Inference of Regular Expressions for Text Extraction from Examples: supplementary material

Alberto Bartoli[1], Andrea De Lorenzo[1], Eric Medvet[1], and Fabiano Tarlao[1]

[1]DIA, University of Trieste, Trieste, Italy

1 Introduction

This document contains supplementary material related to a research paper on the *inference of regular expressions for text extraction from examples*, whose abstract follows.

> A large class of entity extraction tasks from text that is either semistructured or fully unstructured may be addressed by regular expressions, because in many practical cases the relevant entities follow an underlying syntactical pattern and this pattern may be described by a regular expression. In this work we consider the long-standing problem of synthesizing such expressions automatically, based solely on examples of the desired behavior.
>
> We present the design and implementation of a system capable of addressing extraction tasks of realistic complexity. Our system is based on an evolutionary procedure carefully tailored to the specific needs of regular expression generation by examples. The procedure executes a search driven by a multiobjective optimization strategy aimed at simultaneously improving multiple performance indexes of candidate solutions while at the same time ensuring an adequate exploration of the huge solution space.
>
> We assess our proposal experimentally in great depth, on a number of challenging datasets. The accuracy of the obtained solutions seems to be adequate for practical usage and improves over earlier proposals significantly. Most importantly, our results are highly competitive even with respect to human operators. A prototype is available as a web application at `http://regex.inginf.units.it`.

2 Implementation

We implemented the method here proposed as a Java application[1] in which jobs are executed in parallel. We tuned the values for the parameters n_{job}, n_{pop}, n_{gen} and n_{stop} (the latter actually matters only in separate-and-conquer jobs) after exploratory experimentation and taking into account the abundant state of the art about GP. We set $n_{\text{job}} = 16$, $n_{\text{pop}} = 500$, $n_{\text{gen}} = 1000$ and $n_{\text{stop}} = 200$. For the web-based version, we set $n_{\text{job}} = 4$ to save computing resources and support an higher number of concurrent usages.

Our application includes two significant optimizations aimed at speeding up executions. We implemented a *caching* mechanism for reducing repeated evaluations of the same regular expression. This

[1]The source code is available on `https://github.com/MaLeLabTs/RegexGenerator`. A web-based version of the application is available on `http://regex.inginf.units.it`.

mechanism consists of a Java WeakHashMap[2], in which the key is the string transformation of an individual and the value is the set of extractions of that individual in the training data. When a given individual continues to exist across many generations, thus, the corresponding regular expression will be applied to all training data only once, rather than at each generation. Furthermore, this data structure is shared among all jobs in a search. It follows that the cached extractions may be exploited even for identical individuals that come into existence in different jobs We found experimentally that this caching mechanism allows to save roughly 50% of the computation time, on the average.

The second optimization comes into play in those problem instances in which the overall length of the examples is very large. This optimization consists in a procedure, that we call *shrinking*, aimed at reducing the overall length of the examples while not affecting the salient information available for learning the regular expression. The procedure transforms the set of examples E in another set of examples E_{shrunk} in such a way that long strings in E are transformed into much smaller strings in E_{shrunk} while preserving the content around each snippet (and dropping examples with $X_s = \emptyset$, i.e., without any snippet to be extracted). More in detail, for each example $(s, X_s) \in E$, a subset $E' \subset E_{\text{shrunk}}$ exists such that (i) each snippet in X_s occurs exactly once in E', (ii) each string s of an example in E' contains at most $10\ell(x_s)$ characters before and $10\ell(x_s)$ characters after each snippet x_s. Of course, the shrinking procedure leads to a loss of information in the training data. However, we found that this heuristic works well in practice, in particular, because large training data are usually highly unbalanced, with relatively few snippets to be extracted surrounded by very many characters that are not to be extracted. Indeed, the shrinking procedure makes it possible to handle datasets that could hardly be processed otherwise. We chose to trigger the procedure when $\sum_{(s,X_s)\in E} \ell(s) \geq 10^7$ characters, as training data of such size lead to a processing time that is too long to be practical.

[2] https://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html

3 Extraction tasks and datasets

Table 1 shows the entity types along with the corresponding regular expressions. Entity names should be self-explanatory (left column): Username corresponds to extracting only the username from Twitter citations (e.g., only MaleLabTs instead of @MaleLabTs); Email-ForTo corresponds to extracting email addresses appearing after the strings for: or to: (possibly capitalized). It seems fair to claim that these extraction tasks are quite challenging and representative of real world applications.

3.1 Learned regular expressions

Table 2 shows samples of the learned regular expressions (for configurations with $\sum_{(s,X_s)\in E} |X_s| = 100$). Since we executed 5 repetitions for each configuration, the table shows the expression $\hat{r}$ that was generated more times or, in case of tie, the one learned first. A domain-expert user would perhaps appreciate that, in several cases, $\hat{r}$ looks as if it had been generated by a human operator: e.g., for Twitter/Username*, BibTeX/Title*, or ReLIE-Web/HTTP-URL. This is an interesting result, because we did not included in our tool any mechanism aimed at favoring *readability*, other than simply penalizing long regular expressions (which was motivated by the need of avoiding bloating). It can also be seen that, for some extraction tasks, the generated regular expressions contain the or operator |: this occurred, in particular, in those problem instances in which the snippets to be extracted indeed exhibited different formats, e.g., CongressBill/date, ReLIE-Email/Phone-Number.

3.2 Comparison with human operators

In order to assess the ability of our method to compete with human operators, we executed an experiment using a web application which we crafted ad hoc.

The web app presented concise instructions about the experiment ("write a regular expression for extracting text portions which follow a pattern specified by examples") and then asked the user to indi-

Table 1: Regular expressions used to extract the entities of interest. Long regular expressions are split in two lines of the same row.

Entity type	Regular expression used to build examples of E_0
All-URL	(?:^|\b|\W)((?:(https?|s?ftps?|smb|mailto)://)(?:[a-zA-Z0-9][^\s:]+@)?(?:(?:[a-zA-Z0-9\-\.:]+\.[a-zA-Z]{2,3}) |(?:[0-9]{1,3}(?:\.[0-9]{1,3}){3}))(?::\d{1,5})?(?:[A-Za-z0-9.\-$+!*'(),˜_+%=&#/?@%;:]*)?(?<![.(]=]+))
HTTP-URL	(?:^|\b|\W)((?:https?://)(?:[a-zA-Z0-9][^\s:]+@)?(?:(?:[a-zA-Z0-9\-\.:]+\.[a-zA-Z]{2,3})|(?:[0-9]{1,3}(?:\. [0-9]{1,3}){3}))(?::\d{1,5})?(?:[A-Za-z0-9.\-$+!*'(),˜_+%=&#/?@%;:]*)?(?<![.(]=]+))
Phone-Number	(?:\W|^)((?:[0-9](?:␣|-|\.))?(?:\(?[0-9]{3}\)?|[0-9]{3})(?:\\|/|␣|-|\.)?(?:[0-9]{3}(?:␣|-|\.)[0-9]{4}))
href	(href=(?:["][^"]*["]|(?:['][^']*['])))
href-Content	((?<=href=")(?:[^"]*)(?=")|(?<=href=')(?:[^']*)(?='))
Hashtag+Citation	(?:\W|^)((?:@[A-Za-z0-9_]+)|(?:#[\w]+(?<!#\d{0,120})))
Username	(?:\W|^)@([A-Za-z0-9_]+)
IP	(?:\b|^)((?<!\.)(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]−[01]?[0-9][0-9]?) (?!\.))\b
MAC	((?:[0-9A-Fa-f]{2}[:-]){5}(?:[0-9A-Fa-f]{2})
Email-ForTo	(?<=(?:\s|-|^)(?:To|For|to|for):\s{0,200}(?:"?[^@"]{0,200}"?)?\s{0,200}[<"]?)[\w\.\-+]+@[\w\.\-]+ (?:=[>"])?
Email	(?:\b|^)((?:[a-zA-Z0-9_\-+][a-zA-Z0-9_\-\.+]*)@(?:(?:\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(?:(?:[a-zA-Z 0-9\-]+\.)+))(?:[a-zA-Z]{2,4}|[0-9]{1,3}))(?:\b|$)
Heading	(<h\d.*?>.*?</h\d>)
Heading-Content	<h\d[^<>]*?>(.*?)</h\d>
Date	((?:[12]?\d{1,3}\s*[\-/]\s*[012]?\d\s*[\-/]\s*[12]?\d{1,3})|(?:\d{1,2},\s*[A-Z][a-z]+\s+[12]\d{3}) |(?:[A-Z][a-z]+\s+\d{1,2},\s+[12]\d{3}))
Title	(?:\b|^)title␣*=␣*\{\{?(.*)\}?\}
Author	(?<=author␣{0,3}=␣{0,3}\{[^\n]{0,300}?)([A-Zd.{\d\\'"()][A-Z␣a-z{}\d\'"\-;()]*?)(?:(?:␣and␣)|(?:\}, \s*[a-z])|\n)
First-Author	(?<!:.{0,500})([\pL\w\-'"][\pL\w_\-'"␣]*,␣[A-Z.]+)

Table 2: Learned regular expressions with $\sum_E |X_s| = 100$. Long regular expressions are split in two or more lines of the same cell.

Extraction task	$\hat{r}$
ReLIE-Web/All-URL	http(?:\w*+\.*+[^\\>␣])*+\w++|http://([^␣]\w++/?+)++
ReLIE-Web/HTTP-URL	http([^␣][\w/]++)++
ReLIE-Email/Phone-Number	\(\w++\)␣\w++\-\w++|\w\w\w\-\w++\-\w++|\d++[^␣][^8]\d++[^␣]\d++
Cetinkaya-HTML/href	href="[^"]*+"|href[^>␣]++
Cetinkaya-HTML/href-Content*	(?<=href=")[^"]++|(?<=='')[\w\./]++[<']++
Cetinkaya-Web/All-URL	ftp:[^␣]*+
Twitter/Hashtag+Citation	[@#]\w++
Twitter/All-URL	\w++://\w\.co/\w++
Twitter/Username*	(?<=@)\w++
Log/IP	(\w++\.)++\w++
Log/MAC	\w++:(\w\w:)++\w++
Email-Headers/IP	(?<!@)(?<!\.)(?<!\w)(?:2012)*+6?+\d++\.\w*+\.\d++\.\d++
Email-Headers/Email-ForTo*	(?<=To:␣)[\w\.\-]++@[\w\.]++|(?<=\-\w\w\w\w\w\-To:␣<) [\+\w\-]++@[^>]++
NoProfit-HTML/Email	[^@]++[^<]++
Web-HTML/Heading	<\w\d(?:[^;][^>][^\d]*+)*+\w>
Web-HTML/Heading-Content*	(?<=>)(<*+[^<]++(?=<))++(?=</\w\d)
CongressBill/Date	\w++␣\w{1,2}+,␣[^>]*+\d++|\w++/\w\w/\w++| \w++\.\w\w\.\w++|\w++\-\w\w\-\w++| \d++,␣\w\w\w\w\w\w++␣\w\d++
BibTeX/Title*	(?<=title=\{)[^\}]++
BibTeX/Author*	(?<=␣)\w++,␣[\^␣]++(?:␣\w++(?=␣and))?+(?=␣and)| \w++,␣\w\w++(?<![year])(?:␣\w++(?!(?!␣and)␣))*+(?!␣and␣\w++␣)
Reference/First-Author*	(?<=\.␣)[^,]*+,␣(?:\w\.)++

cate the level of familiarity with regular expressions—one among novice, intermediate, and experienced. The web app then proposed a sequence of extraction tasks: for each task the web app showed a text on which the snippets to be extracted were highlighted; the user could write and modify a regular expression in an input field at will; the web app immediately highlighted the snippets actually extracted by the current expression along with the corresponding extraction mistakes (if any). The web app also showed the F-measure and the user was informed that a value of 100% meant a perfect score on the task. The user was not required to obtain a perfect F-measure before going to the next task—i.e., he could give up on a task. In the limit, he could also not write any regular expression for a task (*unanswered task*). The web app recorded, for each task and for each user, the authored regular expression and the overall time spent.

We included in the web app 9 of the extraction tasks. For each task, we chose exactly the E set we used while experimenting with our method for repetition 1 and $\sum_E |X_s| = 24$. We spread a link to the web app among CS graduate and undergraduate students of our University. Each user interacted with the web app autonomously and in an unconstrained enviroment—in particular, users were allowed to (and not explicitly instructed not to) refer to any knowledge base concerning regular expressions.

We gathered results from 73 users—60% novice, 20% intermediate, and 20% experienced. Several tasks were left unanswered: 42% for novice, 40% for intermediate, and 12% for experienced. The average time for solving the answered tasks was 16.1 min, 4.8 min, and 4.7 min, respectively. As a comparison, our method on the very same data required TtL = 10.4 min on the average.

The key finding is in Table 3, which shows the F-measure on $E^\star$ for each task. It can be seen that the F-measure obtained by our method is almost always greater than or equal to the one obtained by human users (on the average). The only exceptions are: the ReLIE-Email/Phone-Number task; the Web-HTML/Heading task, in which our method improves over novice users and only sligthly worse than intermediate users. We believe this result is remarkable and highly encouraging. Indeed, we are not aware of any proposal for automatic generation of regular expressions in which human operators were used as a baseline.

3.3 Comparison with other methods

We executed a comparison to 3 existing methods which can be used to learn text extractors: *Smart State Labeling DFA Learning (SSL-DFA)* [8], *FlashExtract* [6], and *GP-Regex* [1]. These methods are representative of the state of the art for learning syntactical patterns, but differ in the actual nature of the learned artifact: SSL-DFA produces Deterministic Finite Automata (DFA), FlashExtract produces extraction programs expressed in a specific language, and GP-Regex produces regular expressions.

3.3.1 SSL-DFA

We chose to consider SSL-DFA because the problem of DFA learning from examples is long established and several solutions have been proposed. In particular, SSL-DFA was developed a few years after a competition that was highly influential in the grammar learning community and outperformed (optimized versions of) the winners of the competition, on the same class of problems [4, 5] and even in the presence of noisy data. Another reason for our choice is because, in our experience at conferences and received reviews, we were often told that the problem considered in this paper is not interesting because DFA learning is a solved problem.

Three notable differences exist between the DFA learning scenario usually considered in literature and the text extraction scenario here considered. First, DFA learning methods are tailored to short strings defined over a binary alphabet, whereas in text extraction one needs to cope with longer strings defined over a much larger alphabet (in general, UTF-8). Second, DFA learning methods assume that the examples are drawn uniformly from the input space, whereas in text extraction this assumption is hard to be verified in practice. Third, learned DFA are intended as classifiers, i.e., given an input string, their outcome states if the whole string is accepted or re-

Table 3: F-measure for $\sum_E |X_s| = 24$ obtained by human operators (novice (SN), intermediate (SI), and experienced (SE)) and our approach (O).

	F-measure on E				F-measure on $E^\star$				Time spent/TtL			
Extraction task	SN	SI	SE	O	SN	SI	SE	O	SN	SI	SE	O
ReLIE-Web/All-URL	83.6	96.2	87.2	100.0	74.7	90.2	80.6	95.5	3	3	2	15
ReLIE-Web/HTTP-URL	86.4	89.3	91.3	95.8	77.3	83.0	76.6	82.3	20	8	9	11
ReLIE-Email/Phone-Number	88.5	96.3	100.0	100.0	70.2	84.7	91.0	34.6	8	6	6	11
Cetinkaya-HTML/href	91.9	99.6	99.8	100.0	91.6	98.8	98.8	100.0	4	2	3	12
Cetinkaya-Web/All-URL	96.9	100.0	100.0	100.0	95.2	98.3	98.6	99.0	80	2	1	3
Log/IP	91.3	100.0	100.0	100.0	91.0	100.0	100.0	100.0	5	1	2	2
Log/MAC	87.6	91.7	100.0	100.0	87.6	91.7	100.0	100.0	6	6	4	2
Web-HTML/Heading	87.6	99.5	99.6	100.0	82.3	90.9	95.6	90.0	7	5	2	30
BibTeX/Author*	71.3	55.2	88.7	100.0	64.6	50.1	81.4	90.3	20	10	10	8

jected by the DFA: some adaptation is hence needed in order to use a DFA for extraction. DFA learning and text extraction are thus quite different problems. An approach designed for the former problem may or may not perform well for the latter. Thus, it would not be surprising if approaches explicitly designed for text extraction outperformed SSL-DFA—our experimental evaluation suggests that this is indeed the case.

The SSL-DFA method described in [8] takes as input a triplet (S_A, S_R, n), where S_A is a set of strings which should be accepted, S_R is a set of strings which should be rejected, and n is the number of states of the DFA to be learned; the output is a DFA with n states. Given a problem instance $(E, E^\star)$, we obtain (S_A, S_R, n) as follows. Initially, we set $S_A = S_R = \emptyset$; then, for each example (s, X_s) in E, we add to S_A the textual content of each snippet in X_s and we add to S_R the textual content of each snippet in $\mathcal{X}_s \ominus X_s$; finally, we set $n = \max_{s \in S_A} \ell(s)$. With respect to the experimental setting considered in [8], the learning information available to SSL-DFA in our experiments is not balanced (i.e., in general, $|S_R| \gg |S_A|$): we verified experimentally that attempting to balance it by sampling the strings in S_R did not lead to better performance.

Concerning the actual extraction, we define the set $[\mathcal{X}_s]_d$ of the extractions obtained by applying a DFA d to a string s as the set of all the non-overlapping snippets of maximal length which are accepted by d. We implemented this method in C++ basing on the description in [8].

3.3.2 FlashExtract

FlashExtract is s a powerful and sophisticated framework for extracting multiple different fields automatically in semi-structured documents [6]. It consists of an inductive synthesis algorithm for synthesizing data extraction programs from few examples, in which programs are expressed in any underlying domain-specific language supporting a predefined algebra of few core operators. The cited work presents also a language designed to operate on text which perfectly fits the extraction problem considered in this paper. The findings of [6] resulted in a tool included in the Windows Powershell as the ConvertFrom-String command: we used this tool to perform the experiments.

The current FlashExtract implementation does not allow reusing a program induced by a given set of examples. Thus, in our experimentation the two phases of learning and testing were not separated: we invoked the tool by specifying as input the examples in E and the strings in $E^\star$; we obtained as output a set of substrings extracted from $E^\star$ based on the description in E (which we had to recast in the syntax required by the tool). In many cases the tool crashed, thereby preventing the extraction to actually complete. We highlighted these cases in the results.

3.3.3 GP-Regex

GP-Regex is the method we proposed in [1] and the base for the research here presented. The numerous differences between our method and GP-Regex were listed in the introduction. We emphasize again that in GP-Regex each example consists of a string and at most one single snippet to be extracted from that string. In order to build learning examples suitable for GP-Regex, we considered for each (s, X_s), only the leftmost snippet in X_s, if any.

It may be useful to remark that in [1] an experimental comparison was made against the approaches of [3,7] on two datasets previously used by the latter: the proposal in [1] exhibited better accuracy, even when the amount of available learning examples was smaller by more than order of magnitude. Moreover, the authors of [3,7] showed that their approaches exhibited performance similar to *Conditional Random Fields*.

3.3.4 Comparison results

We selected 7 extraction tasks including tasks with context and tasks in which snippets exhibit widely differing formats. We exercised all methods with the same experimental settings described in the main paper, thereby obtaining 105 problem instances—5 repetitions for each of the 21 different combinations of extraction task and number of snippets for learning.

Table 3.3.4 shows the results in terms of F-measure and TtL. The foremost outcome of this comparison is that our method clearly outperforms all the other three methods (except for ReLIE-Email/Phone-Number, discussed below).

The performance gap with both SSL-DFA and FlashExtract is substantial—at the expense of a much longer TtL, though. Concerning SSL-DFA, we believe this finding is interesting because both SSL-DFA and our method are based, broadly speaking, on evolutionary computation and SSL-DFA is representative of the state-of-the-art in its field. On the other hand, based on the previous considerations about the significant differences between typical settings for DFA learning and text extraction, we do not find this result particularly surprising either. Indeed, as pointed out earlier by different authors, benchmark problems for DFA learning from examples are not inspired by any real world application [4] and the applicability of the corresponding learning algorithms to other application domains is still largely unexplored [2].

Concerning FlashExtract, the fact that we obtain much better accuracy in all settings (with the only exception of ReLIE-Email/Phone-Number, discussed below) is also very interesting. We are not able to provide any principled interpretation for this result. We may only speculate that our approach is perhaps more suitable for coping with loosely structured or unstructured datasets than FlashExtract. We also noticed that, for many problem instances, the ConvertFrom-String tool crashed, thereby preventing the extraction to actually complete. For the extraction tasks for which at least one on 5 repetition completed without errors, Table 3.3.4 shows the performances (F-measure and TtL) averaged across the completed executions. In the other cases, we were not able to obtain any extraction program, neither splitting the testing set in small chunks: those cases are denoted with an en dash in the table.

Concerning GP-Regex, we should isolate two groups of extraction tasks: (i) those that requires either a context (Web-HTML/Heading-Content*) or the ability to learn widely differing patterns (CongressBill/Date), (ii) all the other tasks. The key observation is that our current proposal improves over GP-Regex in all cases (except for ReLIE-Email/Phone-Number), the improvement being substantial in case i. Indeed, our current proposal makes it possible to handle both Web-HTML/Heading-Content* and CongressBill/Date with good accuracy, while GP-Regex does not. It is also interesting to observe that in case ii GP-Regex provides much better accuracy than SSL-DFA and FlashExtract, while in case i GP-Regex is either comparable to those methods or worse.

Finally, concerning the ReLIE-Email/Phone-Number extraction task, we observe that this is the same task with a sort of anomalous behavior already discussed in the previous section. In particular, we remark that when executing our method on this task

Table 4: Results. TtL is expressed in minutes. Missing values for FlashExtract, denoted with −, correspond to problem instances for which no repetition completed successfully (see text).

		SSL-DFA		FlashExtract		GP-Regex		Our proposal	
Extraction task	$\sum_E \lvert X_s \rvert$	Fm	TtL	Fm	TtL	Fm	TtL	Fm	TtL
ReLIE-Web/All-URL	24	13.8	1	15.2	1	78.3	5	90.9	15
ReLIE-Web/All-URL	50	18.2	1	25.2	3	88.0	10	93.5	35
ReLIE-Web/All-URL	100	18.1	2	21.5	3	93.0	16	95.6	71
ReLIE-Email/Phone-Number	24	27.7	1	69.5	1	84.0	5	48.3	8
ReLIE-Email/Phone-Number	50	11.3	2	–	–	91.7	13	43.3	16
ReLIE-Email/Phone-Number	100	15.9	2	–	–	90.2	18	35.8	39
Cetinkaya-HTML/href	24	21.2	1	23.5	1	46.9	13	98.9	12
Cetinkaya-HTML/href	50	12.0	1	28.7	1	81.6	26	98.4	26
Cetinkaya-HTML/href	100	9.2	2	32.3	3	89.6	44	98.8	59
Cetinkaya-Web/All-URL	24	18.5	1	33.9	71	83.4	8	99.1	3
Cetinkaya-Web/All-URL	50	14.7	1	52.2	52	92.7	18	96.9	8
Cetinkaya-Web/All-URL	100	17.6	1	61.8	25	94.9	31	98.8	16
Twitter/Hashtag+Citation	24	14.5	1	–	–	94.8	3	99.6	3
Twitter/Hashtag+Citation	50	21.8	1	–	–	97.3	5	99.5	4
Twitter/Hashtag+Citation	100	28.4	1	–	–	100.0	8	99.6	7
Web-HTML/Heading-Content*	24	11.9	1	10.6	4	4.4	34	86.6	76
Web-HTML/Heading-Content*	50	11.9	2	10.2	3	5.0	196	91.8	168
Web-HTML/Heading-Content*	100	18.2	3	–	–	10.2	672	97.7	379
CongressBill/Date	24	37.4	1	–	–	29.8	361	64.1	30
CongressBill/Date	50	25.2	3	–	–	27.8	432	69.7	584
CongressBill/Date	100	46.5	4	–	–	38.0	386	70.7	513

with a learning ratio[3] LR $\approx 80\%$ we obtained $\approx 85\%$ F-measure on the testing data. We could not execute FlashExtract in those conditions because it always crashed: the only result that we could obtain is in Table 3.3.4, where F-measure (with very few examples available for learning) is 69%. The reason why GP-Regex happens to deliver better accuracy on this task is because it tends to overfit the snippets to be extracted more than the method here presented. As discussed in the previous section, processing this task with a very small LR value incurs in a poor representativeness of the text that is not to be extracted; as it turns out, thus, the slightly overfitting behavior exhibited by GP-Regex in this case turns out to be a pro.

References

[1] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47(12):72–80, Dec 2014.

[2] Josh Bongard and Hod Lipson. Active coevolutionary learning of deterministic finite automata. *The Journal of Machine Learning Research*, 6:1651–1678, 2005.

[3] Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1285–1294. ACM, 2011.

[4] Orlando Cicchello and Stefan C Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research*, 4:603–632, 2003.

[5] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, page 1–12. Springer, 1998.

[6] Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, page 55. ACM, 2014.

[7] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and HV Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics, 2008.

[8] Simon M Lucas and T Jeff Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

[3] *Learning ratio (LR)* is defined as the ratio between the number of snippets for learning and the number of snippets in the full extraction task E_0.