

Survey of Global Regular Expression Print (GREP) Tools

March 2, 2004

Tony Abou-Assaleh and Wei Ai
Faculty of Computer Science, Dalhousie University
{taa,weia}@cs.dal.ca

Abstract

The UNIX `grep` utility marked the birth of a global regular expression print (GREP) tools. Searching for patterns in text is important operation in a number of domains, including program comprehension and software maintenance, structured text databases, indexing file systems, and searching natural language texts. Such a wide range of uses inspired the development of variations of the original UNIX `grep`. This variations range from adding new features, to employing faster algorithms, to changing the behaviour of pattern matching and printing. This survey presents all the major developments in global regular expression print tools, namely the UNIX `grep` family, the GNU `grep` family, `agrep`, `cgrep`, `sgrep`, `nrgrep`, and Perl regular expressions.

I. Introduction

Searching for a pattern in a text file is an important task. All modern text editors provide functionality for searching text for a pattern. However, pattern matching also is useful beyond text editing. Users of UNIX found that they constantly use the `ed` command `g/re/p`, which means globally search for the regular expression and print the matching lines, where `re` is a regular expression, that a separate program was developed with the same name, `grep`. In effect, `grep` became an acronym for *Global Regular Expression Print* [Goedelbecker 1995].

It is believed that `grep` is the single most used tool in program comprehension. Empirical studies show that software engineers use `grep` extensively in their daily maintenance tasks [Singer and Lethbridge 1997]. Although it is generally accepted that, at least for basic regular expression matching, `grep` is easy to learn, easy to use, and its limitations are well understood, there is a debate among cognitive psychologists for the true reasons behind `grep`'s success. More specifically, it is a non-trivial task to learn from the success of `grep` in designing and developing new program comprehension tools.

Though useful and powerful, the original UNIX `grep` [IEEE and Open Group 2003b] was not sufficient to satisfy all needs. On occasions, users found feature and optimizations that could be incorporated into `grep`, such as the GNU extensions found in GNU `grep` [GNU 2002], and on others, they found new approaches to pattern matching that fundamentally changed the behaviour of `grep`, such as context `grep` [Clarke and Cormack 1996], structured `grep` [Jaakkola and Kilpeläinen 1996], nondeterministic reverse `grep` [Navarro 2001], and approximate `grep` [Wu and Manber 1992a]. This paper surveys all the major general-purpose global regular

expression print tools—variants of `grep` or `grep`-like tools. For each tool, we present a brief overview of features and give a usage example.

The rest of the paper is organized as follows. Section II presents a brief history of `grep`. Section III gives an overview of various `grep` tools. Section IV concludes the paper with our observations.

II. History of `grep`

Glantz did some of the earliest works on automated pattern matching [Glantz 1957]. Thompson was the first to implement `grep`—pattern matching based on regular expressions [Thompson 1968]. His implementation was based on building a nondeterministic finite automaton (NFA) for the regular expression. In 1976, Alfred Aho made a new version of `grep` called `egrep` (enhanced `grep`). Hume later showed that `egrep` was faster than `grep` for simpler patterns because it used deterministic finite automata (DFA) but slower for longer patterns because of the setup time required to build a complete DFA [Hume 1988]. `Egrep` was later enhanced by using lazy evaluation, which took zero setup time and just one additional test in the inner loop. Currently, `egrep` stands for extended `grep` because it supports extended regular expressions [IEEE and Open Group 2003a], as opposed to the default `grep` patterns which are basic regular expressions [IEEE and Open Group 2003a].

There have been several advances in pattern matching algorithms. Detailed examination of algorithms and their implementations are given in [Aho 1990] and [Hume and Sunday 1991]. Advances in approximate pattern matching are surveyed in [Navarro 2000]. The latest GNU `grep` implementation is based on the algorithm suggested in [Hume and Sunday 1991].

III. Global Regular Expression Print Tools

1. UNIX Grep Family: `grep`, `egrep`, and `fgrep`

The UNIX `grep` utility [IEEE and Open Group 2003b] searches the input text files, line by line, and prints each line the matches (or, optionally, does not match) the supplied pattern. By default, the pattern is treated as a basic regular expression [IEEE and Open Group 2003a]. Fixed-string `grep` (`fgrep`) is optimized for matching strings, and is equivalent to “`grep -F`”. Extended `grep` (`egrep`) includes additional operators for regular expressions, such as the disjunction operator ‘|’ that acts as *or*, and is equivalent to “`grep -E`”.

Example. To print lines containing either the string “quality” or the string “qualities”:

```
grep -E 'qualit(y|ies)' *
```

2. GNU Grep Family: `grep`, `egrep`, and `fgrep`

The GNU `grep` utility [GNU 2002] provides two advantages over the UNIX `grep`. Firstly, it is based on a fast-laze state deterministic matcher hybridized with a Boyer-Moore-Gosper search

for a fixed string that eliminates impossible text from being considered by the full regular expression matcher without having to look at every character. The result is usually many times faster than the UNIX `grep` and `egrep`. Secondly, GNU `grep` includes an extended set of options such as printing a number of lines before and after the matching line, colouring the match, and recursively traversing directories. GNU `grep` is an open-source project with a long list of authors. As a result, features from other variants of `grep` are constantly being incorporated into `grep`. For instance, the latest GNU `grep` supports the ‘-P’ option that causes the patterns to be treated as Perl regular expressions.

Example. To print lines containing either the string “quality” or the string “qualities”, as well as two lines before and after the matching line, and colouring all occurrences of the matched words:

```
grep -E -C 2 -colour 'qualit(y|ies)' *
```

3. Approximate Grep: `agrep`

Approximate `grep` [Wu and Manber 1992a] is a fuzzy string searching tool. It was developed as the search engine for Global Implicit Search (GLIMPSE) tool for searching and indexing whole file systems, which is part of the HARVEST Information Discovery and Access System. `Agrep` selects the best-suited algorithm for the current query from a collection of the known fastest (built-in) string searching algorithms. These algorithms include a variation of the Boyer-Moore algorithm and a set of novel algorithms, namely `bitap`, `mgrep`, `amonkey`, and `mmonkey` [Wu and Manber 1992a, Wu and Manber 1992b].

`Agrep` has three new features. Firstly, it allows for approximate string matching where the number of mismatched characters can be specified. Secondly, `agrep` is record oriented rather than line oriented. The user can define records by specifying a pattern that marks the beginning of a new record. Records can overlap and nest. Thirdly, `agrep` allows multiple pattern search using the logical AND and OR operators.

`Agrep` lacks many of the useful options that GNU `grep` provides, such as limiting the number of matches, displaying context when text is not structured, colouring the matches. It also imposes restriction on the length of the pattern when approximate matching is used.

Example. To print mail messages containing both of the two keywords “good” and “pizza”:

```
agrep -d '^From ' 'good;pizza' mbox
```

4. Context Grep: `cgrep`

The original idea behind context `grep` was to output the context in which a match is found by outputting preceding and following lines—a functionality that has already been incorporated into GNU `grep`.

The most remarkable `cgrep` implementation is due to [Clarke and Cormack 1996]. It is part of the MultiText information retrieval system and was developed to search structured texts were the

line boundary does not present useful partitioning of the text. Cgrep treats the new line character as an ordinary character, permitting the search pattern to span multiple lines, as well as permitting multiple matches within the same line. Cgrep prints substrings that match the pattern. The matches may overlap, but may not nest. Clarke and Cormack implemented the shortest-match principle in cgrep and they argue, with supportive examples, that this principle is more useful than the longest-match principle used in grep when treating text as a continuous stream [Clarke and Cormack 1997]. Cgrep supports union and intersection of matches, creating macros, and defining universes over the input to search within.

Cgrep does not use specialized algorithm for different types of patterns. For that reason, its performance is often inferior to other grep tools. However, it can handle patterns that are very difficult to express in the other tools to the extent that their performance was not compared.

Example. To extract mail messages containing the word vegetarian:

```
cgrep -U '^From .*(^From |>)' 'vegetarian' mbox
```

5. Structured Grep: sgrep

Structured grep [Jaakkola and Kilpeläinen 1996] was developed to search highly structured files, such as source-code, mail folders, news folders, HTML and SGML files, and so on. The file is divided into regions using delimiters, such as SGML tags. Regions can overlap and nest.

Sgrep does not use regular expressions; instead, it introduces its own language for indexing and querying. This language, while powerful and expressive, is difficult to learn and to express. Macro definition can simplify some of the common tasks, but only to a limited extent. The power of the sgrep query language is most evident when making complex queries on SGML like tagged documents.

Example. To extract the outer most block (define in C-style by '{' and '}') that contains the words “sort” or “nest” in the “eval.c” file:

```
sgrep 'outer("{ " .. "}" containing ("sort" or "nest"))' eval.c
```

Example. Extract titles and names of all HTML documents that contain links to www.cs.helsinki.fi:

```
sgrep -o"%f:%r\n" '(HTML_TITLE in (start .. end containing  
(HTML_HREF containing "www.cs.helsinki.fi")))' *.html
```

The same query with macros expanded:

```
((( ( "<TITLE>" or ( ("<TITLE " or "<TITLE\t" or "<TITLE\n" .. ">"))
.. ( "</TITLE>" ) )) in (start .. end containing (((
( " " or "\t" or "\n" or "\r") ___ ">" in (inner(("<" not
in ("</" or "<!" or "<?" )) .. ">" ) extracting
(("<" not in ("</" or "<!" or "<?" )) ___ ( " "
or "\t" or "\n" or "\r") or ">" ) in
inner(("<" not in ("</" or "<!"
or "<?" )) .. ">" ) ))) containing "HREF="
._ ( ( " " or "\t" or "\n" or "\r") or ">")) containing
"www.cs.helsinki.fi"))
```

6. Nondeterministic Reverse Grep: nrgrep

Nondeterministic reverse grep [Navarro 2001] is the newest addition to the grep tools. It was developed to facilitate searching natural language text. Nrgrep is the first pattern matching tool that uses the bit-parallel simulation of a nondeterministic suffix automaton [Navarro and Raffinot 1998]. At the high level, nrgrep provides similar functionality as agrep. However, its implementation is completely different. Its use of a single and uniform algorithmic concept, as opposed multiple specialized algorithms as in agrep and GNU grep, allows it to perform simple, sophisticated, and approximate pattern matching with an efficiency that degrades smoothly as the complexity of the pattern increases. Nrgrep also incorporates some of the useful options found in GNU grep such as

Nrgrep classifies patterns into three classes: simple patterns, extended patterns, and regular expressions. Simple patterns allow the usage of character classes in fixed strings. Extended patterns allow some character classes to be optional or repetitive. Regular expressions are the most general and allow for inclusion of empty strings, concatenation, union, and repetition. This classification is used in constructing optimized version of the algorithm for each class. Further, it allows for subpattern optimization. The result is a tool with comparable performance to grep and agrep for short and exact patterns and a noticeably superior performance for long patterns and for approximate patterns.

One of the limitation of nrgrep is fixed-size buffered processing of files. The size of the buffer is specified in advance and the all each record must fit in the buffer for tool to produce accurate results.

Example. To find all occurrences of the word “algorithm” including at the beginning of a sentence and allow for up to 3 deletion and substitution errors in the file readme.txt:

```
nrgrep -k 3ds '[Aa]lgorithm' readme.txt
```

7. Perl Regular Expressions

Perl regular expressions [Wall 2004] are the most expressive in the of the grep tools. In addition to the standard regular expression features, they include operators for shortest matches, define an extended set of character classes, introduce new positional matchers (e.g., end of word), allow

references for subpattern matches, and can match across new line characters. However, multi-line searches without buffering are nontrivial and are limited in flexibility. Perl's regular expression matching uses heuristics to select an algorithm that is appropriate to the search pattern.

We did not find literature comparing the efficiency of Perl's processing of regular expressions to other grep tools.

Example. To extract all the links (hyper-references) from the file `index.html`:

```
perl -wn -e 'print $1 ."\n" if /<a href=\"?(.*?)\"?>/' \
index.html
```

IV. Conclusion

This paper presents a survey of global regular expression print tools, a set of tools that mimic the behaviour of the `ed` command `g/re/p`. After a brief history on the origins of `grep`, we give an overview of all the major `grep` tools: the UNIX `grep` family, the GNU `grep` family, `agrep`, `cgrep`, `sgrep`, `ngrep`, and Perl regular expressions. For each tool, we presented, where appropriate, the motivation, main features, the main algorithms used, limitations, and an example. We observe in this list of tools that GNU `grep` is the most feature-rich tool that preserves the simplicity of UNIX `grep`, incorporates the power of Perl regular expression, and provides an extended set of useful options. While `agrep` represents the pioneer works in introducing approximate matching to the `grep` world, `ngrep` represents the state-of-the-art in approximate pattern matching. `Cgrep` is probably the most convenient tool for searching beyond the line boundaries. Lastly, `sgrep` is the tool of choice for complex searches on structured files.

References

- [Aho 1990] A.V. Aho. 1990. Algorithms for finding patterns in strings. In Handbook of Theoretical Computer Science . *Algorithms and Complexity*, Vol. A, ed. by J. van Leeuwen, Elsevier, Amsterdam, The Netherlands, pp. 255-300.
- [Boyer and Moore 1977] Robert S. Boyer and J. Strother Moore. 1977. A fast string searching algorithm. *Communications of the ACM*, Vol. 20, No. 10, pp. 762–772.
- [Clarke and Cormack 1996] Charles L. A. Clarke and Gordon V. Cormack. 1996. Context `grep`. *Technical Report CS-96-41*, School of Computer Science, University of Waterloo, Ontario, Canada.
- [Clarke and Cormack 1997] Charles L. A. Clarke and Gordon V. Cormack. 1997. On the Use of Regular Expressions for Searching Text. *ACM Transactions on Programming Languages and Systems*, Vol. 19, No. 3, pp. 413–426.
- [Glantz 1957] Herbert T. Glantz 1957. On the recognition of information with a digital computer. *Journal of the ACM*, Vol. 4, No. 2, pp. 178–188.

- [GNU 2002] GNU. 2002. GNU grep. WWW: <http://www.gnu.org/software/grep/grep.html>. Last accessed March 2, 2004.
- [Goedelbecker 1995] Eric Goedelbecker. 1995. Using grep. *Linux Journal*, Issue 18. WWW: <http://www.linuxjournal.com/article.php?sid=1149>. Last accessed March 2, 2004.
- [Hume 1988] Andrew Hume. 1988. A tale of two greps. *Software – Practice and Experience*, Vol. 18, No. 11, pp. 1063–1072
- [Hume and Sunday 1991] Andrew Hume, Daniel Sunday. 1991. Fast string searching. *Software – Practice and Experience*, Vol. 21, No. 11, pp. 1221–1248.
- [IEEE and Open Group 2003a] The IEEE and The Open Group. 2003. Regular Expressions. *The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2003 Edition, Base Definitions Volume*.
- [IEEE and Open Group 2003b] The IEEE and The Open Group. 2003. Utilities: grep. *The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2003 Edition, Shells and Utilities Volume*.
- [Jaakkola and Kilpeläinen 1996] Jani Jaakkola, and Pekka Kilpeläinen. 1996. Using sgrep for querying structured text files. *Technical Report C-1996-83*, Department of Computer Science, University of Helsinki.
- [Navarro 2000] G. Navarro. 2000. A guided tour to approximate string matching. *ACM Computing Surveys*, Vol. 33, No. 1, pp. 31–88.
- [Navarro 2001] G. Navarro. 2001. NR-grep: a fast and flexible pattern-matching tool. *Software – Practice and Experience*, Vol. 31, No. 13, pp. 1265–1312.
- [Navarro and Raffinot 1998] G. Navarro and M. Raffinot. 1998. A bit-parallel approach to suffix automata: Fast extended search matching. In 9th International Symposium on Combinatorial Pattern Matching (CPM’98), LNCS 1448, pp 14–33.
- [Sim 1998] Susan Elliott Sim. 1998. Supporting Multiple Program Comprehension Strategies During Software Maintenance. *Masters Thesis*, Department of Computer Science, University of Toronto.
- [Singer and Lethbridge 1997] Janice Singer and Timothy C. Lethbridge. 1997. What’s so great about ‘grep’? Implications for program comprehension tools. WWW: <http://wwwsel.iit.nrc.ca/~singer/grep/greptxt.html>. Last accessed March 2, 2004.
- [Thompson 1968] Ken Thompson. 1968. Regular expression search algorithm. *Communications of the ACM*, Vol. 11, No. 6, pp. 419–422.

- [Wall 2004] Larry Wall. 2004. Perl regular expressions. *Perl Manpage, Perl 5.8.0 Documentation*. WWW: <http://www.perldoc.com/perl5.8.0/pod/perlre.html>. Last accessed March 2, 2004.
- [Wu and Manber 1992a] S. Wu and U. Manber. 1992. AGREP- a fast approximate pattern-matching tool. In *Proceedings of the USENIX Winter 1992 Technical Conference*, pp. 153–162.
- [Wu and Manber 1992b] S. Wu and U. Manber. 1992. Fast text searching allowing errors. *Communications of the ACM*, Vol. 35, No. 10, pp. 83–91.