Report from Dagstuhl Seminar 24472

Regular Expressions: Matching and Indexing

Inge Li Gørtz*[1], Sebastian Maneth*[2], Gonzalo Navarro*[3], and Nicola Prezza*[4]

1 **Technical University of Denmark – Lyngby, DK.** inge@dtu.dk
2 **University of Bremen, DE.** maneth@uni-bremen.de
3 **University of Chile – Santiago de Chile, CL.** gnavarro@dcc.uchile.cl
4 **Ca' Foscari University of Venice, IT.** nicola.prezza@unive.it

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 24472 "Regular Expressions: Matching and Indexing".

Seminar November 17–22, 2024 – https://www.dagstuhl.de/24472
2012 ACM Subject Classification Theory of computation → Data structures design and analysis
Keywords and phrases finite automata, regular expressions, complex patterns, text indexing, graph matching and indexing
Digital Object Identifier 10.4230/DagRep.14.11.108

1 Executive Summary

Inge Li Gørtz
Sebastian Maneth
Gonzalo Navarro
Nicola Prezza

The Dagstuhl Seminar "Regular Expressions: Matching and Indexing" (24472) took place from November 17th to 22nd, 2024. The goal of this seminar was to bring together researchers from various research directions dealing with algorithmic aspects of regular expressions and finite automata. Regular expressions and finite automata lie at the foundations of Computer Science and have been used since the sixties in basic problems like compiler design. The key algorithmic challenge is regular expression matching, that is, efficiently identifying words of a regular language within a sequence or within the paths of a labeled graph. Over the years, there have been numerous algorithmic advances around the topic, while at the same time their applications have spread over too many different areas like information retrieval, databases, bioinformatics, security, and others, which not only make use of standard results but also pose new and challenging variants of the regular expression matching problem. The use of regular expressions has made its way even into current standards like SQL:2016 and SPARQL.

The seminar was meant to bring together expertise from communities involved with regular expressions that, despite working on closely-related topics, do not frequently meet: graph databases, compressed data structures, and streaming algorithms. The following specific points were addressed.

* Editor / Organizer

String Indexing for Complex Patterns. The classic and well-studied string indexing problem is to preprocess a string such that subsequent pattern-matching queries (locate all occurrences of the query string within the indexed strings) can be supported efficiently. The goal is to achieve a fast query time in terms of the length of the query string while keeping the memory needed for the index small. Several string indexes that support "complex" pattern matching queries have recently appeared. Examples include indexes that support pattern matching with wildcards, gaps, elastic degenerate, and approximate string matching. Such complex patterns may be viewed as special restricted cases of regular expressions. Thus, studying these in the context of regular expressions may lead to a more unified understanding of these and new research directions.

Advanced Regular Expression Operators. Nearly all existing methods for regular expression matching consider regular expressions constructed using the concatenation, union, and Kleene star operators. In this setting, the complexity of the problem is well understood, and upper bounds with near-matching conditional lower bounds are known. However, more "advanced" operators that allow increased expressiveness are widely used in practice. These include backreferences (matching a previously matched subexpression), intervals (matching a subexpression a given number of times), character classes (matching a character to a set of characters), variable length gaps (matching any string of a length within a specified interval). In contrast to the classic setting, regular expression matching with these operators is much less well-understood. While some of these are known to be much harder from a complexity-theoretic viewpoint (i.e., regular expression matching with backreferences is, in general, NP-hard), recent developments have led to new methods handling many practically essential variants of the problem. Examples include deterministic regular expressions, pattern matching with variables, pattern matching with variable length gaps, and elastic degenerate strings. This enables a new study area that is theoretically challenging and practically relevant.

Indexing Automata and Regular Expressions. The counterpart of the "string indexing for complex patterns" problem is to index/pre-process a regular expression to speed up tasks such as deciding membership of a string in its accepted language. An equivalent formulation of the problem, which is lately receiving much attention in the research community, is to index finite-state automata for *path queries*: given a string at query time, decide whether the string occurs in some walk on the automaton (equivalently, it is a substring of some string in the automaton's accepted language). While recent research has shown quadratic (size of the regular expression/automaton multiplied by the length of the query string) conditional lower bounds for the problem, particular classes of regular expressions and automata do admit efficient solutions: deterministic regular expressions, elastic degenerate strings, and Wheeler automata are examples of this kind. Even more interestingly, other lines of research showed that the problem can be parameterized, thus classifying *all* regular expressions and automata according to their *propensity* to support membership/path queries: examples of such parameters include the number of strings in the regular expression (equivalently, the number of union symbols and Kleene stars), the co-lexicographic width of an automaton, and the density (amount of nondeterminism) of a regular expression. These directions opened an exciting new research area of both theoretical and practical interest with applications including bioinformatics (with the problem of indexing large pan-genomic graphs for path queries) and graph databases.

Graph Databases and Regular Path Queries. Another flavor of regular expression matching arises in the context of graph databases, where the data consists of a labeled directed graph and queries involve different forms of subgraph pattern matching. The SPARQL

standard, as well as several other alternative graph query languages, support in particular "regular path queries (RPQs)", which are essentially regular expressions that are to be matched against the sequence of labels of graph paths. The basic strategy is to traverse the product graph between the database graph and the automaton of the regular expression. This leads to a quadratic time complexity, but other strategies based on graph traversals, pre-indexing the graph, or multiplication of sparse matrices, perform better in many practical cases. The most basic query aims at returning the endpoints of the paths found, but others ask for all the paths, the shortest paths, and so on. There has also been work on lower bounds on this problem, particularly relating it to the well-known AGM bounds that hold for matching a given subgraph shape. Strategies developed for regular expression matching on strings can be leveraged to solve RPQs, but this requires a cross-fertilization between the stringology and the database community in order to transfer the knowledge: most solutions implemented in actual graph database systems are just heuristics that have been shown to be clearly inferior to solutions grounded on more solid algorithmic foundations.

Regular Expression Matching on Streams. Another relevant research area, motivated by big data applications, is that of regular expression matching on data streams: pre-process a regular expression R so that, later, we can identify all suffixes $S[1, i]$ of an incoming stream S such that $S[j, i]$ matches R for some $j \leq i$, using space sub-linear in $|S|$ and $|R|$. This problem generalizes a seminal work by Porat and Porat (2009) on exact pattern matching and on the k-mismatches problem on streams. A subsequent prolific line of works improved the original bounds and extended these results to edit distance, to the dictionary problem, and to wildcard matching. Recent research has shown that the problem admits solutions for arbitrary regular expressions using space polynomial in the logarithm of the stream's length and in the number of union symbols and Kleene stars in the regular expression. This shows that parameterized approaches (as the ones for the problem of indexing automata and regular expressions) are indeed useful also in the streaming setting, and therefore strongly motivates an exchange of ideas between the two sub-fields.

The seminar fully satisfied our expectations. The 26 participants from 14 countries (Chile, Denmark, Finland, France, Germany, Great Britain, Israel, Italy, Japan, Netherlands, Poland, South Africa, Spain, and US) gave invited survey talks covering the state of the art in scientific fields related with the topics of the seminar. The talks presented works related with algorithms for regular expression matching (covering classic algorithms, extensions of regular expressions, indexing, streaming, and compressed data structures), regular expressions in graph databases, and practical tools used at the industrial scale.

We are really thankful to Schloss Dagstuhl for providing an extremely inspiring and professional environment. Scientific talks were interleaved with coffee breaks and cheese tastings which fostered engagements of the participants, and the evening "sessions" in the sauna, wine cellar, and music room offered a relaxed environment to continue chatting about research in a less formal environment.

2 Table of Contents

3 Overview of Talks

3.1 Algorithmic Techniques for Regular Expression Matching

Philip Bille (Technical University of Denmark – Lyngby, DK)

We survey the classic regular expression matching problem, that is, given a regular expression R and a string Q determine whether or not Q is in the language defined by R. We cover the algorithmic techniques and the history of the problem, from Thompson's classic algorithm from the 60'ties [C. ACM 1968] to the present day.

3.2 Bit-parallel Sequential Search for Regular Expressions

Gonzalo Navarro (University of Chile – Santiago de Chile, CL)

In this talk I overviewed bit-parallel techniques for complex pattern matching, going from simple strings to regular expressions, via some useful complex patterns of intermediate complexity. I showed how bit-parallelism allows easy and efficient simulation of complex patterns whose deterministic automaton has no simple structure, yet their nondeterministic structure is regular and allows a bit-parallel simulation. This enables not only efficient bit-parallel search of those complex patterns, but also simulating their suffix automata, such as to implement search algorithms that can skip text positions, which without bit-parallelism can only be done for simple strings. Further, I showed how approximate searching can be incorporated into those searches, so that there can be up to k differences between the matched text and some string in the language of the complex pattern. I finished with a nice application to natural language that exploits all the flexibility of this matching both at the intra-word and the inter-word level.

3.3 Regular Expressions in Information Extraction and Graph Databases

Wim Martens (Universität Bayreuth, DE)

I will give an overview to the use of regular expressions in two areas in databases: information extraction and graph databases. In information extraction, regular expressions play a central role in the document spanner framework. This framework aims at transforming text into a relation of spans, i.e., intervals of start and end positions in the text. Within the spanner framework, the class of regular spanners is particularly interesting, since it is based on regular expressions with capture variables. I will also briefly touch upon frequent sequence mining. In graph databases, regular expressions have been interesting since the beginning, since they form the foundation of regular path queries, which are the quintessential feature of graph database query languages. Recent developments in graph query languages, among which the development of Cypher and the standardization of GQL and SQL/PGQ, motivate new ways of evaluating regular path queries in practice, which raises many new research questions.

3.4 REmatch: theory and practice for evaluating regex and finding all matches

Cristian Riveros (PUC – Santiago de Chile, CL)

In this talk, I will present REmatch, a novel regex engine that always finds all matches. REmatch is based on document spanners, a formal framework for information extraction based on regular expressions and finite automata with capture variables. Given a document (i.e., a string), this framework allows for the definition of extraction tasks by regular expressions with variables in a denotational manner, namely, without depending on the leftmost longest semantics of regex or any evaluation strategy. I will present the main features of REmatch, such as its query language called REQL (RegEx Query Language), the use of variables, and multimatch capturing. Furthermore, I will overview the theory behind REmatch, such as regular expression with capture variables, variable-set automata, and its query evaluation algorithm, based on the theory of constant-delay enumeration algorithms.

3.5 Treating Regex as Database Queries

Dominik D. Freydenberger (Loughborough University, GB)

Most modern implementations of regular expressions have back-references, which express repetitions of submatches. These allow the definition of non-regular languages, like the copy language ww. The price for this gain in expressive power is that matching becomes NP-hard.

In this talk, I present a relational algebra on strings that allows us to adapt techniques from database theory to matching of regular expressions with back-references. It also provides us with a framework to combine these with parsing techniques.

3.6 Regular expression membership in small space

Pawel Gawrychowski (University of Wroclaw, PL)

Joint work of Bartłomiej Dudek, Pawel Gawrychowski, Garance Gourdel, Tatiana Starikovskaya
Main reference Bartlomiej Dudek, Pawel Gawrychowski, Garance Gourdel, Tatiana Starikovskaya: "Streaming Regular Expression Membership and Pattern Matching", in Proc. of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022, pp. 670–694, SIAM, 2022.
URL http://dx.doi.org/10.1137/1.9781611977073.30

An elegant formalism for describing scenarios where we need to process large amounts of string data arriving online is the streaming model. In this model, the input characters arrive one by one; we cannot go back to any previously seen characters and need to output the answer after receiving the next character. The primary goal is minimizing the amount of working space measured in bits, and the secondary goal is to optimize the time to process a character. It has been shown by Porat and Porat [FOCS 2009] that the classical exact

pattern matching problem can be solved in this model using only $O(\log n \log m)$ bits of space with high probability. I will briefly present the properties of the streaming model and an overview of their algorithm.

As pattern matching can be encoded as a regular expression, it is natural to consider, given a general regular expression R, checking which prefixes of the text belong to $L(R)$. While it is not difficult to prove that we cannot solve the general case in sublinear space, we can hope to obtain such a bound for restricted classes of regular expressions. In particular, it is natural to limit the number of strings appearing in the regular expression by d, and aim at obtaining space complexity that is polynomial in d and $\log n$ (where n is the length of the text). I will sketch some of the ideas used in designing such an algorithm.

3.7 Text Indexing in the context of Regular Expressions

Moshe Lewenstein (Bar-Ilan University – Ramat Gan, IL)

In Regular Expression indexing we are given a text T which we seek to index – preprocess a data structure on T – to answer queries q, a regular expression, to find locations in T where a substring begins that belongs to the language of q.

We show (a) conditional lower bounds using fine-grained complexity, (b) the algorithm of Baeza-Yates and (c) the algorithm of Gibney and Thankachan. We then consider restricted cases of regular expressions; (a) text indexing with don't cares in the pattern queries and (b) gapped indexing, where the pattern queries are of the form (p1,p2,a,b), and we seek all appearance of pairs p1 and p2 with the distance between p1 and p2 being in the range of a and b.

3.8 From text indexing to regular language indexing

Nicola Prezza (University of Venice, IT)

Since the invention of suffix sorting (in particular, of suffix trees) in the 70s, the problem of indexed pattern matching has been heavily studied in the literature. This problem has a natural language-theoretic interpretation: given a string S, build a (linear-space) data structure answering membership queries in the substring closure of S. This interpretation was recently made more interesting by several works showing that suffix sorting can be naturally extended to some nonlinear structures, notably labeled trees and de Bruijn graphs. This line of work culminated in the invention of Wheeler automata, a class of NFAs admitting efficient and elegant solutions to a large number of hard problems on automata (including membership). In this talk, I will first give an introduction to the rich theory of Wheeler automata and Wheeler languages. I will then show how these ideas can be generalized to arbitrary NFAs, comparing this solution to other existing approaches for indexing arbitrary regular languages.

3.9 Complexity Analysis of Regular Expression Matching Based on Backtracking

Yasuhiko Minamide (Institute of Science Tokyo, JP)

Until recently, regular expression matching has mostly been implemented using backtracking, which led to a denial-of-service vulnerability known as ReDoS. In ReDoS, matching does not complete in linear time and can take an excessive amount of time. In this talk, we will review previous works that detect such problematic regular expressions through ambiguity analysis of nondeterministic automata and growth rate analysis of string-to-tree transducers. For a given regular expression, it is possible to precisely determine the time complexity (order) of its matching with respect to the length of the input string.

3.10 Efficient Matching of Regular Expressions with Lookaround Assertions

Konstantinos Mamouras (Rice University – Houston, US)

Regular expressions can be extended with lookaround assertions, which are subdivided into lookahead and lookbehind assertions. These constructs are used to refine when a match for a pattern occurs in the input text based on the surrounding context. Current implementation techniques for lookaround involve backtracking search, which can give rise to running time that is super-linear in the length of input text. In this talk, we first present a formal mathematical semantics for lookaround, which complements the commonly used operational understanding of lookaround in terms of a backtracking implementation. This formal semantics allows us to establish several equational properties for simplifying lookaround assertions. Additionally, we propose a new algorithm for matching regular expressions with lookaround that has time complexity $O(m \cdot n)$, where m is the size of the regular expression and n is the length of the input text. The algorithm works by evaluating lookaround assertions in a bottom-up manner. It makes use of a notion of nondeterministic finite automata (NFAs), which we call oracle-NFAs. These automata are augmented with epsilon-transitions that are guarded by oracle queries that provide the truth values of lookaround assertions at every position in the text. We provide an implementation of our algorithm that incorporates three performance optimizations for reducing the work performed and memory used. We present an experimental comparison against PCRE and Java's regex library, which are state-of-the-art regex engines that support lookaround assertions. Our experimental results show that, in contrast to PCRE and Java, our implementation does not suffer from super-linear running time and is several times faster.

3.11 Subsequence Expressions with Gap Constraints

Markus L. Schmid (HU Berlin, DE)

A subsequence expression is a regular expression of the form $p = A^*x_1A^*x_2A^* \dots A^*x_mA^*$, where A is an alphabet and $x_i \in A$. Matching p to a string w means to search w for the subsequence $x_1x_2 \dots x_m$. A match can be formalised as an embedding $e : |p| - > |w|$, which then, for every $1 \leq i < j \leq |p|$, induces an (i, j)-gap, i.e., the substring of w that occurs strictly between $w[e(i)]$ and $w[e(j)]$. Gap constraints are constraints for the gaps induced by an embedding, e. g., "the (i, j)-gap should be no longer than 7", "the (i, j)-gap should not contain the symbol c", etc.

We investigate the problem of matching a subsequence expression to a string under the presence of gap constraints, where the gap constraints are given as regular languages. Our results cover hardness results as well as polynomial upper bounds and conditional lower bounds.

3.12 Evaluating Regular Path Queries on Compressed Adjacency Matrices

Gonzalo Navarro (University of Chile – Santiago de Chile, CL)

Joint work of Diego Arroyuelo, Adrián Gómez-Brandón, Gonzalo Navarro

Regular Path Queries (RPQs), which are essentially regular expressions to be matched against the labels of paths in labeled graphs, are at the core of graph database query languages like SPARQL and GQL. A way to solve RPQs is to translate them into a sequence of operations on the adjacency matrices of each label. We design and implement a Boolean algebra on sparse matrix representations and, as an application, use them to handle RPQs. Our baseline representation uses the same space and time as the previously most compact index for RPQs, outper- forming it on the hardest types of queries – those where both RPQ endpoints are unspecified. Our more succinct structure, based on k2 -trees, is 4 times smaller than any existing representation that handles RPQs. While slower, it still solves complex RPQs in a few seconds and slightly outperforms the smallest previous structure on the hardest RPQs. Our new sparse- matrix-based solutions dominate a good portion of the space/time tradeoff map, being outperformed only by representations that use much more space. They also implement an algebra of Boolean matrices that is of independent interest beyond solving RPQs.

3.13 Optimizing RPQs over a Compact Graph Representation

Adrián Gómez Brandón (University of Coruña, ES)

We propose techniques to evaluate regular path queries (RPQs) over labeled graphs (e.g., RDF). We apply a bit-parallel simulation of a Glushkov automaton representing the query over a Ring: a compact wavelet-tree-based index of the graph. To the best of our knowledge, our approach is the first to evaluate RPQs over a compact representation of such graphs, where we show the key advantages of using Glushkov automata in this setting. We introduce various optimizations, such as the ability to process several automaton states and graph nodes/labels simultaneously, and to accurately estimate relevant selectivities. Experiments show that our approach uses $3-5\times$ less space, and is over $5\times$ faster, on average, than the next best state-of-the-art system for evaluating RPQs.

3.14 McDag: An Index for Maximal common subsequences

Roberto Grossi (University of Pisa, IT)

Joint work of Roberto Grossi, Giovanni Buzzega, Alessio Conte, Giulia Punzi

Maximal Common Subsequences (MCSs) – the inclusion-maximal sequences of non-contiguous symbols shared by strings – are a natural extension of known methods like Longest Common Subsequences but have only recently gained attention. This talk presents McDag, an efficient graph-based tool to index MCSs on real genomic data, showing it can handle sequence pairs over 10,000 base pairs in minutes with minimal extra storage.

3.15 Parameterized linear-time algorithms for string matching to DAGs

Manuel Cáceres (Aalto University, FI)

Joint work of Manuel Cáceres, Massimo Equi, Paweł Gawrychowski, Veli Mäkinen, Jakub Radoszewski, Alexandru I. Tomescu.

In this talk I will present parameterized linear-time algorithms for SMLG (string matching to labeled graphs) on DAGs. Our algorithms run in time $O(k|G| + |P|)$. We obtain the result separately for a parameter capturing the topology of the DAG G, and for a parameter capturing the periodicity of the pattern P. Additionally, we present an algorithm running in time proportional to the number of prefix-incomparable matches, which is able to capture both parameters previously mentioned. To obtain the parameterization in the topological structure of G, we generalize in-trees, out-trees and funnels to the classes S_k, T_k and ST_k, respectively.

3.16 Regular Expression Denial of Service: Past, Present, and Future

James Davis (Purdue University – West Lafayette, US)

Regular expression denial of service (ReDoS) is an asymmetric cyberattack that has become prominent in recent years. This attack exploits the slow worst-case matching time of regular expression (regex) engines. In this talk, I describe the history of ReDoS, recent developments, and open problems for the future. I trace the history of slow regex performance due to backtracking back to the seminal work of Thompson in 1968, describe how this property can be applied for ReDoS, and indicate how it has been discovered and rediscovered over the years. In the past decade, ReDoS has shifted from a curiosity to a mainstream security concern, thanks to industrial disasters, changes in web software architectures, and the availability of measurement tools. Most mainstream regex engines have now been defended against ReDoS to varying degrees of success. I close by indicating open problems focused on the need to parameterize our worst-case regex analyses to modern algorithms that no longer rely on naïve backtracking.

Participants

- Antoine Amarilli
INRIA Lille, FR
- Hideo Bannai
Institute of Science Tokyo, JP
- Ruben Becker
University of Venice, IT
- Giulia Bernardini
University of Trieste, IT
- Philip Bille
Technical University of Denmark – Lyngby, DK
- Manuel Cáceres
Aalto University, FI
- Davide Cenzato
University of Venice, IT
- James Davis
Purdue University – West Lafayette, US
- Dominik D. Freydenberger
Loughborough University, GB
- Pawel Gawrychowski
University of Wroclaw, PL
- Adrián Gómez Brandón
University of Coruña, ES
- Inge Li Gørtz
Technical University of Denmark – Lyngby, DK
- Roberto Grossi
University of Pisa, IT
- Moshe Lewenstein
Bar-Ilan University – Ramat Gan, IL
- Konstantinos Mamouras
Rice University – Houston, US
- Sebastian Maneth
Universität Bremen, DE
- Wim Martens
Universität Bayreuth, DE
- Yasuhiko Minamide
Institute of Science Tokyo, JP
- Gonzalo Navarro
University of Chile – Santiago de Chile, CL
- Nicola Prezza
University of Venice, IT
- Cristian Riveros
PUC – Santiago de Chile, CL
- Markus L. Schmid
HU Berlin, DE
- Teresa Steiner
Technical University of Denmark – Lyngby, DK
- Michelle Sweering
CWI – Amsterdam, NL
- Simon Rumle Tarnow
Technical University of Denmark – Lyngby, DK
- Brink van der Merwe
University of Stellenbosch, ZA