-amble materials can be put into a main LaTeX file which, in turn, includes all the files produced by `mweave`. The result of formatting the file `lmem.tex` is shown in the box below.

1.3 The theory `lmem`

This theory contains definition of the constant LMEM and a theorem.

1.3.1 Definition of LMEM

The constant LMEM is a predicate asserting the membership of an element in a list.

$$\vdash (\forall x.\, \mathsf{LMEM}\, x\, [\,] = \mathsf{F}) \wedge (\forall x\, h\, t.\, \mathsf{LMEM}\, x\, (\mathsf{CONS}\, h\, t) = (x = h) \vee \mathsf{LMEM}\, x\, t) \quad (1)$$

1.3.2 A theorem

The theorem `NULL_NOT_LMEM` states that nothing is an element of an empty list.

$$\vdash \forall x\, l.\, \mathsf{NULL}\, l \supset \neg \mathsf{LMEM}\, x\, l \quad (2)$$

Note that `mweave` assumes a set of LaTeX macros for formatting the document. A set of default definitions of macros is in the file `mwecmac.tex` which should be included in the LaTeX file explicitly.

2 The command line

The command given to the operating system to invoke the `mweb` programs takes the following form:

progname [*options*] [*infile*] [*outfile*]

where *progname* is the program name, e.g., `mtangle` or `mweave`. *infile* is the input file name. If this name has no suffix, i.e., the dot character `.` does not occur in the name, the current value of the parameter `insuffix` is appended to *infile*. Similarly, *outfile* is the name of the output file. If it has no suffix, the current value of the `outsuffix` parameter of the program is appended. For example, the `outsuffix` parameter of `mweave` is `outsuffixmweave`. The default input file suffix is `.m`. The default `mtangle` output file suffix is `.ml`, and the default `mweave` output file suffix is `.tex`.

Both the input and output file names can be omitted. In such a case, the program reads from the standard input and writes to the standard output. If only one file name is given, it is taken as the name of the input file. The name of the output file is derived from this by replacing (if it has suffix) or appending the suffix. If two file names are given, no suffix checking is carried out. They are taken literally as the input and output file names.

Options which may change the behaviour of the program can be specified when invoking the `mweb` programs. For example, the command

```
mweave -f filename
```

will cause `mweave` to run in filter mode. All options are single character and begin with a hyphen(-). Some of the options can take an argument. All options are listed below.

`-h -H` — help: print this message

`-v` — verbose: print processing information (default)

`-s` — silent: do not print processing information

`-L` ***lang*** — specify a language for `mtangle`

`-I` ***file*** — specify an include file

`-D` ***para=value*** — set the parameter *para* to *value*

`-m` — activate ML only mode

`-f` — activate filter mode

`-d` — activate document only mode

`-r` — activate raw text mode

`-p` — deactivate parallel mode

`-l` — activate line mode

The exact behaviour of the different operation modes will be described in Section 3.3

3 Operation of `mweb`

Since `mweb` is highly programmable, the actual input commands and output strings can be changed very easily. In the example described in the previous section, the concrete strings of the default settings have been shown. In the following description, generic command names and parameter names will be used instead of the concrete strings. For instance, the default command for starting a section is the string `@␣`, instead the command begin_section will be used in the following description. Command names will be typeset in a sans serif font, and parameter names will typeset in a `typewriter` font.

3.1 Master file format

The master file is divided into a number of *sections*. Each section is divided into three parts: the *document* part, the *other-language* part and the *native-language* part. They should appear in the order as illustrated in Figure 2. However, all the parts are optional. For instance, a section may contain only the document part or the document part and the native-language part.

The document part contains text describing the HOL proof script or specification in the other two parts. It will be formatted by the text formatter, i.e., LaTeX. All text formatter commands can appear in this part providing the first character of any line is not a command escape character (by default `@`). The native-language part contains program text which will be output to a file which will eventually be executed. HOL or ML is the default native language. The purpose of the other-language part is to allow specifications written in other languages to be included in the same document so that it can be compared to the native language program. For instance, a circuit specification in a hardware description language can be included in the other-language part. The translation of this specification into HOL is written in the native-language part. The document part describes the circuit and any differences in syntax or semantics in the two. These two specifications will appear in the final document side-by-side. A human reader will be able to compare them very easily.

A section starts with one of the four section begin commands: begin_section, begin_star_section, begin_ml_section and begin_anon_section. The other-language

```
...
begin_section optional section name
    ...
    document part
    ...
other_part
    ...
    other-language part
    ...
native_part
    ...
    native-language part
    ...
end_section
    ... more sections
```

Figure 2: Master file format

part starts with the other_part command, and similarly the native-language part starts with the native_part command. The command end_section terminates a section.

The intention of providing four different kinds of sections is to allow them to be processed in different ways. The sections started by the commands begin_section and begin_star_section are ordinary sections. The difference between them is that they are output into the document as two levels of sectioning commands. The purpose of the ML sections, i.e., started by the command begin_ml_section, is to hide the contents of the whole section. For example, some libraries have to be loaded to set up the operation environment in HOL before the real work begins. The ML code for this can be put in an ML section, and it will not appear in the final document. The anonymous section stops the section heading from being output to the document file. Thus, one can divided a logical section that will appear in the final document into several smaller sections. All these sections except the first one become anonymous sections. No extra section headings will be printed.

3.2 Command format

All commands are on separate lines called *command lines*. A command can be divided into three parts. The first is the command escape character which must be the first character of a command line. It marks the beginning of a command. The command escape character is usually a single character, but it can be programmed to be a multi-character string.

The second part is the command string. It must follow the command escape character immediately without leaving any intervening spaces. The actual string representing each command is the current value of the parameter associated with that command. For instance, the parameter associated with the begin_section command is char_begin_sec whose default value is a single space character ␣. In fact, the default strings of all commands are single characters.

The last part of a command is the argument. Some commands have mandatory arguments, such as the incl_file command which requires an argument to specify the name of the include file. Other commands have optional arguments, such as the begin_section command whose argument is the name of the section. If the argument is absent, the section will have no name. Only the section number will appear in the document. Still, other commands require no argument at all, such as the native_part command.

The command argument, if it is present must follow immediately the command string without leaving any intervening spaces. It must be enclosed by a pair of braces({}). In fact, the braces are just the default values of the parameters char_begin_para and char_end_para. They can be changed to other characters, but they must be a single character. Using the default settings, a concrete example of a command starting a section whose name is 'First section' is

```
@ {First section}
```

3.3 Operation modes

In ordinary operation, text before the first section, after the last section and in between sections is ignored by the programs. Only the command lines and text lines enclosed by the begin_section and end_section commands are significant.

The mtangle program copies to the output file all the text lines in the native-language part. For the purpose of debugging, a comment line in the native language precedes the text of each part in the output file. It contains the line number and the section number. If a name is given to a section, it also appears in this comment line. The comment line starts with the current value of the parameter ml_comm_begin and ends with the current value of the parameter ml_comm_end. The default strings of these parameters are %- and -%, respectively. Figure 3 shows the general form of input and the corresponding output written by mtangle.

The mweave program copies to the output file all the text lines in the document part. If a begin_tag command occurs in this part, instead of copying the command, mweave will output the text block whose name is the same as the name specified in the command. This block of text is preceded by the string which is

input	**mtangle** *output*
begin_section *section name*	***ml_comm_begin*** *line No. section No.*
document lines	*section name* ***ml_comm_end***
other_part	
otherline 1	
otherline 2	
...	
native_part	
nativeline 1	*nativeline 1*
nativeline 2	*nativeline 2*
...	...
end_section	

Figure 3: Input format and corresponding output from `mtangle`

the current value of the parameter `mac_before_tag` and terminated by the string which is the current value of the parameter `mac_after_tag`. Tagged text blocks are read by `mweave` from the include files, and saved in an internal buffer. If such a text block cannot be found, `mweave` issues a warning message and carries on.

By default, `mweave` outputs to the output file all the lines in the language parts in a side-by-side formatting environment. The default environment is called `paralines`. When entering an other-language part, `mweave` saves all lines in an internal buffer until it finds the native-language part. It then saves all lines in the native-language part in another internal buffer until the end of the section. When it sees the end_section command, it processes the two lists of saved lines and outputs the 'begin environment' string which is the current value of the parameter `mac_begin_env`. It then outputs the corresponding lines from the two parts, i.e. Line 1 of both parts and then Line 2 of both parts, etc., as the arguments to a line macro which is the current value of the parameter `mac_line`. This is terminated by the 'end environment' string. For instance, the default line macro is the string `\pline`. The output of a section looks like

```
\begin{paralines}
\pline{otherline 1}{nativeline 1}
\pline{otherline 2}{nativeline 2}
...
\pline{otherline m}{nativeline m}
\pline{otherline m+1}{}
```

input	*mweave output*
begin_section *section name*	*mac_begin_sec* *section name*
	mac_begin_doc
document lines	*document lines*
begin_tag	*mac_before_tag*
	tagged text block
	mac_after_tag
more lines	*more lines*
	mac_end_doc
other_part	*mac_begin_env* *mac_changed*
otherline 1	*mac_line*
otherline 2	*mac_begin_arg* *otherline 1* *mac_end_arg*
...	*mac_begin_arg* *nativeline 1* *mac_end_arg*
native_part	
nativeline 1	*mac_line*
nativeline 2	*mac_begin_arg* *otherline 2* *mac_end_arg*
...	*mac_begin_arg* *nativeline 2* *mac_end_arg*
...	...
	mac_end_env
end_section	*mac_end_sec*

Figure 4: Input format and corresponding output from `mweave`

```
...
\pline{otherline n}{}
\end{paralines}
```

If any part is missing or has fewer lines, the corresponding argument to the line macro is simply empty. The braces enclosing the lines are also programmable. They are the current value of the parameters `mac_begin_arg` and `mac_end_arg`. Figure 4 shows the general form of input and the corresponding output from `mweave`.

Text in the default `paralines` environment will be printed in two columns. Each column contains text from a language part. Different fonts can be used to typeset them to provide visual distinction between the two languages. A LaTeX macro file, named `mwebmac.tex`, containing the definition of this parallel environment is distributed with the package. This should be explicitly input into the LaTeX file.

There are a number of special operation modes which can be activated by the user to perform special tasks. By default, all modes are in their inactive states except parallel mode which is active. The operation modes are:

document only mode — If this mode is activated, `mweave` will only output the text in the document part to the LaTeX file. Thus, one can omit the program text from the document.

filter mode — If this mode is activated, `mweave` will copy all lines outside any sections to the output file.

ML only mode — If this mode is activated, `mweave` will ignore all sections which start with the command begin_ml_section. They will not appear in the final document at all. However, `mtangle` is not affected by this mode, so lines in the native language part of these sections will still be output to the ML file.

raw text mode — If this mode is activated, the program lines in both the other-language part and the native-language part will be copied to the output file as they are. Otherwise, the program lines are processed. Special characters are converted to their ASCII code in hexadecimal notation and prefixed with the string which is the current value of the parameter `mac_spec_char`. Special characters are defined as one of the characters in the string which is the current value of the parameter `spec_chars`. For example, with the default setting, the character `#` is a special character which is converted to `\char"23` since its ASCII code is 23 in hexadecimal.

parallel mode — This is the default operation mode of `mweave`. If it is deactivated, the programs in the other-language part and in the native-language part will output as two separate blocks of text. The lines from the other-language part will be output first, and they are followed by the lines from the native-language part. The lines from the other-language part are enclosed by the strings which are the current values of the parameters `mac_begin_other` and `mac_end_other`. Similarly, the lines from the native-language part are enclosed by the strings which are the current values of the parameters `mac_begin_native` and `mac_end_native`.

line mode — This operation mode affects `mtangle` only. When this mode is active, the program lines in the output file will have the same line number as in the master file. The lines of document part and other language part will become blank lines in the output file. This is useful in debugging since many compilers will report errors with a line number. By keeping the same line number in the program file, one can easily find and fix the errors in the master file.

All these modes can be activated either by using command line options or by changing the value of their associated parameter. If using command line options, the modes are active from the beginning of the file. If changing the parameter values, filter mode will become active from the next input line. The other modes will become active from the next section. All modes can be deactivated by changing the parameter value. The new value takes effect in the same way as described above.

3.4 Multiple language parts

Although two different languages are used in many applications in formal verification, sometimes more than two languages or more than two specifications in the same language are needed. To handle such cases, the language parts can be generalised to allow any number of different language parts in a section. A new command language_part can be used instead of the other_part and native_part commands. This command takes the name of the language as an argument. In addition to the standard command argument syntax described in Section 3.2, a shorthand format is also allowed in this command. This takes the form of a string immediately following the command character with no braces. It is terminated by the first blank character. For example, given the default command character of language_part is L, the following two command lines have the same meaning which begins the language part whose name is `top-level`:

```
@L {top-level}
@Ltop-level
```

A command line option for `mtangle` allows the user to specify which language to output to the program file. The default action of `mtangle` is to output the lines in the language part whose name is the same as that specified at the command line option. If no language option is specified, the native-part is output. Note that no checking is performed on the language names so be sure to spell them precisely in every section and in the command line option, otherwise, lines in the misspelled part will be missing from the output file.

The lines in the language parts are output to the LaTeX file by `mweave` if the *document only* mode is not active. Each language part is enclosed in an environment whose name is the name of the language as illustrated below:

mac_begin_lang mac_begin_arg *⟨name⟩* ***mac_end_arg***
lines in the language part whose name is ⟨name⟩
mac_end_lang mac_begin_arg *⟨name⟩* ***mac_end_arg***

Within a section, both the generalized language parts and the other- and native-parts can be occur. The generalised language parts may be place either before or after the other- and native parts. However, the use of only generalised language parts are recommended if side-by-side printing of two language parts is not required. Note also that `mtangle` is unable to output lines in the other-language part.

4 Commands

The utility programs have been implemented in a way that the actual command strings and the exact output strings can be customised by changing the values of a set of parameters by the user. The exact meanings of the commands and their associate parameters are described in this section.

NAME:	begin_anon_section	**ARG:**	Optional
PARMETER:	`char_anon_sec`	**DEFAULT:**	`@A`

This command begins an anonymous section. The actions for this command is the same as the begin_section command except that `mweave` does not output the section macro and the section name to the LaTeX file.

NAME:	begin_ml_section	**ARG:**	Optional
PARMETER:	`char_ml_sec`	**DEFAULT:**	`@M`

This command begins an 'ML only' section.

`mtangle` ACTION: save the section name if it is present.

`mweave` ACTION: same as begin_section command if the value of the parameter `ml_only` is false, otherwise, nothing is generated in the output file for the whole section.

NAME:	begin_section	**ARG:**	Optional
PARMETER:	`char_begin_sec`	**DEFAULT:**	`@␣`

This command begins an ordinary section.

`mtangle` ACTION: save the section name if it is present.

`mweave` ACTION: output section macro `mac_sec` and the section name as the argument to the macro if a name is present, otherwise, output a default section macro `mac_sec_default`. Then, copy all input lines to the output until one of the following commands is found: end_sec, other_part, native_part.

NAME: begin_star_section **ARG:** Mandatory
PARMETER: char_begin_star_sec **DEFAULT:** @*

This command begins a major section.

mtangle ACTION: save the section name.

mweave ACTION: output major section macro mac_star_sec and the section name as the argument to the macro. Then, copy all input lines to the output until one of the following commands is found: end_sec, other_part, native_part.

NAME: begin_tag **ARG:** Mandatory
PARMETER: char_begin_tag **DEFAULT:** @t

In a master file, this command marks the position a tagged text block should be placed. The argument to this command is the name of the block. There must be a block having the same name appear in a tag file. This file is included by mweave when processing the master file. In a tag file, this command begins a text block. All input lines are saved until an end_tag command is found.

mtangle ACTION: do nothing.

mweave ACTION: output the text block whose name is the same as the argument. If no block of the same name can be found, a warning message is printed. The block is enclosed by the parameters mac_before_tag and mac_after_tag.

NAME: end_section **ARG:** None
PARMETER: char_end_sec **DEFAULT:** @-

This command terminates a section.

mtangle ACTION: do nothing.

mweave ACTION: output the section. The text line saved in the internal buffers (if any) is processed and output in an environment enclosed by the parameters mac_begin_env and mac_end_env.

NAME: end_tag **ARG:** Optional
PARMETER: char_end_tag **DEFAULT:** @e

This command marks the end of a tagged text block in a tag file. If the optional argument is present it should be the same as the argument to the associated begin_block command.

NAME:	incl_file	**ARG:**	Mandatory
PARMETER:	`char_incl_file`	**DEFAULT:**	`@I`

This command inputs an include file. The argument specifies the name of a file to be read. It can only appear outside any section. The file is expected to contain only parameter definitions and tagged text blocks.

NAME:	language_part	**ARG:**	Mandatory
PARMETER:	`char_lang`	**DEFAULT:**	`@L`

This command begins a programming language part. It can only occur after one of the 'begin section' commands. It takes the name of the language as its argument. Two formats are allowed for this argument. The first is a string enclosed in a pair of braces which are the values of the parameters `char_begin_para` and `char_end_para` or a pair of quotes which are the value of the parameter `char_str_quote`. Blank space between the command and its argument in this form is optional. The second format is a string immediately after the command. It is ended by the first blank character.

`mtangle` ACTION: if the language name is equal to the language specified in the command line option, output a comment line, then output subsequent input lines until an end_sec, other_part, native_part or another language_part command is found.

`mweave` ACTION: output the subsequent input lines in an environment whose name is the language name.

NAME:	native_part	**ARG:**	None
PARMETER:	`char_new`	**DEFAULT:**	`@N`

This command begins the native-language part. It can only occur after one of the 'begin section' commands. It must follow an other_part command if one is present in the same section.

`mtangle` ACTION: output a comment line, then output subsequent input lines until an end_sec command is found.

`mweave` ACTION: save the subsequent input lines in an internal buffer until an end_sec is encountered.

NAME:	other_part	**ARG:**	None
PARMETER:	char_old	**DEFAULT:**	@O

This command begins the other-language part. It can only occur after one of the 'begin section' commands.

mtangle ACTION: do nothing.

mweave ACTION: save the subsequent input lines in an internal buffer until a command line is encountered.

NAME:	para_def	**ARG:**	Mandatory
PARMETER:	char_para_def	**DEFAULT:**	@D

The command defines the value of a parameter. It can only appear outside any section. The parameter value defines by this command will take effect from the next section. The argument to this command should be an equation. The left hand side is a string representing the name of a parameter. The right hand side is also a string representing the value of the parameter. If the parameter takes only a boolean value, a null string, i.e., "" is interpreted as false.

5 Parameters

This section describes all parameters understood by mweb. Their values are all strings. A few of them must be single character string which are indicated in the description. Some of the strings are interpreted as boolean value. A null ("") string means false, and any other string means true.

The value of a parameter can be redefined by the user using the para_def command or the command line option -D. The syntax of parameter definition is

name = *value*

Both *name* and *value* are strings. A string is a sequence of non-blank characters optionally enclosed by a pair of characters which is the value of the parameter char_str_quote or by a left bracket which is the value of char_begin_para and a right bracket which is the value of char_end_para. The default characters are double quote("), left brace({) and right brace(}), respectively. For example, {String} and "String" are legal strings. A null string is a string of zero character. This can be written as either "" or {}.

Spaces around the equal sign are optional. The string on the left hand side of the equation should be the name of one of the parameters listed below. The string on the right hand side denotes the new value of this parameter.

Within a string, all characters are literal except the escape character. This is the current value of the parameter char_str_esc. The default string escape character is backslash(\). The interpretation of the string escape will be explained in the entry of the parameter `char_sstr_esc`. The default string of the parameters shown in the entries is in the form one has to type on the right hand side of the parameter definition equation. For example, the equation below defines the parameter `mac_sec` to the LaTeX macro `\subsec`:

```
"mac_sec" = "\\subsec"
```

where the double backslash is interpreted as a single backslash by the `mweb` programs.

NAME: `char_anon_sec` **DEFAULT:** `A`

This parameter associates with the `begin_anon_section` command which begins an anonymous section.

NAME: `char_begin_para` **DEFAULT:** `{`

This parameter begins the argument to a command. It must be a single character string.

NAME: `char_begin_sec` **DEFAULT:** `␣`

This parameter associates with the `begin_section` command which begins an ordinary section.

NAME: `char_begin_star_sec` **DEFAULT:** `*`

This parameter associates with the `begin_star_section` command which begins a major section.

NAME: `char_begin_tag` **DEFAULT:** `t`

This parameter associates with the `begin_tag` command. which begins a tagged text block in a tag file or marks the position for replacing by a tagged text block in a master file.

NAME: `char_cmd` **DEFAULT:** `@`

This parameter is the command escape character.

NAME: `char_end_para` **DEFAULT:** `}`

This parameter terminates the argument to a command. It must be a single character string.

NAME: `char_end_sec` **DEFAULT:** `-`

This parameter associates with the `end_section` command which terminates a section.

NAME: `char_end_tag` **DEFAULT:** `e`

This parameter associates with the `end_tag` command which terminates a tagged text block in a tag file.

NAME: `char_incl_file` **DEFAULT:** `I`

This parameter associates with the `incl_file` command which specifies the name of a tag file to be included.

NAME: `char_lang` **DEFAULT:** `L`

This parameter associates with the `language_part` command which begins a language part within a section.

NAME: `char_ml_sec` **DEFAULT:** `M`

This parameter associates with the `begin_ml_section` command which begins an 'ML only' section. Sections started with this command will not appear in the document if the ML only mode is active.

NAME: `char_new` **DEFAULT:** `N`

This parameter associates with the `native_part` command which begins the native-language part within a section.

NAME: `char_old` **DEFAULT:** `O`

This parameter associates with the `other_part` command which begins the other-language part within a section.

NAME: `char_para_def` **DEFAULT:** `D`

This parameter associates with the `para_def` command which defines the value of a parameter.

NAME: `char_str_esc` **DEFAULT:** `\\`

This parameter specifies the escape character in a string. Its value must be a single character string. When defining the value of a parameter, the string at the right hand side of the equation may contain this escape character. All characters in the string are literal except this escape character. It is interpreted together with the next character. If the next character is also an escape character, it becomes a single escape character. If the next character is an '`n`', it is interpreted as a 'newline' character. If the next character is a '`t`', it becomes a TAB character (ASCII code 9). If the next character is the current value of the parameter `char_str_quote`, it becomes that character. If the next character is a digit from 0 to 7, then the next two characters must be digits as well. This three digit number is interpreted as an octal number specifying the ASCII code of a character.

NAME: `char_str_quote` **DEFAULT:** `\"`

This parameter specifies the quote character for strings. Its value must be a character string. This character marks the beginning and the end of a string in parameter value definition.

NAME: `doc_only_mode` **DEFAULT:** null string

The value of this parameter is boolean. When it is true, the document only mode is active.

NAME: `filter_mode` **DEFAULT:** null string

The value of this parameter is boolean. When it is true, the filter mode is active.

NAME: `insuffix` **DEFAULT:** `.m`

The value of this parameter is the default suffix of the input file. If the name of the input file given on the command line has no suffix, this string is appended to it. The `mweb` programs attempt to open a file of this resulting name for inputting.

NAME: `line_mode` **DEFAULT:** null string

The value of this parameter is boolean with default value false. It affects `mtangle` only. When this mode is active, the program lines in the output file will have the same line number as in the master file. The lines of the document part and other language part will become blank lines in the output file. This is useful in debugging since many compilers will report errors with line munber. By keeping the same line number in the program file, one can easily find and fix the errors in the master file.

NAME: `mac_after_tag` **DEFAULT:** `\\end{verbatim}\n`

The value of this parameter is output to the document file after the tagged text block.

NAME: `mac_before_tag` **DEFAULT:** `\\begin{verbatim}\n`

The value of this parameter is output to the document file before the tagged text block.

NAME: `mac_begin_arg` **DEFAULT:** `{`

The value of this parameter is output to the document file before a line from other-language part or native-language part.

NAME: `mac_begin_doc` **DEFAULT:** null string

The value of this parameter is output to the document file to mark the beginning of the document part of every section.

NAME: `mac_begin_env` **DEFAULT:** `\\begin{paralines}`

The value of this parameter is output to the document file to mark the beginning of the environment for program lines.

NAME: `mac_begin_lang` **DEFAULT:** `\\begin`

The value of this parameter is output to the document file before the subsequent lines in a language part. It is followed immediately by the name of the language enclosed in the delimiters which are the values of the parameters `mac_begin_arg` and `mac_end_arg`.

NAME: `mac_begin_native` **DEFAULT:** `\\begin{native}\n`

The value of this parameter is output to the document file before the lines from the native-language part if the operation mode is not parallel.

NAME: `mac_begin_other` **DEFAULT:** `\\begin{other}\n`

The value of this parameter is output to the document file before the lines from the other-language part if the operation mode is not parallel.

NAME: `mac_changed` **DEFAULT:** `[c]\n`

The value of this parameter is output to the document file following the string of the parameter `mac_begin_env` if the contents of the two language parts are different.

NAME: `mac_end_arg` **DEFAULT:** `}`

The value of this parameter is output to the document file after a line from other-language part or native-language part.

NAME: `mac_end_doc` **DEFAULT:** null string

The value of this parameter is output to the document file to mark end of document part of every section.

NAME: `mac_end_env` **DEFAULT:** `\\end{paralines}\n`

The value of this parameter is output to the document file to mark the end of the environment for program lines.

NAME: `mac_end_lang` **DEFAULT:** `\\end`

The value of this parameter is output to the document file after the lines in a language part. It is followed immediately by the name of the language enclosed in the delimiters which are the values of the parameters `mac_begin_arg` and `mac_end_arg`.

NAME: `mac_end_native` **DEFAULT:** `\\end{native}\n`

The value of this parameter is output to the document file after the lines from the native-language part if the operation mode is not parallel.

NAME: `mac_end_other` **DEFAULT:** `\\end{other}\n`

The value of this parameter is output to the document file after the lines from the other-language part if the operation mode is not parallel.

NAME: `mac_end_sec` **DEFAULT:** `\n`

The value of this parameter is output to the document file to mark the end of a section.

NAME: `mac_filler` **DEFAULT:** `\n`

The value of this parameter is output to the document file following the string of the parameter `mac_begin_env` if the contents of the two language parts are the same.

NAME: `mac_line` **DEFAULT:** `\\pline`

The value of this parameter is output to the document file at the beginning of every pair of program lines.

NAME: `mac_native_line` **DEFAULT:** `\n`

The value of this parameter is output to the document file at the beginning of every line of the native-language part if parallel mode is inactive.

NAME: `mac_other_line` **DEFAULT:** `\n`

The value of this parameter is output to the document file at the beginning of every line of the other-language part if parallel mode is inactive.

NAME: `mac_sec` **DEFAULT:** `\\subsect`

The value of this parameter is output to the document file to start an ordinary section, or ML only section if the ML only mode is inactive.

NAME: `mac_sec_default` **DEFAULT:** `\\subsect{}`

The value of this parameter is output to the document file to start an ordinary section if there is no section name given.

NAME: `mac_spec_char` **DEFAULT:** `\\char\"`

The value of this parameter is prefixed to the ASCII code in hexadecimal notation of each special character which occurs in the program text in either other-language part or native-language part when raw tex mode is not active.

NAME: `mac_star_sec` **DEFAULT:** `\\sect`

The value of this parameter is output to the document file to start a major section.

NAME: `ml_comm_begin` **DEFAULT:** `%-`

The value of this parameter is output to the program file to mark the beginning of a comment.

NAME: `ml_comm_end` **DEFAULT:** `-%`

The value of this parameter is output to the program file to mark the end of a comment.

NAME: `ml_line_count` **DEFAULT:** `200`

The value of this parameter specifies the maximum number of program lines in a section. It must be a string consisting of only decimal digits.

NAME: `ml_line_length` **DEFAULT:** `80`

The value of this parameter specifies the maximum number of characters on a program line. It must be a string consisting of only decimal digits.

NAME: `ml_only` **DEFAULT:** null string

The value of this parameter is boolean. When it is true, the ML only mode is active.

NAME: `outsuffixmtangle` **DEFAULT:** `.ml`

The value of this parameter is the default suffix of the output file of `mtangle`. If only one file name is given on the command line, this string is appended to it after any suffix has been stripped to obtain a name for the output file.

NAME: `outsuffixmweave` **DEFAULT:** `.tex`

The value of this parameter is the default suffix of the output file of `mweave`. If only one file name is given on the command line, this string is appended to it after any suffix has been stripped to obtain a name for the output file.

NAME: `outsuffixwinnow` **DEFAULT:** `.m`

The value of this parameter is the default suffix of the output file of `winnow`. If only one file name is given on the command line, this string is appended to it after any suffix has been stripped to obtain a name for the output file.

NAME: `parallel_mode` **DEFAULT:** `true`

The value of this parameter is boolean. When it is false, the lines from the other-language part and from the native-language part are output to the LaTeX file as two blocks of text with the other-language block appearing first. The blocks are enclosed by the strings which are the current values of the parameters `mac_begin_other`, `mac_end_other` and `mac_begin_native`, `mac_end_native`, respectively.

NAME: `spec_chars` **DEFAULT:** `‘#$%&~_^{}\\`

Characters in this string are special characters. If the raw text mode is inactive, every special character in either other-language part or native-language part will be converted to their ASCII codes and prefixed with the string value of the parameter `mac_spec_char` when output to the document file.

NAME: `str_blank` **DEFAULT:** `\n\t␣`

This parameter string contains all characters which are interpreted as blank character in a parameter definition.

NAME: `raw_text_mode` **DEFAULT:** null string

The value of this parameter is boolean. When it is false, the lines from the other-language part and from the native-language part are copied to the LaTeX file without being processed. When it is true, special characters appearing in these lines are converted into their ASCII codes and prefixed with the string value of the parameter `mac_spec_char`. The parameter string `spec_chars` defines which character is special.

NAME: `tab_spaces` **DEFAULT:** `8`

A TAB character in any language parts will be converted into a number of space characters when the raw text mode is not active. The value of this parameter specifies the exact number of space characters to be output into the LaTeX file. It must be a string consisting of only decimal digits.

NAME: `tangle_lang` **DEFAULT:** null string

The value of this parameter is the name of the language `mtangle` is to output to the program file. It is set either by the command line option `-L` or by assigning a new value directly to this parameter.

Note: a new parameter can be defined by simply using a name on the left hand side of a parameter definition equation, but basic `mweb` programs do not understand it. There is no way to tell the programs how to use new parameters yet.

6 An Extension to the `mweb` suite

The **winnow** and **mmerge** programs were added to the suite of `mweb` tools to permit the creation of a "master" format file from a fairly ordinary LaTeX source in which interactive HOL sessions were included. The circumstances included a very large document with many formatting-introduced errors, and generally datedness as the HOL system evolved during the lengthy preparation of the material. The intent was to generate a file which could be used with the `mweb` suite of tools to (nearly automatically) update and maintain the interactive HOL material in the document.

By providing a filter from a LaTeX source, there is flexibility in chosing which file to edit. Both the LaTeX source and the master files are possible, depending entirely on the inclination of the user. Using the `mweb` suite to update the files with newly run interactive HOL sessions when required greatly reduces the overhead of maintaining the currency of a document.

In order to make the process as automatic as possible, each file needed to contain all of the ML source code required for execution in HOL. In some cases this required material which was not to appear in the final document to be included in the source. A "hidden hol" environment was created for this purpose.

The modified processing sequence is shown in Figure 5. Observe that the two instances of LaTeX files in the diagram can be interpreted as a cycle in the file processing.

6.1 Using `winnow`

An informal specification for the function of the winnow utility is given in Figure 6. There are two envionments in the LaTeX document which are modified by the filter. The first is the environment which contains displayed interactive HOL sessions, delimited by `\beginhol` and `\endhol`. These commands are LaTeX macros which define how the material will be displayed (in this case, a framebox the allowed textwidth size encloses the verbatim-displayed material.)

The other environment contains the HOL commands which will not appear in the final document, but which are necessary for the extracted ML file to be

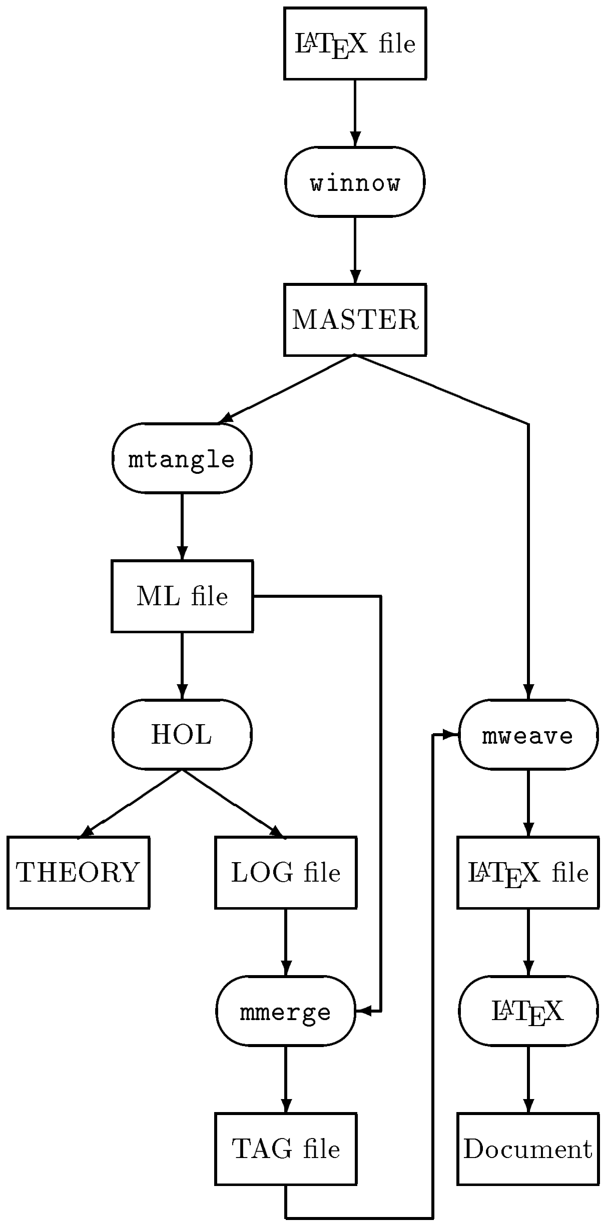

Figure 5: Extended operation of `mweb`

input	*winnow output*
\beginhol	\beginhol
	@ {<infilename>_<seccount>}
	@t{<infilename>_<seccount>}
	@N
% hol comment %	% hol comment %
#command line;;	command line ;;
returned value	
% comment line 1	% comment line 1
comment line 2 %	comment line 2 %
#command line 1	command line 1
command line 2;;	command line 2;;
returned value/feedback	
...	
\endhol	@-
	\endhol
other input	other input
...	...
\begin{hh}	\begin{hh}
	@ {$<infilename>_<seccount>}
	@t{$<infilename>_<seccount>}
	@N
command 1;;	command 1;;
...	...
command n;;	command n;;
\end{hh}	@-
	\end{hh}

Figure 6: Input format and corresponding output from `winnow`

executable. The definition of this macro follows.

```
\makeatletter
\def\hh{\@bsphack
            \let\do\@makeother\dospecials\catcode`\^^M\active
            \let\verbatim@startline\relax
            \let\verbatim@addtoline\@gobble
            \let\verbatim@processline\relax
            \let\verbatim@finish\relax
            \verbatim@}
\let\endhh=\@esphack
\makeatother
```

A number of formatting requirements and assumptions exist.

1. All displayed HOL commands (ie. those enclosed by the commands `\beginhol` and `\endhol`) are preceded by a single prompt `#`, regardless of how many lines the command occupies. This prompt is the first character on the line on which it occurs. The program expects that the character sequence beginning after the prompt and continuing until the **end** of the line on which the characters `;;` appear, will comprise an executable HOL command. Thus, comments may be included within a command for example.

2. Comments begin with the ML comment character `%` as the first character on a line, and continue up to the next occurrance of a comment character. Multiline comments are permitted, but "supercomments" (delimited by `%<` and `>%`) are not supported to enclose simple comments.

3. The commands `\beginhol`, `\endhol`, `\begin{hh}`, and `\end{hh}` must all be at the beginning of a line. No indentation is permitted.

4. Hidden HOL commands (ie. those enclosed by `\begin{hh}` and `\end{hh}`) must not have a prompt; everything thus enclosed is expected to be suitable for feeding directly to HOL without modification or stripping prompt characters. All commands must be terminated with `;;`.

The program shares the same front end as the other utilities in the `mweb` suite, so the command line argments, etc. have the same format. There are different defaults for some parameters, as well as a few added parameters. These are listed below.

The most visible difference is that all commands in the LaTeX source file begin with the `\` character since they are also commands to LaTeX. As with the other mweb tools, the exact command strings and output strings can be customized by changing the values of a set of parameters.

NAME:	begin_section	**ARG:**	None
PARMETER:	char_begin_sec	**DEFAULT:**	\\beginhol

This command begins a section of displayed interactive HOL material.

winnow ACTION: pass the command unchanged to output. Then output three mweb macros as follows: out_char_cmd out_begin_sec_cmd with an argument made up of the current values of infilename and seccount, enclosed by mac_begin_arg and mac_end_arg; out_char_cmd char_begin_tag with an argument made up of the current values of infilename and seccount, enclosed by mac_begin_arg and mac_end_arg; out_char_cmd out_char_new_cmd. Then, copy all HOL commands and comments to the output until the command char_end_sec is found. HOL commands begin with a char_hol_prompt and end with char_end_hol_in. Comments are enclosed by char_hol_begin_comment and char_hol_end_comment.

NAME:	end_section	**ARG:**	None
PARMETER:	char_end_sec	**DEFAULT:**	\\endhol

This command terminates a section of displayed interactive HOL material.

winnow ACTION: Output mweb macro out_char_cmd out_end_sec_cmd followed by the command itself unchanged.

NAME:	begin_ml_section	**ARG:**	None
PARMETER:	char_ml_sec	**DEFAULT:**	\\begin{hh}

This command begins a 'hidden HOL' section.

winnow ACTION: pass the command unchanged to output. Then output three mweb macros as follows: out_char_cmd out_begin_sec_cmd with an argument made up of char_hide_indicator and the current values of infilename and seccount, enclosed by mac_begin_arg and mac_end_arg; out_char_cmd char_begin_tag with an argument made up of char_hide_indicator and the current values of infilename and seccount, enclosed by mac_begin_arg and mac_end_arg; out_char_cmd out_char_new_cmd. Then, copy all HOL commands and comments to the output until the command char_end_hide is found.

NAME: end_hide_hol_section **ARG:** None
PARMETER: char_end_hide **DEFAULT:** \\end{hh}

This command terminates a section of displayed interactive HOL material.

winnow ACTION: Output mweb macro out_char_cmd out_end_sec_cmd followed by the command itself unchanged.

The parameters understood and used by winnow in a different way or in addition to those of the rest of mweb are described in the following. All the values are strings. The values of the parameters may be changed by the user using the para_def command or the command line option -D. First we present the parameters which are used differently or have different defaults.

NAME: char_begin_sec **DEFAULT:** beginhol

This parameter associates with the begin_section command which begins a section of visible interactive HOL.

NAME: char_cmd **DEFAULT:** \\

This parameter is the command escape character.

NAME: char_end_sec **DEFAULT:** endhol

This parameter associates with the end_section command which terminates a section of visible interactive HOL.

NAME: char_begin_star_sec **DEFAULT:** \n

This parameter is an unused entry in an internal procedure to process a section of text.

NAME: insuffix **DEFAULT:** .tex

The value of this parameter is the default suffix of the input file. If the name of input file given on the command line has no suffix, this string is appended to it. The winnow program attempts to open a file of this resulting name for input.

NAME: char_ml_sec **DEFAULT:** begin{hh}

This parameter associates with the begin_ml_section command which begins a 'hidden HOL' section.

NAME: `filter_mode` **DEFAULT:** true

The value of this parameter is boolean. When it is true, the filter mode is active. It must be set to true for `winnow`.

Added parameters follow.

NAME: `out_char_cmd` **DEFAULT:** `@`

This is the output command escape character.

NAME: `out_begin_sec_cmd` **DEFAULT:** ␣

This is the output `begin_section` command string.

NAME: `out_end_sec_cmd` **DEFAULT:** `-`

This is the output `char_end_sec` command string.

NAME: `out_char_new_cmd` **DEFAULT:** `N`

This is the output `native_part` command string.

NAME: `out_ml_sec` **DEFAULT:** `M`

This is the output `begin_ml_section` command string.

NAME: `char_hol_prompt` **DEFAULT:** `#`

This string identifies the start of a HOL command.

NAME: `char_end_hol_in` **DEFAULT:** `endhol`

This string identifies the end of a HOL command.

NAME: `char_hol_begin_comment` **DEFAULT:** `%`

This string identifies the start of a HOL comment.

NAME: `char_hol_end_comment` **DEFAULT:** `%`

This string identifies the end of a HOL comment.

NAME: `char_end_hide` **DEFAULT:** `end{hh}`

This parameter associates with the end_hide_hol_section

NAME: `char_hide_indicator` **DEFAULT:** `$`

This parameter marks a section as containing hidden hol commands.

6.2 Using mmerge to prepare the tag file

Once the "master" file has been created, an ML source file can be extracted using the command `mtangle <filename>`. Given the presence of a master file named "`<filename>.m`", this will generate an ML file named "`<filename>.ml`". A log of executing the source file should be saved in a file named `<filename>.log`. This file can be combined with the original ML file to generate the "tag" file, which will contain segments of (slightly modified) interactive sessions of HOL to be reinserted into a LaTeX document.

The `mmerge` utility is independent of the `mweb` system. Its purpose is to create the "tag" files. In the presence of files `<filename>.log` and `<filename>.ml` the proper usage is given as follows.

```
mmerge [-p prompt] [-hH] <filename>
```

The `p` option permits recognizing other than the default HOL prompt `#`. The `h` and `H` options cause a usage help message to be printed to standard output. A file labelled `<filename>.tag` will be generated if the program successfully executes.

The source and log files are merged together such that:

- Imbedded comments are passed to the output untouched, with no preceding prompt.
- Every visible command is passed to the output file, preceded by a single prompt character `#`. Hidden commands are passed to the output file, but without a prompt.
- The system response to visible commands is passed to the output.
- Imbedded comments placed by the `mtangle` program identify the extent of "sections", which were delineated by the commands `\beginhol`, `\endhol` and `\begin{hh}`, `\end{hh}` in the original file. The corresponding tag sections are delineated by commands of the form `@t{<filename>_<seccount>}` and `@e` for visible sections, and `@t{$<filename>_<seccount>}` and `@e` for hidden sections,

It has been the experience of the author that some manual editing of the tag files may be necessary to cope with unusual demands on the program. Examples include eliminating the listing of a long series of proved subgoals when a goal is finally solved (an imbedded comment that this is being done serves both to inform the reader, and remind the person processing the text), splitting a section into two if the original LaTeX file had split the system feedback so (the empty section serves as a useful reminder in this case), and reformatting the output to limit line lengths if the HOL pretty printer was not configured appropriately. A list of all such manual changes should be kept to ease repeating of the process without introducing random errors.

6.3 Using `mweave` to put it all together

For the most part the standard `mweb` utilities when applied to the output of the `winnow` program are run as described in the main documentation. The only exception to this is the `mweave` utility, which requires the setting of particular parameters. Filter mode must be turned on by using the command option `-f`. In addition, the following parameters must be set as shown, typically in an include file.

```
@D"doc_only_mode"="true"
@D"char_begin_sec"="$"
@D"char_anon_sec"=" "
@D"mac_before_tag"=""
@D"mac_after_tag"=""
@D"mac_end_sec"=""
```

7 The implementation

This section describes the implementation briefly to provide a guide to anyone who needs to extend or modify the programs to suit special applications.

The `mweb` utility program is currently implemented in the C language. The building of the programs is controlled by a `Makefile`. Typing the command

```
make all
```

will compile all the programs. To install the programs, the path name of the binary directory specified in the `Makefile` may need to be change. After modifying this to reflect your system set up, typing the command

```
make install
```

will install all the programs and the LaTeX macro file for the default parallel environment.

The source of the programs consists of several `.c` files and two header files. The code common to all programs is in the file `common.c`. The header files are also common to all programs except mmerge. The code specific to each program is in a file named after the program, e.g., the file mweave.c contains the code for mweave.

The three programs, mtangle, mweave and winnow, are basically simple recursive descent parsers. They read the input file line by line. If the input line is a command line, the program acts accordingly. Otherwise, it will write the line to the output file or discard it depending on the state the program is in.

7.1 Parameter handling

The code for handling parameters is common to all programs, so it is in the file common.c. The parameters are stored in a parameter table, paratab. Each entry of this table represents a parameter. It is in the form of a name-value pair. Both the name and the value are strings. The table is a static array of structures which is initialized with default strings. Symbolic names representing the indexes to the entries are defined in the header file mweb.h. They all begin with the upper case letter 'P'. For example, the index to the parameter char_cmd is Pchar_cmd. To improve the speed of accessing the parameter values, a set of static pointers are declared and initialized to point to the values of more frequently used parameters. The names of these pointers are identical to the names of the parameters whose values are accessing by the pointers.

The function set_para takes the name and the value of a parameter as its arguments. It updates the value of the parameter in the table if the named parameter is already in the table. Otherwise it adds the parameter into the table. This function returns the index to the entry. The function get_para takes the name of a parameter, and returns its value. If it cannot find the named parameter, a NULL pointer is returned. Note that the value of a parameter may be a NULL string. A pointer to a NULL string is not equal to a NULL pointer.

Some parameters are special. In addition to its value, each special parameter has an associated flag. The association between a parameter and a flag is stored in a special parameter table, special_ptab. When a special parameter is updated, the value of the associated flag is also updated.

The procedure proc_para_def takes an equation in the format accepted by the command para_def. It updates the entry in the parameter table. If it is a special parameter, the flag in the special parameter table is also updated. This function is called whenever a para_def command is encountered.

7.2 Initialization

When the mweb programs start, the first task is to call the initialization routine, init. This sets up the initial values of some variables which are common to all programs. It then calls the routine proginit which does the program specific initialization.

Next, the programs process the command line. The action for simple options, such as `-h`, is trivial. The help message is printed and the program terminates. the same action is taken if an unknown option is specified. The action for `-I` is to open and process the named file which is expected to contain parameter definition and/or tag sections. This is done by the routine `proc_incl_file`. The action for other options is to assign the appropriate value to the parameter by simply calling the function `proc_para_def`.

The names of the input and output files specified in the command line are stored in the strings `infilename` and `outfilename`. As mentioned in Section 2, if only one name is given, it is taken as the input file name. The output file name is derived from the input file name. This is done by the routine `fileinit`. It also attempts to open the files and connects the input and output streams `instream` and `outstream` to these files. All subsequent I/O is perform via these streams.

The global variables `linecount` and `seccount` keep track of the number of lines and sections, respectively, which have been read from the current input file.

7.3 The `mtangle` program

The task of `mtangle` is relatively simple. When a section is encountered, the program enters the `proc_sec` routine. Within this routine, the variable `status` keeps track of which part the program is in. Initially, it has the value `NULL`. When entering native-language part, `status` is set to the value `NEW`. When `status` has this value, all input lines are copied to the output stream. When it has other value, all input lines are discarded.

7.4 The `mweave` program

Similar to the `mtangle` program, `mweave` enters the routine `proc_sec` when entering a section. However, `mweave`'s `proc_sec` routine is more complicated. It first checks which section-begin command has been seen, and then outputs the corresponding parameter. The program should now be in the document part. It simply copies all input lines to the output verbatim until a command line is found.

Commands are checked by the function `is_command`. It takes two arguments: the first is the current input string, and the second is an index to a parameter. It returns a non-zero value if the input string is equal to the parameter. This value is the length (number of characters) of the parameter.

The variable `status` keeps track of which part of a section the program is in. The value `IN_DOC` indicates that the program is in document part. When an other_part or native_part command is found, `status` is set to the values `IN_O` or `IN_N`, respectively.

When mweave is in one of the language parts, the input lines are saved in a buffer by the function store_buf. The buffers old_buf and new_buf are for this purpose. When the end of section is reached, the buffers are processed and written to the output.

7.5 Tag processing

Text lines of all tags are stored in a table, incltext. This is indexed by a tag table tags. Both of these tables are of type array of LINEs. A LINE is a structure of two fields: an integer field which is the index to a table entry, and a string field which contains the text of a line. The string field of tags contains the name of a tag section. The corresponding integer field contains the index to the first text line of the tag section in the incltext table. The integer field of the incltext table points to the next text line of the same tag section. The integer field of the last line is equal to zero. The pointers tagptr and textptr point to the first available space in the table tags and incltext, respectively.

When a begin_tag command is found in an include file, i.e., in the routine proc_incl_file, the name of the tag section is stored in the tags table. The variable state is set to INTAG. All subsequent text lines until an end_tag command are stored in the incltext table.

When a begin_tag command is found in the document part of a section, the mweave program enters the routine proc_tag. This routine searches the tags table to find a tag section of the same name, and writes the text lines to the output in place of the tag command. A warning is issued if no such tag section can be found.

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.

[2] Donald E. Knuth. *TEX: The Program*, volume B of *Computers & Typesetting*. Addison-Wesley, Reading, MA, 1986.

Index