

Regular Expressions on the Web

Renáta Hodován, Zoltán Herczeg and Ákos Kiss*

University of Szeged, Department of Software Engineering

Honvéd tér 6., 6722 Szeged, Hungary

Email: {hodovan,zherczeg,akiss}@inf.u-szeged.hu

Abstract—On the web, static pages fade into the past; web sites use server and client-side scripting techniques to improve the user experience. For client-side scripting, JavaScript is by far the most commonly used dynamic language. In these scripts, regular expressions are widely used for several purposes, e.g., for text filtering and form validation.

In this paper, we take a closer look at regular expressions on the web. We investigate historical data and determine the trends of the use of regular expressions over time on various web sites. Then, we also take a snapshot of the current status. We find that although regular expressions are being used more and more often, their use is highly repetitive: on the most popular web sites only 4% of the regular expressions are unique. Based on this result, we show that web browsers can cut down regular expression compilation time to the third by applying caching techniques.

Index Terms—Computer languages, pattern matching, history

I. INTRODUCTION

Nowadays, web is one of the most popular application platforms [1]. Static web pages fade into the past; more and more web sites use advanced techniques like server and client-side scripting, and AJAX to improve the user experience. For client-side scripting, JavaScript is by far the most commonly used dynamic language [2], being supported by all major browsers. Not surprisingly, the optimization of JavaScript execution is a hot research topic [3], [4].

An interesting part of JavaScript, which deserves to be discussed separately, are regular expressions. They are widely used on the web for several purposes, e.g., for text filtering and form validation. Thus, in this paper we take a closer look at them: we investigate historical data and we also take a snapshot of the current status. The first contribution of this paper is the study on the use of regular expressions over time on various web sites. The trends show that although regular expressions are being used more and more often, their use is becoming highly repetitive: on the most popular web sites only 4% of the regular expressions are unique. The second contribution of this paper is the exploitation of this result. We show that web browsers can cut down regular expression compilation time to the third by applying caching techniques.

The remainder of this paper is organized as follows: in Section II we present our study on the past and present of regular expressions on the web. In Section III, we show how browsers can benefit from the results of the study. Related work is discussed in Section IV, and the paper ends with conclusions and future work in Section V.

II. STUDYING REGULAR EXPRESSIONS

A. Methodology

To study the use of regular expressions on the web, a tool is needed which is capable of recording the regular expressions that are seen on a web page and the URLs that are visited during a browsing session. We chose QtTestBrowser as the basis of our recording tool, the example browser of the Qt port of WebKit [5], a popular browser engine used in several desktop and mobile browsers as well. The modified QtTestBrowser dumps each parsed regular expression into a file, accompanied with the URLs of its containing web page. Having the data in a file, further analysis can work offline and does not have to deal with the ever-changing nature of the web.

B. Historical Data

Historical data has been collected by visiting pages archived by the Wayback Machine of the Internet Archive [6]. The archive is far from complete (which heavily limits our choices), but we found some web sites that were and are still well-known and popular, and have been archived to a great depth (following links through several steps from the main page): www.adobe.com, www.weather.com, and money.cnn.com. About 100 pages were visited each year from 2000 to 2008 (the Machine does not publish any information for 2009), mimicking usual browsing sessions. We also measured the present state of these sites.

Figure 1 shows how the number of regular expressions changed on the selected web pages over the years. The data is normalized to the present to help seeing the trends. The number of regular expressions parsed during the 2010 browsing session is 72,830 for www.adobe.com, 121,055 for www.weather.com, and 32,097 for money.cnn.com. The chart exposes that the trends are basically the same on all web sites: in the last few years (starting from 2006-2008) the use of regular expressions increased dramatically.

Figure 2 shows how the uniqueness of regular expressions changed in the same time frame. For each year, we show the ratio of the unique patterns compared to the total number of regular expressions parsed in the browsing session of that year. This second chart shows completely different trends. Between 2000 and 2004, the ratio was quite high, since the web sites used only a few regular expressions, and most of them were unique. But as the total number of patterns increased, the ratio of unique expressions fell below 3%.

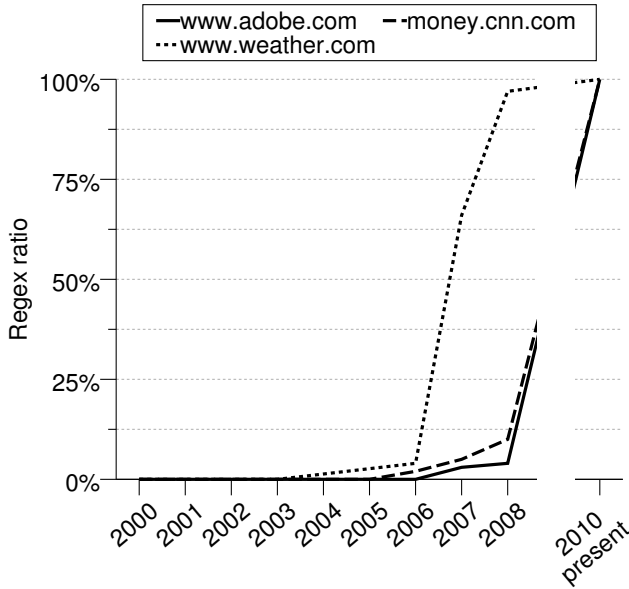


Fig. 1. Trends of the use of regular expressions.

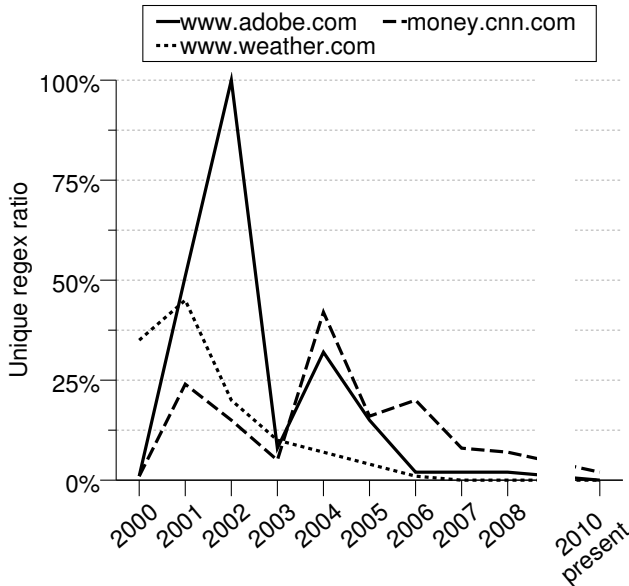


Fig. 2. Ratio of unique regular expressions.

C. The Present of Regular Expressions on the Web

Data on the current use of regular expressions is easier to collect. To gather realistic data, we visited the most popular sites of four categories that are considered being responsible for a considerable amount of web traffic: community [7] (also known as social networking), news [8], torrent [9], and adult [10]. Table I shows the number of regular expressions parsed during the browsing sessions, and the number of unique patterns as well. The results match the trends shown in

TABLE I
NUMBER OF REGULAR EXPRESSIONS ON POPULAR WEB SITES.

	Community	News	Torrent	Adult
Total	50243	63165	22605	46591
Unique	1529	1972	889	700
	3.04%	3.12%	3.93%	1.50%

TABLE II
THE MOST FREQUENTLY PARSED REGULAR EXPRESSIONS.

Regular expression	Parsed	Ratio	No. of sites	No. of URLs
$(^ \backslash\.) (\backslash\. \$)$	27997	17.4%	61	170
=	12937	8.0%	11	15
lytebox	2074	1.3%	1	1

TABLE III
REGULAR EXPRESSIONS USED ON MOST WEB SITES.

Regular expression	No. of sites	No. of URLs	Parsed
\s+	111	438	1689
webkit	91	348	968
opera	79	345	1463
msie	79	315	789

Figure 2. In all four categories, the ratios of unique regular expressions are below 4%.

Some more interesting findings are shown in the following two tables. Table II shows the regular expressions parsed most frequently during the browsing sessions. The first column shows a regular expression, the second column gives how many times the regular expression engine had to parse that pattern, and the third column gives its ratio to the total number of parsed patterns. The fourth and fifth columns show the number of web sites (domains) and distinct URLs (pages) on which the pattern was found, respectively. Interestingly, a very basic pattern (`lytebox`) came to the third place, even though it is used on a single page. However, on that page, the pattern was compiled 2074 times.

If we sort the regular expressions by the number of web sites on which they are used, we get a different order. This ranking is shown in Table III.

III. EXPERIMENTAL RESULTS WITH CACHING

The result on the ratio of unique patterns motivated us to experiment with caching, since regular expression engines usually first parse a pattern and then build an internal representation as a result. It can be a data structure or even a just-in-time compiled matching algorithm. This internal representation is used afterwards to do the actual matching.

We implemented a fully associative cache into QtTest-Browser to store pattern strings and references to their compiled forms. The cache can be configured with different cache sizes and to use either the Round Robin or the Least Recently Used caching policy [11].

We re-played the browsing sessions that have been recorded during the study presented in Section II-C and measured the efficiency of various cache configurations. Table IV shows the

TABLE IV
HIT RATIO OF THE CACHE.

Policy	Size	Community	News	Torrent	Adult
LRU	64	70.6%	54.1%	68.0%	23.8%
	128	90.0%	79.3%	84.7%	93.7%
	256	94.7%	90.0%	92.4%	97.1%
RR	64	69.3%	54.3%	67.6%	23.4%
	128	88.7%	78.3%	84.4%	92.7%
	256	94.1%	89.5%	91.0%	95.9%

TABLE V
THE EFFECT OF CACHING ON REGEX COMPILATION TIME.

Policy	Size	Community	News	Torrent	Adult
None		378.7 ms	193.9 ms	186.4 ms	212.3 ms
LRU	64	227.8 ms (60.2%)	132.5 ms (68.3%)	96.1 ms (51.6%)	166.4 ms (78.4%)
	128	146.5 ms (38.7%)	93.9 ms (48.4%)	63.4 ms (34.0%)	87.2 ms (41.1%)
	256	123.0 ms (32.5%)	72.3 ms (37.3%)	50.1 ms (26.9%)	81.2 ms (38.2%)
RR	64	248.2 ms (65.5%)	135.3 ms (69.8%)	102.4 ms (54.9%)	174.3 ms (82.1%)
	128	157.8 ms (41.7%)	95.9 ms (49.5%)	66.9 ms (35.9%)	89.7 ms (42.3%)
	256	130.7 ms (34.5%)	72.6 ms (37.4%)	54.1 ms (29.0%)	83.0 ms (39.1%)

hit ratios, i.e., how often was a pattern and its compiled form found in the cache.

We should note here, that the caching solution is not limited to a single page but the cache can contain patterns from previous pages and generate hits later. This is especially useful, if the user visits several pages on the same site, probably using the same JavaScript libraries with the same regular expressions.

Table V presents the effect of caching on the performance of regular expression parsing and compilation. As the table shows, caching can cut down regular expression parsing and compilation time to as low as 27% (with LRU caching policy and 256 cache entries), which equals to a 3-fold speedup.

IV. RELATED WORK

Regular sets were invented by Stephen Kleene [12] in the 1950s. Kleene’s theory was implemented by Ken Thompson [13] a decade later. He employed Deterministic Finite Automata (DFA) for fast pattern matching and introduced regular expressions for his text editor called *ed*. The advantage of DFA based implementation is its linear runtime. Henry Spencer [14] took a different path, and his regular expression library used a Nondeterministic Finite Automata (NFA) based implementation. Although his implementation was recursive, it offered greater flexibility than a DFA-based implementation. The PERL programming language chose this implementation for pattern matching.

Since JavaScript uses PERL-style regular expressions, all current browsers employ a backtracking-based regular expression engine, even though they can be very slow for special patterns, e.g., *a?.a?a...a*. Google tried to bring the NFA and DFA-based implementations closer in RE2 [15].

The idea of caching the compiled form of regular expressions already emerged in some systems. The .NET framework caches the last 15 regular expressions [16] for static method calls. The cache employs Least Recently Used caching policy.

The RegexpKit Framework [17], written in Objective-C, has a 4-way set associative cache, where each line contains 13 regular expressions. The line size can be changed at compile time, and should be a prime number. Caching is based on the hash code calculated for the pattern strings. The hash function is tied to the string hash implementation of the Core Foundation library [18].

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the result of a study on the use of regular expressions on the web. Our investigations revealed that the number of regular expressions on web pages and those parsed during web browsing increased dramatically in the last few years. Parallely, the uniqueness of the patterns fell: currently only 4% of the regular expressions used on the most popular web sites are unique. Based on this result we experimented with caching regular expressions in a test browser and found that regular expression parsing and compilation time can be cut down to third by choosing the right cache size and caching policy.

As the trends show, regular expressions gain importance on the web, improvement of regular expression engines get into the focus of browser developers. However, realistic regular expression benchmarks are hard to find. Thus, we plan to release our recorded browsing sessions to the public to make real test data available. Moreover, in the future we plan to perform more analysis on the collected data to see how the complexity of regular expressions changed over time and what the typical, most frequently used pattern features are. Finally, we also plan to evaluate the effect of regular expression caching on memory consumption as well.

REFERENCES

- [1] B. A. Huberman and L. A. Adamic, “Growth dynamics of the world-wide web,” *Nature*, vol. 401, p. 131, Sep. 1999.

- [2] Z. Herczeg, G. Lóki, T. Szirbucz, and A. Kiss, "Guidelines for JavaScript programs: Are they still necessary?" in *Proceedings of the 11th Symposium on Programming Languages and Software Tools (SPLST'09) and 7th Nordic Workshop on Model Driven Software Engineering (NW-MODE'09)*. Tampere, Finland: Tampere University of Technology, Aug. 26–28, 2009, pp. 59–71.
- [3] M. Stachowiak, "Introducing SquirrelFish Extreme," Sep. 2008, <http://webkit.org/blog/214/introducing-squirrelfish-extreme/>.
- [4] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. R. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Franz, "Trace-based just-in-time type specialization for dynamic languages," in *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (PLDI'09)*, Dublin, Ireland, Jun. 2009, pp. 465–478.
- [5] "The WebKit open source project," <http://webkit.org/>.
- [6] Internet Archive, "Wayback machine," <http://www.archive.org/>.
- [7] Wikipedia, "List of social networking websites," http://en.wikipedia.org/wiki/List_of_social_networking_websites.
- [8] Net Top 20, "News top 20: The pick of the best news sites on the net," <http://news.nettop20.com/>.
- [9] eBizMBA, "Top 15 most popular torrent websites," <http://www.ebizmba.com/articles/torrent-websites>.
- [10] Alexa Internet, Inc, "Top sites by category: Adult," <http://www.alexa.com/topsites/category/Top/Adult>.
- [11] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Prentice Hall Professional Technical Reference, 1973, ISBN:0136378684.
- [12] S. Kleene, "Representation of events in nerve nets and finite automata," *Automata Studies*, vol. 34, pp. 3–42, 1956.
- [13] K. Thompson, "Regular expression search algorithm," *Communications of the ACM*, vol. 11, no. 6, pp. 419–422, Jun. 1968.
- [14] H. Spencer, *Software Solutions In C*. Academic Press, 1994, ch. 3 – A Regular-Expression Matcher.
- [15] R. Cox, "RE2 regular expression engine," <http://code.google.com/p/re2/>.
- [16] Microsoft Corporation, ".NET regex caching," <http://msdn.microsoft.com/en-us/library/8zbs0h2f.aspx>.
- [17] J. Engelhart, "RegexKit framework," <http://regexkit.sourceforge.net/>.
- [18] Apple Inc., "Core foundation," <http://developer.apple.com/corefoundation/>.