EXTENDING GNU TROFF TO PRODUCE HTML THROUGH THE TECHNIQUE OF NEXT EVENT SIMULATION

GAIUS MULLEY, WERNER LEMBERG

School of Computing, University of Glamorgan, CF37 1DL, UK
E-Mail: gaius@glam.ac.uk
Kl. Beurhausstr. 1, 44137 Dortmund, Germany
E-Mail: wl@gnu.org

Abstract: This paper reports on a technique used to generate accurate HTML output from GNU Troff. GNU Troff is a typesetting package which reads plain text mixed with formatting commands and produces formatted output. It supports a number of devices and now supports the production of HTML. The paper discusses the design of the HTML device driver grohtml and modifications made to GNU Troff. The front end program troff was modified to maintain a reduced state machine which is examined each time a glyph is passed to the back end device driver (post-grohtml). Any change in system state between the production of two glyphs results in a sequence of events being passed to the device driver. There is a direct correspondence between this technique and creating a script for a next event simulation queue. Furthermore the device driver reconstructs the system state and formats the HTML according to state changes caused when processing the event queue. This technique works well, as it minimises the state information passed from front end to back end device driver whilst still preserving the high level layout of the text. Using this technique GNU Troff effectively translates input source into another mark-up language and thus this technique could be extended to translate GNU Troff documents into any of the OpenOffice supported formats. Troff has been in use for three decades now and is still actively used by authors. Troff’s biggest use, however, is to format manual pages for GNU/Linux and other UNIX like operating systems. Introducing this facility into GNU Troff provides users with the ability to translate legacy documents into HTML and in the future to a format supported by OpenOffice.

Keywords: groff, troff, grohtml, HTML, simulation.

INTRODUCTION

GNU Troff is a reimplementation of the program troff which is available on the UNIX operating system. The original troff was written in PDP-11 assembly language by Joe Ossanna in 1973. Two years later it was rewritten in C and afterwards it went through a series of revisions until Joe Ossanna died in 1977.

Brian Kernighan continued troff development for the next 15 years and it is testament to the original design that the input language remained the same. Much of the original documentation received minor changes as new troff releases were issued and therefore Joe’s name was retained on these manuals [Ossanna, 1992].

Published in the International Journal of SIMULATION Vol. 6 No 7-8, ISSN 1473-804x online, 1473-8031 print, 2005

Brian Kernighan modified troff so that it could produce output for a number of different typesetting devices, while at the same time retaining the same input language specification. The input language was so robust that a number of preprocessors were written to provide reference [Tuthill, 1986], table [Lesk, 1976a], picture [Kernighan, 1991, Wyk, 1982] and equation handling [Kernighan, 1977, Kernighan, 1976] (refer, tbl, pic and eqn respectively). These programs were executed in a pipeline and troff transformed the heavily preprocessed formatting commands and text into device dependent output [Kernighan, 1978]. Although the input to troff was device independent it was also very low level and therefore users were encouraged to use macro sets when producing documents [Lesk, 1976b]. During the 1980s the internals were modestly revised and a number of new macro sets were written [Allman, 1980, Smith, 1980]. The macro sets provided freedom in document styling (similar to modern HTML style sheets) and they included: arbitrary style headers and footers; arbitrary style footnotes; automatic sequence numbering for
paragraphs, sections, etc; multiple column output; dynamic
font and point-size control; arbitrary horizontal and vertical
local motions at any point; mathematical bracket construction,
and line drawing functions [Ossanna, 1992].

During the 1980s troff was extended to handle many
different devices, this was accomplished by
splitting the task of troff into two components.
The front end ditroff produced device independent
code and the back end device driver which simply
translated the device independent code into the
target device commands.

James Clark began to work on the GNU implementa-
tion of the troff family of tools in 1989. This was
to be a completely new implementation of all the
preprocessors, the ditroff program and the macro
sets. The first release of groff (version 0.3.1)
occurred in June 1990 and it included a replacement
for ditroff, eqn, tbl and pic. It also included a
replacement for the me macros and the man macros.
The replacement programs were mostly written in
C++ and often supported extensions and removed
various static data size limitations. Since 1999 the
groff package has had new maintainers and has
undergone active development. It supports the
common macro sets associated with troff (man, me, ms,
mdoc and mm). However groff also provides a
modern macro set (mm) and also provides new pre-
processors and support for colour.

DESCRIPTION OF THE PROBLEM

Troff was widely used in the 1970s and 1980s. All
UNIX documentation, release notes for both the
AT&T and BSD UNIX variants was written in
troff. Many papers in the CACM and Software
Practice and Experience journals were also produced
using troff. Even today all UNIX and GNU/Linux
manual pages are written in troff using the man
macro set and some authors advocate using troff
above other WYSIWYG tools to typeset their
books [Tanenbaum, 1997, Stevens, 1998, Schaffer,
2004].

Groff provides compatibility with troff as well as
many modern enhancements, image handling, colour
and limited pdf mark capability. Clearly the addition
of an HTML device driver would be useful.

GNU Troff copied the original design of UNIX Troff,
which maps the input source onto a physical device
through the use of a device independent language.
Through this language it effectively plots each glyph
at Cartesian coordinate position on a page. The dif-
ficulty in translating troff source into HTML is
exacerbated by the fact that both HTML and troff
source are mark-up languages. As the GNU Troff
and original UNIX Troff packages rely heavily on the
pipeline principle it has naturally led to the pre-
processors (pic, tbl and eqn) translating their high
level commands onto much lower level troff com-
mands. In turn, this has meant that much of the high
level information (for example where a table starts
and ends) is lost. Furthermore, using any of the
macro packages will result in many low level com-
mands to format abstracts, titles, footnotes, para-
graphs, lists, hanging indents etc. Consequently,
translating troff input source into HTML cannot be
achieved by writing a device driver which simply
reads ditroff input and produces HTML output.
Consider the diagram in Figure 1 which shows the
key components of a simple groff command line
invocation together with a synopsis of the data pass-
ing though the pipeline. In Figure 1 the troff input
consists of requests or calls to macros (lines prefixed
with a period) and text. Troff input might also
include text with escapes, for example the word
program can be typeset by temporarily reducing the
point size by 1 and altering the font to Courier. The
word and escapes can be encoded as \s-1\fCprogram\s+1\fP.

In Figure 1 the final output is PostScript (the default
device) and the invocation also includes the ms
macro package. Figure 2 shows how a PostScript
printer interprets the output from Figure 1.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig2.png}
\caption{displaying the PostScript output}
\end{figure}

Notice how the ditroff output only knows which
font is to be used, the font size and the position that
the glyph should be placed. Another example that
an HTML device driver must translate accurately is
shown in Figure 3.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig3.png}
\caption{examples of indented paragraphs}
\end{figure}

Here we see from the output shown in Figure 4 that
the ms macro set diverted the indented paragraph
label parameter once, twice and threefold into a
macro. It then tests to see whether the macro width
is greater than a default length and if so it breaks the line before starting the indented paragraph. Clearly the HTML device driver needs to cope with these constructs, as they are also heavily used in manual pages.

Furthermore many documents will include encapsulated PostScript as shown in Figure 5. The processed output is shown in Figure 6. The issue here is that PostScript has become the default device driver and groff allows encapsulated PostScript to be handled as a special device case. Nevertheless this feature is so useful and it would be expected to be implemented in the HTML device driver. Groff also allows users to embed their own PostScript inside Troff source files (a technique similar to inlining assembly language within a high level language). In both cases users could reasonably expect the HTML device driver to translate the PostScript into a PNG image which is referenced by the HTML output.

A further problem which needs to be addressed is the limited number of glyphs available in HTML. For example HTML has a restricted set of mathematical glyphs and it is restricted in its ability to accurately position these glyphs [Musciano, 1998]. Groff in contrast can be used to typeset mathematical formulae and users will at least expect the groff HTML device driver to reproduce the formulae as an image. Figure 7 shows Troff input, which describes the definite integral.

When processed with the command `groff -e -ms` it produces the output as shown in figure 8.

![An example of encapsulated PostScript](image)

Figure 6: result of processing Figure 5 with `groff -ms`

![Figure 5: example of encapsulated PostScript usage](image)

Figure 4: result of processing Figure 3 with `groff -ms`

![Figure 3](image)

Figure 3: Simple title, heading and paragraph

![Figure 1](image)

Figure 1: Simple title, heading and paragraph

![An example of encapsulated PostScript](image)

Figure 5: example of encapsulated PostScript usage

![Figure 2](image)

Figure 2: Troff source and ditroff intermediate language

![Figure 7](image)

Figure 7: eqn example

![Figure 8](image)

Figure 8: output of Figure 7 as rendered on a
Finally the \texttt{troff} language is very rich in commands and requests. The language has the ability to construct macros at run time and it does this by diverting text and commands into a \textit{diversion}. Some authors of macro sets perform some highly impressive programming tricks using this technique. For example this technique has been used in the \texttt{ms} macro set to handle the \texttt{IP} request and the \texttt{TL} request seen earlier. It is also used to manage abstracts, authors names and references for papers. The problem is that when \texttt{troff} generates the \texttt{ditroff} output the meaning of the text is lost. The HTML device driver needs to know whether a line was centred, if the line was indented, when the centring ends, when indentation ends, when text is a section heading and when text is a title. It cannot guess this reliably by examining the glyph position and font information.

Conveying the indentation information to the device driver is particularly difficult as diversions can be created to manipulate this value when the diversion is re-read. Consider the example given in Figure 9.

```
.LP
.di foo
.br
.in +1i
hello
.in -1i
.br
.di
.in +2i
.br
.foo
.in +li
.br
.foo
```

Figure 9: diversion and indentation example

When processed with the command \texttt{groff -ms} it produces the output as shown in Figure 9.

```
 hello
```

Figure 10: output of Figure 9 as rendered on a PostScript device

Clearly an HTML device driver must also understand the diversions sufficiently so that the important formatting requests are accurately translated.

\section*{PREVIOUS WORK}

There have been a number of scripting solutions implemented which translate \texttt{troff} documents into HTML. These scripts operate at the macro set level and they independently translate the functionality of the various macros into HTML. For example scripts exist to translate the \texttt{ms} macros \texttt{IP}, \texttt{TL}, \texttt{LP}, \texttt{PP}, \texttt{DS} into indented paragraph; title; left aligned paragraph; paragraph and block text. The advantage of this approach is its simplicity but the disadvantages are that the scripts need to be maintained independently of the corresponding macro set and more importantly the scripts do not understand user defined macros. Often these scripts do not honour all the macro optional arguments or allow inclusion of encapsulated PostScript.

Another solution is \texttt{unroff} [Laumann, 1996], which is a Scheme-based, programmable and extensible \texttt{troff} translator which translates \texttt{troff} files into HTML-2.0. It supports the \texttt{me}, \texttt{ms}, and \texttt{man} macro sets as well as the front end preprocessors \texttt{tbl} and \texttt{eqn}. This is an ambitious project and it has been successful in translating manual pages and \texttt{troff} documents. The success is partly due to \texttt{unroff} understanding many of the underlying \texttt{troff} requests and having extra tag information inside its macro set descriptions. The main disadvantages to \texttt{unroff} are that it does not have a seamless interface to the \texttt{pic} preprocessor and currently it does not handle encapsulated PostScript or diversions.

The \texttt{doclifter} project [Raymond, 2005a] provides a means whereby many documents written in \texttt{mm}, \texttt{ms}, \texttt{me} and \texttt{mdoc} can be translated into XML-DocBook [Walsh, 1999]. Eric Raymond designed \texttt{doclifter} to operate at the semantic level and he acknowledges that to do a really good job requires some human polishing. Nevertheless \texttt{doclifter} has successfully translated 95\% of all manual pages in Fedora Core 3.

\texttt{Doclifter} translates \texttt{tbl} tables into DocBook table markup and \texttt{pic} input into SVG, however \texttt{eqn} is not translated.

\section*{OVERVIEW OF GROHTML}

The approach taken by the authors in the design of \texttt{grohtml} was to modify the front end \texttt{groff} binary so that when the HTML device is requested the \texttt{troff} input is run through the PostScript device driver and later through the HTML device driver. All images are created by converting the relevant PostScript page into a bitmap image after which the required area is cut and converted into the PNG [Roelofs, 1999] format. The pipeline structure is outlined in Figure 11. The preprocessors \texttt{tbl}, \texttt{eqn} and \texttt{pic} were modified to indicate where the start and end of a table, equation and picture occur. This information is then transparently read by the front end and determines the area of the bitmap file which is required for an image. The advantage of this approach is that \texttt{grohtml} can fully support
The macro sets \texttt{ms} and \texttt{man} were also slightly modified to issue a tag when \texttt{troff} encounters a section heading, title or indented paragraph. All tags are emitted together with the \texttt{ditroff} intermediate code to the device driver \texttt{post-grohtml}.

The \texttt{troff} program was also modified to handle another escape \texttt{	extbackslash 0} which has five variants and they are described in Table 1. These escapes are important as they allow preprocessors to tell \texttt{troff} to skip a block of text or issue a box of coordinates which define the area for a set of glyphs. This functionality is used during the PostScript pass to determine the region used by a table, picture or equation. Conversely it is used by the HTML pass to stop producing glyphs when a table, equation or picture is encountered; but replace these items with an HTML reference to an image.

\section*{STATE MACHINE SOLUTION}

If grohtml is implemented as described in the last section it will be able to translate all: encapsulated PostScript, tbl, pic, eqn, section headings, numbered headings and titles accurately. However it will still fail to accurately translate: centred lines, any indentation, preformatted blocks, tabulated text, line breaks and vertical spaces. A solution to this problem is to extend the repertoire of tags which are passed from \texttt{troff} to post-grohtml. This requires further modification to the \texttt{troff} so that it will issue tags whenever the above formatting requests are encountered. In essence the back end of \texttt{troff} maintains a simple state machine which represents the minimum information required to correctly reproduce the formatting requests listed above. Whenever a glyph is about to be written in the diversion output, \texttt{troff} needs to check whether this state has changed since the last glyph was written. If the state has changed then \texttt{troff} emits a series of tags which define the change in state. There is clear correspondence between tags appearing in the \texttt{ditroff} intermediate output and a next event queue albeit without any time component. The HTML driver \texttt{post-grohtml} reconstructs the system state according to the events it reads and in the process translates the glyphs into HTML text surrounded by the appropriate HTML tags [Musciano, 1998]. Furthermore \texttt{troff} can handle requests from diversions by pushing the current minimal HTML state before a diversion is read and popping it later once the diversion has finished. This allows diversions to centre lines, alter the indentation, and emit preformatted text, whilst restoring the state once the diversion has ended.

The complete set of tags which post-grohtml understands is shown in Table 2. Figure 16 shows the extended ditroff output when GNU Troff processes Figure 9. The command line to generate this information is shown below:

\begin{verbatim}
groff -Z -ms -Thtml ex9.n
\end{verbatim}

and the output in Figure 16 contains the minimum tags necessary in order to convey the meaning of the glyph formatting. The example shown in Figure 9 produces two lines with a single word \texttt{hello} on each line. The first line \texttt{hello} is indented at three inches and on the second line \texttt{hello} is indented at four inches. The example starts by declaring a left adjusted paragraph \texttt{.lp} and then it creates the diversion \texttt{.di foo}. All subsequent text and requests are diverted into \texttt{foo} until the second \texttt{.di} is encountered. The next request \texttt{.in +2i} informs the type-setter that an indentation of two inches is required. When \texttt{.foo} is encountered it rereds the diversion which performs another relative indent of one inch before emitting the glyph \texttt{hello}. The diversion then reduces the indentation by one inch before finishing. Textual input is resumed after the diversion \texttt{.foo} at the request \texttt{.in +1i} which performs a relative indent of one inch (at this point the indentation is set at three inches). Lastly the diversion is run again which emits the final \texttt{hello} four inches from the left hand margin.

Figure 16 shows the ditroff output together with an annotated description. It also shows extensions made to ditroff which include the extra tag information (lines starting with \texttt{x X devtag:} convey the minimum events necessary such that post-grohtml can reconstruct the \texttt{troff} state machine. As post-grohtml reconstructs the state machine it generates the appropriate HTML. Critically for this example it can be seen that here are only two \texttt{x X devtag:.in} events, one at 3 inches and the second at 4 inches, despite the example in Figure 9 having four \texttt{.in} requests.

\section*{EXAMPLES OF GROHTML OUTPUT}

The examples shown in this section were produced with \texttt{groff} version 1.19.2 and the test cases are those from Figures 3, 5, 7 and 9. All the examples were constructed using the following pipeline:

\begin{verbatim}
groff -Thtml -e -ms filename.ms | html2ps
\end{verbatim}

and these are shown in Figures 12, 13, 14 and 15 respectively. The test results appear satisfactory and show that the indentation using the IP macro works as expected. This macro uses a diversion to detect
**Escape** | **Meaning**
---|---
\00 | Disable any glyphs from being emitted at the outer level. †
\01 | Enable output of glyphs, provided that the escape occurs at the outer level. †
\02 | Providing the escape occurs at the outer level, enable output of glyphs and also write out to stderr the page number and four registers encompassing the glyphs previously written since the last call to \0.
\03 | Begin a nesting level.
\04 | End a nesting level.
\0[5P filename] | HTML device specific. Provided that this escape occurs at the outer nesting level write the filename to stderr. The position of the image, P, must be specified and must be one of l, r, c or i (left, right, centre, inline).

† Both \00 and \01 reset four troff registers which mark the top left and bottom right hand corners of an area containing all written glyphs.

Table 1: new troff escapes and their meaning

the width of its first argument before emitting the paragraph. The example shows how each line 1, line 2 and line 3 are vertically aligned. Notice that line 3 has a vertical space between itself and threefold. Figure 13 shows that groff can translate eqn input into HTML and images and Figure 15 shows that indentation within diversions behaves correctly.

At present the GNU Troff macro sets: ms, man, and mwww have the extra HTML tags included. Figure 17 shows the output of a component from the GNU pic guide [Raymond, 2005b] when processed through the following pipeline.

groff -pet -ms -Thtml pic.ms | html2ps -s2

It can be seen that grohtml correctly determines that the equation is inside a picture and generates one image. Figure 18 shows the grohtml manual page after it was processed by the pipeline:

groff -pet -man -Thtml grohtml.man

and viewed through a browser. In this case we can clearly see the vertical alignment correctly represented between lines which contain a command line option and those lines which are a component of a hanging paragraph. Figure 19 shows how the GNU Troff homepage appears when printed from a browser. The homepage was created by the following command line.

groff -pet -ms -Thtml webpage.ms

The file webpage.ms make extensive use of the mwww macro set which provides drop capital, unordered list, URL and image related macros. It also makes use of colour which was recently added to GNU Troff.

**CONCLUSIONS AND FURTHER WORK**

In conclusion the grohtml device driver has been constructed and integrated into groff. This work has taken six years of part time effort and amounts to approximately 9600 lines of C++ code (not including macro set modifications). In due course the groff-1.19.2 will be formally released and this will be the first beta release of grohtml. During the development of grohtml a number of additional
features were needed to produce useful HTML. These features were applied as generically as possible and include the introduction of colour. Colour is now supported by the ASCII, PostScript, HTML, and DVI devices. GNU pic has also been extended to support colour.

The principles behind the HTML device driver appear sound, however there are still minor bugs in grohtml. Future work will include bug fixing and also modifications to tbl so that troff tables can be translated into HTML <table> constructs rather than an image. The remaining macro sets (mm, me, mdoc and mom) will also be updated to include the devtag requests necessary for grohtml to detect section headings, titles and indented paragraphs.

As the OpenOffice file formats are fully published it should be possible to construct a new GNU Troff device driver (grodoc) which utilizes the extended ditroff intermediate language. This new device driver could be built using a similar design to that of grohtml.

ACKNOWLEDGEMENTS

The authors would like to thank all the readers and contributors of the groff mailing list for their patience with this project and also for their many valuable bug reports over the last six years. We would also like to thank our respective employers for funding our time while we were working on groff.

Finally a great debt of thanks is owed to James Clark without whom there would be no groff.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.SH</td>
<td>start of section heading</td>
</tr>
<tr>
<td>.NH</td>
<td>start of numbered heading</td>
</tr>
<tr>
<td>.col n</td>
<td>start of column n</td>
</tr>
<tr>
<td>.eo.h</td>
<td>end of heading</td>
</tr>
<tr>
<td>.tl</td>
<td>start of title</td>
</tr>
<tr>
<td>.eo.tl</td>
<td>end of title</td>
</tr>
<tr>
<td>.fi n</td>
<td>if n is 0 then treat as preformatted text</td>
</tr>
<tr>
<td>.sp</td>
<td>vertical line space</td>
</tr>
<tr>
<td>.in n</td>
<td>indent next line n units</td>
</tr>
<tr>
<td>.ll n</td>
<td>line is n units long</td>
</tr>
<tr>
<td>.po n</td>
<td>left offset is n units</td>
</tr>
<tr>
<td>.ta list</td>
<td>list defines the tab settings</td>
</tr>
<tr>
<td>.ti n</td>
<td>next line is temporarily indented n units</td>
</tr>
<tr>
<td>.ce n</td>
<td>centre next n lines</td>
</tr>
<tr>
<td>.eol</td>
<td>end of input line</td>
</tr>
<tr>
<td>.br</td>
<td>line break forced</td>
</tr>
</tbody>
</table>

Table 2: tags passed from troff to post-grohtml

An example of encapsulated postscript

Figure 12: the result of HTML output from Figure 3

Figure 13: the result of HTML output from Figure 5

Figure 14: the result of HTML output from Figure 7

Figure 15: the result of HTML output from Figure 9

Figure 16: extended ditroff intermediate output
17.4. PIC and EQN

The Kernighan paper notes that there is a subtle problem with complicated equations inside `pic` pictures; they come out wrong if `eqn()` has to leave extra vertical space for the equation. If your equation involves more than subscripts and superscripts, you must add to the beginning of each equation the extra information \texttt{space 0}. He gives the following example:

\[
\frac{H(\omega)}{1 - H(\omega)}
\]

Figure 17: extract from "Making Pictures With GNU PIC" by Eric S. Raymond

Figure 17-1: Equations within pictures

Figure 18: grohtml manual page showing vertical alignment of hanging paragraphs

Figure 19: GNU Troff homepage when printed from a browser

REFERENCES

Allman Eric P. 1980, Writing Papers with NROFF using \texttt{−me}, University of California, Berkeley, California 94720.


Laumann Oliver 1996, Unroff - programmable, extensible troff translator \url{http://www-rn.informatik.uni-bremen.de/software/unroff}.

Lesk M. E. 1976, Tbl — A Program to Format Tables, Bell Laboratories, Murray Hill, New Jersey 07974.


**BIOGRAPHY OF AUTHORS**

Gaius Mulley is a senior lecturer at the University of Glamorgan. He is the author of GNU Modula-2 and the groff HTML device driver grohtml. His research interests also include performance of microkernels and compiler design. He obtained a BSc(Hons) and PhD in Computer Science from the University of Reading and later worked for Meiko Scientific.

Werner Lemberg (born 1968 in Vienna, Austria) is working as a conductor and singers’ coach at the municipal theatre of Koblenz, Germany. In his spare time he maintains the groff package and actively develops the FreeType font rendering library.