Ulm University | 89069 Ulm | Germany

Faculty for Engineering, Computer Science and Psychology
Institute of Databases and Information Systems (DBIS)

Reducing a Set of Regular Expressions and Analyzing Differences of Domain-specific Statistic Reporting

Bachelor Thesis at Ulm University

Presented by:
Tobias Kalmbach
tobias.kalmbach@uni-ulm.de
1046229

Examiners:
Prof. Dr.-Ing. Ansgar Scherp

Supervisors:
Marcel Hoffmann
Nicolas Lell

2022

Version November 2, 2022

Contents

List of Figures

List of Tables

Reducing a Set of Regular Expressions and Analyzing Differences of Domain-specific Statistic Reporting

Tobias Kalmbach
tobias.kalmbach@uni-ulm.de
Ulm University
Germany

ABSTRACT

Daily publication of scientific papers makes it impossible to manually review each one. Therefore, an automatic extraction of key information is desirable. In this paper, we examine STEREO, a tool for extracting statistics from scientific papers using regular expressions. By adapting an existing regular expression inclusion algorithm for our use case, we decrease the number of regular expressions used in STEREO by about 33.8%. We derive common patterns from the condensed rule set that can be used for the creation of new rules. We also apply STEREO, which was previously trained in the medical domain, to a new scientific domain, namely Human-Computer-Interaction (HCI), and reevaluate it. According to our research, statistics in the HCI domain are similar to those in the medical domain, although a higher percentage of APA-conform statistics were found in the HCI domain. Additionally, we compare extraction on PDF and LaTeX source files, finding LaTeX to be more reliable in extraction.

CCS CONCEPTS

• **Information systems** → *Information extraction*; • **Applied computing → Document management and text processing**; • **Theory of computation → Regular languages**.

KEYWORDS

regular expression inclusion, statistics extraction, structured data extraction

1 INTRODUCTION

An abundance of scientific papers are published daily. The large and rapidly growing number of papers is too extensive to scan manually. In particular, assessing the published results in terms of insights generated by the statistical analyses is very challenging. A quick overview of published statistics can be used by authors to find statistical errors in their studies more easily or improve their notation. Additionally, readers can easily access statistics to verify and check them. Moreover, extracting statistics together with metadata (authors, title, etc.) can enable researchers to get an impression of an article without the need to fully read it. Reliably knowing if and optimally what statistics a paper publishes can also be important for further research or usage in new papers, such as related work. Tools like *statcheck* [15] provide very accurate extraction of statistics reported in accordance with the commonly used writing style guide of the American Psychology Association (APA) [2]. However, previous research found that in a sample of 113k statistics, extracted from pre-prints in the life sciences and medical domain, less than one percent were APA-conform [8]. In this work, we extend STEREO (STatistic ExtRaction of Experimental cOnditions) [8], an automatic statistics extraction pipeline for statistics presented in APA style notation as well as non-APA notation. STEREO learns regular expressions (rules) to extract statistics using hierarchical active wrapper learning. During the development of STEREO, many rules were added manually for specific corner cases, totaling 1,510 rules.

Inspecting these 1,510 rules can show that rules added later in the learning process generalize well and previously added rules become obsolete. We measure the generalization of a rule by the number of absorbed other rules. We make the assumption that all created rules have a distinct reason to exist, as all current rules were added due to a specific sentence that did or did not include a statistic. However, in some cases, a more general rule was added later, which can now absorb less generalized rules, reducing the rule set size. It is of interest how many of these rules are covered by others and how much a potential size reduction of the rule set might impact performance. Importantly, performance is not the only criteria. Reducing the number of rules can help identify common patterns of non-APA statistics (e. g., incomplete reporting) and derive recommendations to improve statistics reporting. Subsequently, general rule patterns that indicate a sentence does not contain a statistic can serve as guidance for future active wrapper approaches. One can make use of these general rule patterns to avoid creating excess or unnecessary rules. With this reasoning, we apply a Deterministic Finite Automaton based algorithm [5] to minimize the existing set of regular expressions.

The current implementation of STEREO was trained on the COVID-19 Open Research Dataset (CORD-19), a dataset containing papers on COVID-19 and coronaviruses in general. Inspecting other scientific domains may result in finding uncovered statistic types and reporting styles as well as different variations of non-APA reporting or deviations from existing errors. Therefore, we aim to transfer STEREO to a new domain of science, namely Human-Computer-Interaction. Thus, one needs to extend the rule set with additional rules. A growing rule set motivates a size reduction of the rule set, as well.

Furthermore, STEREO currently uses JSON files, which contain the extracted text parts of scientific papers. We extend and apply STEREO to both LaTeX and PDF files that were converted to text in order to explore what impact the input format has for statistics extraction. We focus on unflagged statistics and formatting errors for all three file types.

In summary, this work makes the following contributions:

- We analyze the extraction rules from STEREO, aim to reduce the large rule set, and analyze general patterns that identify sentences that do not contain statistics. These patterns can be used in future statistic extraction to effectively identify false positive statistics.
- We extend the rule set by repeating the active wrapper learning on a new domain, Human-Computer-Interaction. We also evaluate the reduced rule set in the new domain. As measures, we consider precision, coverage, and runtime performance with the reduced rule set and compare it with the full rule set.

- We investigate the extraction from LaTeX vs. PDF files converted to text to measure statistic extraction performance and false positive counts.

Section 2 presents related work. Section 3 specifies the computation of the rule set minimization, and Section 4 explains the implementation of the experimental apparatus. Sections 5 and 6 present and discuss the results.

2 RELATED WORK

First, we present some regular expression inclusion algorithms. Then, we give an overview of approaches for the extraction of statistics, and finally, we briefly summarize.

2.1 Rule Set Inclusion Algorithms

We give a concise overview of the formal definition of regular expressions we use in this paper, following the definition of Chen and Xu [5].

- $a, b, c, \ldots$: members of the alphabet used in the expression
- ϵ: an empty string
- ab or $a\&b$: the concatenation of a and b (i. e., a must be directly followed by b)
- $a|b$: the choice of a or b (i. e., either a or b is accepted). Other literature often uses + as an choice operator
- a^*: the Kleene Star operator (i. e., a can be repeated 0 or more times)

Regarding regular expression inclusion (i.e., minimal rule set computation), many algorithms are limited to determining inclusion using one-unambiguous regular expressions. One-unambiguous regular expressions were first presented by Brüggemann-Klein and Wood [4] and are a subset of regular expressions in general. They are regular expressions that can match every word (in their respective language) in a unique way, without looking ahead. For example, $(a_1|b_1)^*(a_2|\epsilon)$ (numbered for clarity) is not one-unambiguous, as the word baa can be formed as $b_1a_1a_1$ or $b_1a_1a_2$. However, $(b^*a)^*$ describes the same language but is one-unambiguous.

Chen and Xu [5] present two algorithms for regular expression inclusion. The first, is an automata-based algorithm that converts the given one-unambiguous regular expressions into Deterministic Finite Automatons (DFAs) and subsequently compares those. A DFA is a finite automaton that accepts or rejects an input by traversing states in the automaton. An input is accepted if the process terminates in an accepting state and rejected otherwise. The second algorithm is a derivative-based algorithm. Derivatives of regular expressions are sub-expressions, which in turn are valid regular expressions themselves. The idea here is that if an expression A is included in an expression B, all derivatives of A are also derivatives of B. The algorithm generates all derivatives of both expressions. If all derivatives of one expression are included in the other, then the first expression is included in the second.

Similar to the first algorithm of Chen and Xu [5], Nipkow and Traytel [14] present a framework to determine if two given regular expressions are equivalent. Equivalence is a stricter requirement than in the paper by Chen and Xu [5]. The presented framework works by dynamically creating automata from one regular expression and using "computations on regular expression-like objects" [14, p. 2] as a substitute for the traditional transition table. "Regular expression-like" means the embedding of the regular expression in a state object represented by the language of the regular expression.

A different approach is presented in Hovland [10]. Instead of using automata, an inference system is used to inductively determine a binary relationship between one-unambiguous regular expressions. This inference system abstracts types of regular expressions and simplifies them for further use. If all simplifications reach the trivial case $\epsilon \subseteq l$, where l is any regular expression, the two regular expressions are in an inclusion relationship. The algorithm uses a depth-first search in combination with the inference system. The algorithm always runs in polynomial time, performing better than the quadratic runtime of an automaton-based approach like the one Chen and Xu [5] present.

2.2 Statistics Extraction

As mentioned in the introduction, we aim to extract reported statistics from scientific papers. Lanka et al. [11] present a regular expression-based approach to extract statistics from scientific papers. They use a single regular expression to match the p-value in its representation according to the statistical test, e. g., one regular expression to match a t-Test reported as **`t(df)=float, p (<, >, =) float`**. Their model achieved an extraction accuracy of 90.2% when no accompanying test statistic was present (only a p-value) and 79.0% with a test statistic.

A similar, generally well-known approach is *statcheck* [15]. *statcheck* is an R package that allows the user to extract and verify published statistics. *statcheck* is limited to extracting statistics that match APA guidelines exactly. However, this limitation always allows recalculation as well as an assessment of the consistency of the p-value, as all reported statistics and degrees of freedom must be present by construction. In 2017, Schmidt [22] disputed the effectiveness of *statcheck* in their paper aptly named "Statcheck does not work: All the numbers. Reply to Nuijten et al. (2017)". Here they criticize the *statcheck* authors for the testing conducted in their follow-up paper [16]. Upon retesting, Schmidt [22] finds that *statcheck* simply does not detect many reported statistics and therefore does not flag the statistical tests, meaning these undetected tests cannot be inconsistent. This leads to high accuracy and specificity, which is the proportion of unflagged tests classified correctly. The original calculation of sensitivity, the proportion of truly inconsistent tests flagged by the program, ignores tests the program failed to detect. If added, the sensitivity drops from the reported 85.3% to 51.8%. All in all, the reported accuracy and overall performance of *statcheck* are called into question, even though *statcheck* is still widely used [9, 17, 20].

Both *statcheck* [15] and Lanka et al. [11] do not have any built-in feature to match non-APA statistics and can therefore only match the very strict pattern. In order to expand this strict pattern, Böschen [3] uses a similar, yet more sophisticated approach than *statcheck*. They present a text-mining approach on XML documents that are structured following the Journal Archiving Tag System NISO-JATS. The algorithm differentiates between three result types. Computable results, where the p-value is given and can be recalculated; checkable results, where the p-value is not given but can be calculated; and lastly, uncomputable results, where the p-value may or may not be reported but cannot be calculated. The algorithm uses CERMINE [24] to format papers. CERMINE (Content ExtRactor and MINEr) extracts the contents and metadata from papers given as PDFs and formats them in accordance with NISO-JATS. With this structure, Böschen [3] applies several transformations to letters and numbers (e. g., greek letters or fractions). The authors note that with just these transformations, the original *statcheck* performs better, due to fewer missed statistical tests. After these transformations, the extraction algorithm by Böschen [3] works as follows: Sentences are only selected if they contain at least one letter, followed by an operator ($<, >, =, \leq, \geq$), which in turn is followed by a number. With this, the sentences are split at a set of given words (e. g., and, or, of,...) and at commas following a word. Surrounding text is removed using regular expressions. Individual heuristics, to cope with varying reporting styles, are applied to extract the recognized test statistics, the operator, degrees of freedom, and p value. As the requirements are not as strict as *statcheck*'s, Böschen [3] generally achieves a higher accuracy.

Lastly, STEREO by Epp et al. [8]. STEREO uses hierarchical active wrapper learning to learn regular expressions (rules) that determine whether or not a sentence contains statistics. The rules are divided into R^+ and R^- rules. R^+ rules are rules that match statistics presented in a sentence, while R^- rules denote that a sentence does not include statistics. R^+ rules have additional sub-rules, which are used to capture statistic parts (e. g., p-value) after the statistic type has been identified by the main R^+ rule. During the active wrapper learning, every sentence that does not contain a number is ignored. For any remaining sentences not matched by any rules, the user is prompted to create a new rule to cover the new case. STEREO achieved a precision close of to 100% for APA-conform statistics and a precision of 95% for non-APA statistics.

2.3 Summary

All in all, there are already many approaches for both regular expression inclusion and statistics extraction. As previously mentioned, this work is based on STEREO and therefore uses methods for statistics extraction presented by Epp et al. [8]. In particular, we transfer STEREO to a new scientific domain and potentially find new rules for statistics extraction. Additionally, we use the automaton-based algorithm presented in Chen and Xu [5] to find a minimal rule set, as it is the most straightforward implementation and the runtime is negligible, as our regular expressions are usually short (< 100 characters).

3 COMPUTING THE RULE SET INCLUSION

In Section 3.1, we explain the benefits of R^- rules. Section 3.2 details our inclusion algorithm, which tests two regular expressions for inclusion using the algorithm given by Chen and Xu [5]. In a preprocessing step, we convert Python regular expressions into expressions following the formal definition, which Chen and Xu [5] did not need, as they apply their algorithm to regular expressions already following the formal definition. We are the first to implement such a transformation that we are aware of. Section 3.3 shows the algorithm with a simple example.

3.1 Why R^- rules?

STEREO applies many R^- rules (1, 425), while not using many R^+ rules (85). The reasons supporting the use of these R^- rules will be discussed here.

Extracting text parts that follow a predefined pattern or system, in this context APA-conform statistics, is very simple. The pattern produces finitely many base-structures, which can then in turn be matched. This method was used by Nuijten et al. [15].

Due to lack of knowledge or proficiency, not all statistics in papers are reported in APA fashion, but with deviations ranging from slight spelling errors to disregarding the format completely. In contrast, these deviations are in principal arbitrary, and finding rules that exactly capture these deviations is very difficult or impossible in a large-scale dataset. As a solution to this problem, Epp et al. [8] use an adapted version of statistic recognition and extraction. The rules that match statistics (R^+) are extended to be more lenient, thus accepting close matches or common mistakes. As more structures need to be matched, STEREO has more matching rules (85 for statistics detection with 52 sub-rules for value extraction) in comparison to the matching rules in *statcheck* (8 for detection, 23 for value extraction).

However, during the active wrapper learning phase, when searching for statistics not yet captured by R^+ rules, all sentences containing numbers are marked as potential statistics, which is a very wide cast net. In order to eliminate the need to manually rule out every non-statistic sentence, STEREO adds R^- rules which rule out that specific sentences report statistics. For example, these rules capture a number as a part of a table caption (i. e., "Table 1") and eliminate it from further processing.

A reduction of these rules can show common patterns that can provide a good guideline how R^- rules should be written for similar future active wrapper approaches. Moreover, a size reduction will result in a faster runtime as fewer rules need to be applied for each sentence.

3.2 Algorithm Description

In the following, we describe the algorithm for the inclusion of regular expressions. Each given regular expression is cleaned and normalized. Then a DFA is constructed for comparison, following the inclusion algorithm given by Chen and Xu [5]. At first, it might seem easy to apply an inclusion algorithm to regular expressions directly. This is not necessarily the case. Regular expressions, especially Python regular expressions, can represent the same thing in many ways. For example, a three letter word containing **`a`**, **`b`**, **`c`** in any order and quantity can be represented as **`[abc]{3}`**, **`(a|b|c){3}`**, **`[abc][abc][abc]`**, and more. Therefore, applying an inclusion algorithm to regular expressions directly requires more differentiation. As mentioned in Section 2, Hovland [10] uses an inference system to test regular expression inclusion directly. In their algorithm, they use ten different cases to abstract a given regular expression until the inclusion problem is easily decidable. In contrast, every regular expression can be converted into an equivalent DFA. Using DFAs, it is possible to apply the same algorithm state-by-state without the need to differentiate between different cases. While the algorithm presented by Hovland [10] is more efficient (polynomial run-time) than an algorithm, like Chen and Xu [5], using the construction of equivalent automatons (possibly quadratic run-time), the implementation of the latter is simpler and the efficiency is not of utmost importance in our case. The regular expressions we work with are usually[1] less than 100 characters long before preprocessing, with a few exceptions, making the performance difference mostly negligible. Therefore, we use the algorithm described by Chen and Xu [5].

The notation [12] we use for automata is defined by:

- A: an automaton
- Q_A: set of all states in A
- q_A: start state in A
- F_A: set of all accepting states in A

[1] 16 sentences were longer than 100 characters and 3 sentences were longer than 150 characters (longest 225 characters)

- δ_A: all transitions in A
- Σ_A: alphabet used in A
- $L(A)$: the language accepted by A

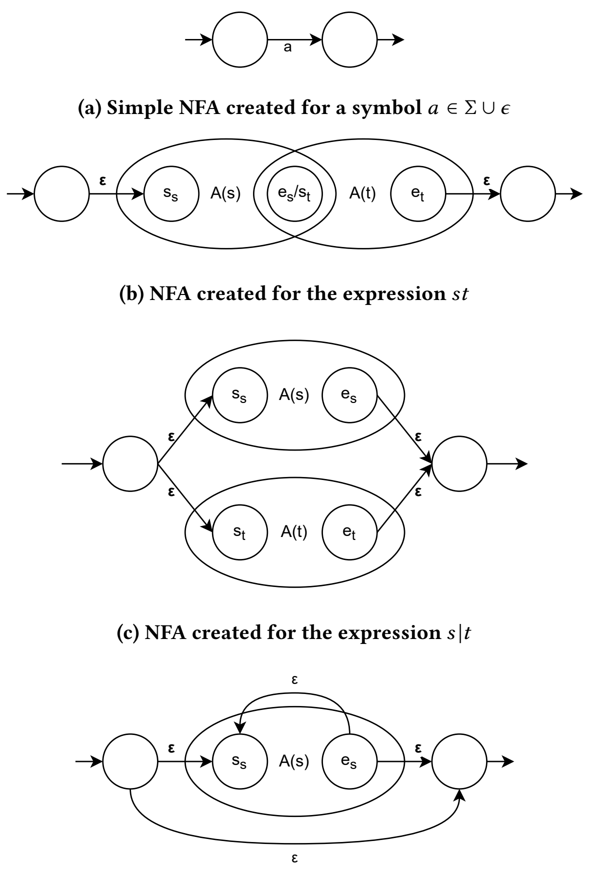
Algorithm 1: Regular Expression Inclusion

```
   Data: List of regular expressions $R$
   Result: Array detailing every regular expression
           included by an other
 1 Initialize empty array $S$ with length of regular expression
   list;
 2 for $r_1$ in $R$ do
 3     Initialize empty set $T$;
 4     $c_1 \leftarrow normalize(r_1)$;
 5     $n_1 \leftarrow nfa(c_1)$;
 6     $a_1 \leftarrow dfa(n_1).asComplete()$;
 7     for $r_2$ in $R, r_2 \neq r_1$ do
 8         $c_2 \leftarrow normalize(r_2)$;
 9         $n_2 \leftarrow nfa(c_2)$;
10         if $\Sigma_{n_2} \not\subseteq \Sigma_{n_1}$ then
11             continue with next $r_1$;
12         end
13         $a_2 \leftarrow dfa(n_2).asComplete()$;
           /* Inclusion check with Algorithm 2    */
14         if $inclusion(a_1, a_2)$ then
15             Add $r_2$ to $T$;
16         end
17     end
18     Save $T$ at the index of $r_1$ in $S$;
19 end
```

Our adapted implementation can be seen in Algorithm 1. The algorithm can be split into two steps, which are repeated for every rule. In the first step, a regular expression is preprocessed and an equivalent, complete DFA is created. This regular expression is the rule we want to find all the included expressions for. Next, excluding the rule used in step one, we iterate through each rule in the set of rules. We use the same preprocessing and automaton creation for these rules. The resulting automata are subsequently checked for inclusion. The algorithm's output is an array of lists, where each position in the array corresponds to a rule's index in the given list of regular expressions. The respective lists in the array contain all the rules included by the rule at that index.

Preprocessing. We begin by transforming the given Python regular expression to a formal definition by preprocessing them. For readers not familiar with the syntax of Python regular expressions, we recommend the official documentation[2] as a reference. In the first step, we replace all meta characters (e. g., `\d` or `\w`) with their expanded form (e. g., `( 0 | 1 | 2 | ... | 8 | 9 )`). Also, character sets (e. g., `[a-z]`) and repetition qualifiers are expanded as well, e. g., `a{2,4}` becomes `aa|aaa|aaaa`. Additionally, Python does not have a symbol for concatenation, so we fill all implicit concatenation with an `&` for the next step, e. g., `ab` becomes `a&b`. Next, we use Dijkstra's shunting-yard algorithm (SYA) [7] for a definitive normalized form that does not include any parentheses. The SYA transforms expressions, in this case, regular expressions, given in infix notation into postfix notation, e. g., `a|b` becomes `ab|`. In short, SYA uses an output and

[2] https://docs.python.org/3/howto/regex.html



(a) Simple NFA created for a symbol $a \in \Sigma \cup \epsilon$

(b) NFA created for the expression st

(c) NFA created for the expression $s|t$

(d) NFA created for the expression $s*$

Figure 1: Basic NFAs used in Thompson's Construction. s, t **are regular (sub-)expressions.** $A(s)$ **refers to the automaton created for** s. s_s **and** e_s **refer to the starting and final state of** $A(s)$ **respectively.** ϵ **is an empty transition.**

operator stack and a predefined operator precedence. For regular expressions, the operator precedence is '∗', '&', '|' from highest to lowest. The given expression is read character by character. All elements of the alphabet are immediately pushed onto the output stack, while operators are pushed to the operator stack. If the operator stack is non-empty, all contained operators are popped off the operator stack and added to the output stack while the top-most operator has an equal or higher precedence. Once an operator with lower precedence is on top of the operator stack, add the regarded operator to the operator stack. A special case is parentheses. When finding an opening bracket, it is always added to the operator stack. The normal procedure is continued until the closing bracket is found, upon which all operators are popped and added to the output, until the corresponding opening bracket is found. The parentheses are then discarded.

We show the application step-by-step in Table 1. Another thorough example of an application of SYA is given by Rastogi et al. [19, p. 3]. In this example, the algorithm is applied to an arithmetic example, which still demonstrates the procedure.

Automaton Construction. As mentioned in Section 2, Chen and Xu [5] present two approaches, of which we use the automaton-based algorithm. When given two regular expressions, they convert these into Glushkov automata, using Glushkov's construction, and check the inclusion of the resulting DFA. It should be noted that in our implementation we do not use Glushkov's construction but Thompson's construction, which creates an Nondeterministic Finite Automaton (NFA). However, converting the NFA to a DFA, the resulting automata have been shown to be equivalent [21]. Thompson's construction, given by Thompson [23], builds an NFA with a bottom-up approach. Given a regular expression, first NFAs are constructed for the sub-expressions and then merged. The NFAs for sub-expression do not necessarily have a start or accepting state and may have non-connected arrows, which will be connected when merging. In Thompson's construction, there are four basic NFAs for a simple transition with a given letter, concatenation, e. g., ab, alternative, e. g., $a|b$, and the Kleene star expression, e. g., $a*$. A general representation of these basic NFAs can be found in Figure 1. This representation is adapted from Aho et al. [1]. Some more complete overviews of the construction are presented by Watson [26] and Aho et al. [1].

After applying Thompson's construction to a preprocessed regular expression, we convert the resulting NFA into a DFA with powerset construction [18].

In the following, we further discuss the inclusion algorithm used to determine `a1.includes(a2)`.

Algorithm 2: An optimized automaton inclusion algorithm. Algorithm 2 is an optimized version of Algorithm 3 presented by Chen and Xu [5].

```
   Data: Two DFAs, $A'$ the complement of $A1$ and $A2$
   Result: Boolean whether $A2 \subseteq A1$
1  Initialize empty stack $Q$;
2  Push $(q_{A'}, q_{A2})$ onto $Q$;
3  while $Q$ is not empty do
4      $(p, q) \leftarrow Q.pop()$;
5      if $(p, q)$ is unmarked then
6          for $a$ in $\Sigma$ do
7              if $\delta_{A2}(q, a)$ is defined then
8                  if $\delta_{A'}(p, a)$ and $\delta_{A2}(q, a)$ are accepting
                    states then
9                      return $FALSE$;
10                 else
11                     Push $(\delta_{A'}(p, a), \delta_{A2}(q, a))$ onto $Q$;
12                     Mark $(p, q)$;
13                 end
14             end
15         end
16     end
17 end
18 return $TRUE$;
```

Inclusion Algorithm. Following Chen and Xu [5], we require $\Sigma_{r2} \subseteq \Sigma_{r1}$ as a necessary condition for a regular expression $r1$ to include another regular expression $r2$. This is already checked in Algorithm 1. The specific inclusion algorithm consists of four steps. The first step is the construction of automata, done in Algorithm 1. The inclusion algorithm for two DFAs given by Chen and Xu [5] can be seen in Algorithm 3. We use the optimized version also given by Chen and Xu [5], see Algorithm 2.

The intuitive reasoning behind the inclusion algorithm is that the complement of an automaton accepts the "exact opposite" language, i. e., all previously accepted words are not accepted and vice versa. We use an example automaton called $A1$ and its complement A'. When combining A' with a different automaton, $A2$, a path from the beginning state to an accepting state can only exist when the language of the combined automaton is non-empty. In other words, $A2$ accepts a word that A' also contains in their language. $A2$ accepts a word that $A1$ does not, so $A2$ is not included since an automaton and its complement cannot accept the same word.

Following Chen and Xu [5], combining two automata A' and $A2$, while A' is the complement of a different automaton, $A1$, is done by constructing an automaton with the cross product of the respective states (i. e., $Q_{A'} \times Q_{A2}$), merging the alphabets (i. e., $\Sigma = \Sigma_{A'} \cup \Sigma_{A2}$), and defining the start state q as the tuple $(q_{A'}, q_{A2})$ and the accepting states $F = F_{A'} \times F_{A2}$. Furthermore, we define the two-fold transition function, $\delta((p, q), a) = (\delta_{A'}(p, a), \delta_{A2}(q, a))$. We can always assume that $\delta_{A'}(q, a)$ exists if $\delta_{A2}(p, a)$ does, as $\Sigma_{A2} \subseteq \Sigma_{A1}$ is required and checked, and A' and $A2$ are complete automata.

All in all, the algorithm checks whether $\overline{L(A1)} \cap L(A2) = \emptyset$, which is exactly non-empty if $A2$'s language contains some word that is not in $A1$'s language.

Algorithm Optimization. As one can see in Algorithm 2, the optimized version does not explicitly create any cross product of states. Instead, we iterate state-by-state using the same transition letter for both automata, keeping track of our current position with a tuple (p, q). The algorithm terminates as soon as we reach a goal state in both automata, returning earlier than without the optimization. Only if the automata are in an inclusion relation (i. e., no goal state is found), the graph is searched completely, resulting in the same performance as without the alteration.

This improvement benefits us as many regular expressions have a very specific use-case. These regular expressions most likely cannot be generalized and included by some other rule, so an early exit is possible.

3.3 Algorithm Example

To illustrate our approach, we use the regular expression **`[a-b](a|b)*`** as an example. We start with escaping our expression, resulting in **`(b|a)&(a|b)*`**. Continuing with the shunting-yard algorithm, we get **`ba|ab|*&`**. A step-by-step conversion for our example can be seen in Table 1. The Thompson's construction for our example can be seen in Figure 2 and the subsequent powerset construction is found in Figure 3.

We now show how the improved inclusion algorithm works by using a simple example. We check if **`ab`** $\subseteq$ **`[a-b](a|b)*`**, which is true. As the DFA for **`[a-b](a|b)*`** is complete already, we only need to form the complement. The DFAs for both expressions can be seen in Figure 4. The state numbering for the automaton in Figure 4b has been shifted for clear identification.

We begin by pushing the combined state $(1, 6)$, i. e., beginning state of the automaton in Figure 4a and beginning state of the automaton in Figure 4b. When popped, we can traverse from $(1, 6)$ to $(3, 9)$ with the transition b and to $(2, 7)$ with the transition a. Neither of these states have two accepting states so we mark $(1, 6)$ as completed and continue. Next, we pop $(2, 7)$. Reachable from here are $(4, 9)$ with a and $(5, 8)$ with b. Again, we have not

Table 1: Step-by-step shunting-yard conversion of `(b|a)&(a|b)*`. The reason column indicates what rule was used to get the *next* line. Σ refers to the alphabet of the expression, 'op.' is short for operator, and > refers to higher precedence.

Input	Regarded Letter or Operator	Operator Stack	Output Stack	Reason
$(b\vert a)\&(a\vert b)*$	-	-	-	-
$b\vert a)\&(a\vert b)*$	(	-	-	Opening (
$\vert a)\&(a\vert b)*$	b	(	-	$b \in \Sigma$
$a)\&(a\vert b)*$	$\vert$	(	b	op.
$)\&(a\vert b)*$	a	$(\vert$	b	$a \in \Sigma$
$\&(a\vert b)*$	)	$(\vert$	ba	Closing)
$\&(a\vert b)*$	)	(	$ba\vert$	Closing)
$(a\vert b)*$	&	-	$ba\vert$	op.
$a\vert b)*$	(	&	$ba\vert$	Opening (
$\vert b)*$	a	&(	$ba\vert$	$a \in \Sigma$
$b)*$	$\vert$	&(	$ba\vert a$	op.
$)*$	b	$\&(\vert$	$ba\vert a$	$b \in \Sigma$
$*$	)	$\&(\vert$	$ba\vert ab$	Closing)
$*$	)	&(	$ba\vert ab\vert$	Closing)
-	$*$	&	$ba\vert ab\vert$	op., $* > \&$
-	-	&$*$	$ba\vert ab\vert$	Pop op.
-	-	&	$ba\vert ab\vert *$	Pop op.
-	-	-	$ba\vert ab\vert * \&$	-

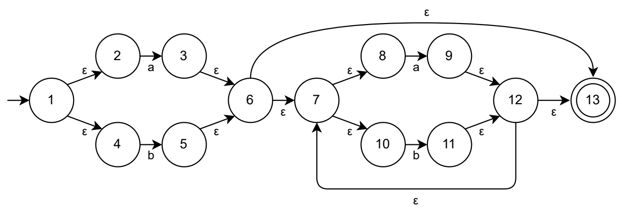


Figure 2: Thompson's construction for `ba|ab|*&` (input was `[a-b](a|b)*`). Nodes 1 through 6 represent `[a-b]`, while nodes 6 through 13 represent `(a|b)*`. For a larger version see Appendix C.

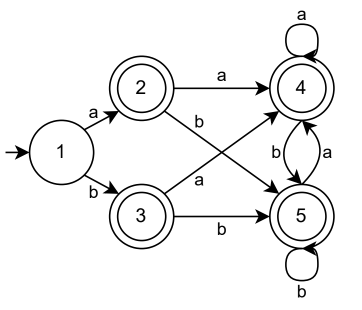
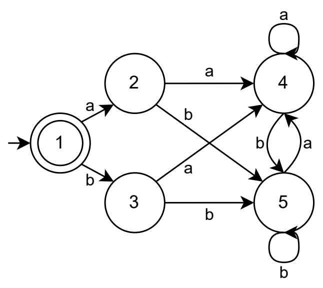
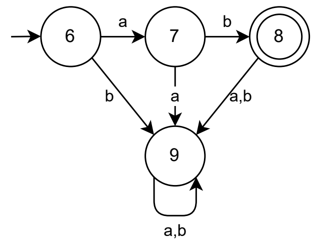


Figure 3: Powerset construction for `ba|ab|*&` (input was `[a-b](a|b)*`). Note: This is not an optimally minimized DFA.

reached a state with two accepting states, so we mark (2, 7) completed and continue. From (5, 8), we can reach (4, 9) and (5, 9), as no accepting state was found we mark (5, 8) completed and continue. Next, (5, 9) is only followed by (4, 9) and (5, 9). As no accepting state was found we mark (5, 9) completed and continue. As we can only reach (4, 9) and (5, 9) from (4, 9) again, we



(a) Complement of the DFA created for the expression `[a-b](a|b)*`, see Figure 3.



(b) The complete DFA created for the expression `ab`

Figure 4: The DFAs created for the expressions `[a-b](a|b)*` and `ab` respectively. Note that the DFA for `[a-b](a|b)*` has been converted to the complement and both DFAs are complete.

mark (4, 9) as completed. The last unmarked state in our stack is (3, 9). However, we can not reach any unmarked states from here. Now we have traversed the whole combined automaton without explicitly creating it and have not found a two-fold accepting state. Therefore, `ab` ⊆ `[a-b](a|b)*` holds (see Algorithm 2 in Section 3.2).

3.4 Summary

We discussed what R^- rules are and how they benefit our active wrapper learning, improving performance and usability. Reducing the size of the R^- rule set can show common patterns and provide guidelines for future R^- rule creation.

Second, we present the procedure for regular expression inclusion. Overall, our algorithm, adapted from Chen and Xu [5], takes two Python regular expressions and formalizes them. We show how standard Python regular expressions can be converted to equivalent expressions following the formal definition. An NFA is generated for each regular expression before being converted into a DFA. A combined automaton is built using the complement of the presumed larger DFA and leaving the other as is. The two automata are then tested for inclusion by iterating through the combined automatons' states, following the algorithm given by Chen and Xu [5].

4 EXPERIMENTAL APPARATUS

In this section, we discuss the datasets used in this paper. Next, we describe the preprocessing steps, followed by our procedure. Lastly, we discuss measures used during the experiments.

4.1 Datasets

The datasets used in this paper are divided into two categories: rule sets and scientific papers.

4.1.1 Rule set. This dataset contains the rule set on which minimal rule set analysis is performed. Additionally, this rule set is used for domain comparison.

STEREO rule set. This is the rule set created in the original STEREO paper using CORD-19. The dataset consists of in total 1, 510 rules, divided into 85 R^+ and 1, 425 R^- rules. The rule set of STEREO is manually created using an active wrapper induction learning. Each rule as an internal ID (determined at the time of creation) and its corresponding regular expression. In the inductive wrapper learning, a new rule is created if the regarded sentence is not covered by any existing rules. This new rule is then appended to the existing set of rules, and the next higher ID is assigned to this rule (IDs are incremented by one). Implicitly, a higher rule ID means that the rule was later in the process of applying the active wrapper.

It is unlikely that the 85 R^+ rules can be optimized greatly, as these rules match a variety of statistics. Statistic types are distinguished by letter, e. g., U for Mann-Whitney U tests vs. t for t-Test. Thus, R^+ rules can most likely only include rules of the same statistic type. However, rules for the same statistic type primarily capture different cases, for example, different statistic values missing. On the other hand, 1, 425 R^- rules can be optimized to possibly significantly improve performance and maintainability, as well as common patterns used to identify sentences as non-statistic.

4.1.2 Scientific Papers. These datasets provide the raw data we use for statistics extraction. An overview of documents and sentences of these datasets can be seen in Table 2.

COVID-19 Open Research Dataset [25]. This is the original dataset used in STEREO that can be used to test the minimal rule set. This dataset contains more than 100, 000 papers on COVID-19, SARS-CoV-2, and all coronaviruses in general. In STEREO, the date of access is given as 21st September 2020. The CORD-19 dataset version 52[3] (date of publication: 2020-09-21, date of access: 2022-07-11) is a very close match, which we use for comparison. Note, that our version is not an exact copy and contains a few more papers than the original, which leads to some slight deviations in the extraction results and a direct comparison is not possible.

arXiv Dataset [6]. A collection[4] (date of access: 2022-07-11) of more than 1.7 million STEM papers. The dataset comes with a JSON file containing metadata (author, title, category, etc.) that can be used to filter papers. We focus on Human-Computer-Interaction (HCI). HCI studies the use of technology, focusing on the interface between people and computers. Thus, HCI is a very promising non-medicine domain for publishing studies and the corresponding statistics. There are 9, 730 papers tagged with the "cs.HC" (HCI tag on arxiv.org) category, but we focus only on papers with the HCI tag as the primary tag. We also only use papers that were provided as both LaTeX and PDF files. This is needed for a fair comparison of the statistics extraction on both file types. With these restrictions, 4, 023 papers remain. For more information, see Appendix E.

[3]https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge/versions/52
[4]https://www.kaggle.com/datasets/Cornell-University/arxiv

Table 2: An overview of the number of documents and sentences in the datasets used. Only sentences containg at least one digit were counted.

Dataset	# of Documents	# of sentences
CORD-19	110, 427	9, 393, 662
HCI	4, 023	222, 544

4.2 Preprocessing

For the rule set inclusion, we transform the regular expressions given in the Python format into the formal definition, as described in Section 3.2. The Python syntax is too versatile, so this transformation into a formal, stricter expression is required to build the automata.

For the extension to the HCI domain, the papers are given as `.tar.gz` archives containing LaTeX source code. We first read the archive and extract the containing filenames using *tarfile*. We filter `.tex` files and use *pylatexenc*[5] to read and parse the content to plain text while removing all table, figure, lstlisting, and tikzpicture environments. As in STEREO, line breaks are removed and the plain text is split into sentences using the regular expression `\.\s?[A-Z]`. Every sentence that does not contain a digit is removed. The same process is repeated for the corresponding PDF files. We use *pdftotext*[6] to easily convert the PDF files into raw text. As before, line breaks are removed and the text is split into sentences.

4.3 Procedure

In the first step, we compute the minimal set of extraction rules on the existing rule set. We do a pairwise comparison for all the rules in the rule set using Algorithm 3. This results in an array of lists containing every rule that is included by another rule.

Next, we repeat the manual evaluation from STEREO in the HCI domain. Of the 4, 023 papers, we use 200 randomly selected papers (or about 5% of all papers) to learn new regular expression to extract statistics from HCI papers. The rule learning is applied to both LaTeX and PDF files, sampling 200 papers for each file type. This step generates our new rule set. The resulting rules are an addition to the STEREO rule set. As we use a new input format (i. e., LaTeX and PDF), this will result in new format-specific rules being added, which further improves the robustness and completeness of the extraction.

Finally, we evaluate the precision for every statistic type. We follow a very similar pattern to the original STEREO paper [8]. We extract all sentences from all papers in the respective corpus that are matched by R^+ rules. We then sample 200 sentences for every statistic type, or use all extracted sentences if there were fewer than 200 extractions, to check if the statistic types are matched and extracted correctly. We highlight the difference in APA versus non-APA reporting. Furthermore, we extract 200 sentences, from random documents we did not use for rule learning, matched by R^- rules, to test whether R^- rules do not reject statistics correctly. Lastly, we extract 200 sentences that were neither matched by R^+ nor R^- to check for unrecognized statistics or parsing errors.

4.4 Measures

While reducing the rule set, we evaluate each rule. For every R^+ and R^- rule, we measure the runtime of inclusion and the number

[5]https://pypi.org/project/pylatexenc/
[6]https://pypi.org/project/pdftotext/

of included rules. We surmise that every rule has a specific reason, as all rules were added due to a sentence that did or did not report a statistic. Nevertheless, in some cases, a general rule was added later, which includes less generalized rules, which allows to reduce the rule set size. A simplified example would be one rule specifying **`Fig\.\s\d`** with a later rule specifying **`(Fig\.|Table|Equation)\s\d`**.

Similar to STEREO, our main measure is precision. We calculate the precision by checking the 200 rules per statistic type for false positive matches and then using the following formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (1)$$

Furthermore, we focus on the difference in coverage of the reduced rule set in comparison to the complete, original rule set.

Finally, to gain insights on the runtime improvement of our reduced set, we evaluate the R^- rules using the reduced rule set in comparison to the larger, original rule set. We measure the time it takes to extract 200 sentences matched by R^- rules as a baseline, as here all sentences need to be checked against the list of rules.

5 RESULTS

We structure our results as in the previous sections, beginning with the rule set inclusion, then the transfer to the HCI domain, considering both PDF and LaTeX files as an input. Finally, we re-evaluate the statistics extraction using our newly created rule set.

5.1 Rule Set Inclusion

First, we ran the inclusion algorithm on R^+ rules. We did not see much improvement in the R^+ rules. One rule was removed, but this was an exact duplicate, most likely added by mistake. The sub-rules of an R^+ rule are only used for value extraction. Therefore, sub-rules are also removed when removing R^+ rules, as they serve no purpose anymore.

After running the rule inclusion algorithm for the R^- rules, 483 unique rules out of the total 1,426 rules were included by others. This is a reduction of about 33.8%. A distribution of the amount of included rules per rule can be seen in Figure 5. 1,253 rules included no other rules, 83 included one, and 28 included two rules. On the other hand, 13 rules included more than 20 rules, and 4 had more than 100 rule inclusions. These rules can be seen in Table 3. Naturally, some rules were included more than once. Figure 6 shows which rules were included most often. **`figure \d{1,2}`** was the most included rule with 14 inclusions, and **`table \d+`** was the second-most included rule with 13 inclusions.

We also show the included rules by rule ID and grouped as hundreds, see Figure 7. We see that many rules that were included by others had a rule ID between 400 and 700. Further, approximately 47% of included rules had an ID below 500 and 88% had an ID below 1,000. In general, lower ID rules are included in higher ID rules.

5.2 Transfer to HCI Domain

LaTeX. Using the LaTeX files, we added 13 new R^+ rules and 77 R^- rules. Furthermore, 6 previously added R^- rules were changed alongside the active wrapper to be more general, see Appendix F. The R^+ rules added two new statistics types, which were not used in STEREO. First, the Z-Test, which tests the mean of a

Table 3: Number of R^- rules included for each rule. Sorted by number of included rules and rules with less than 20 inclusions excluded.

Regular Expression	# included rules
`[a-zA-Z]{3,}\s?\d+[\.\,\s\dabcdef]*`	173
`[a-zA-Z]{2,}\s?\d+(\.\d)?`	173
`[a-zA-Z]{3,}\s?\d+[\.\,\s\d]*`	171
`[a-zA-Z]{3,};?\s?\d+`	130
`[a-zA-Z"]+\s?\d{1,3}$`	83
`[a-zA-Z]{3,20}\s\d+(\,\d+)*(\.\d+)?`	72
`[a-zA-Z]{3,20}\d+`	71
`[a-zA-Z]{3,}\s-?\d+(\.\d+)?`	62
`\d+(\,\d+)*(\.\d+)?\s[a-zA-Z]{3,10}`	51
`[a-zA-Z]{4,10}\s\d+(,\s\d+)*`	47
`\d+(\s\d+)?\s[a-zA-Z]{3,10}`	40
`[a-zA-Z]{3,20}-\d+`	21
`[a-zA-Z]{2}\d+(\.\d)?`	21

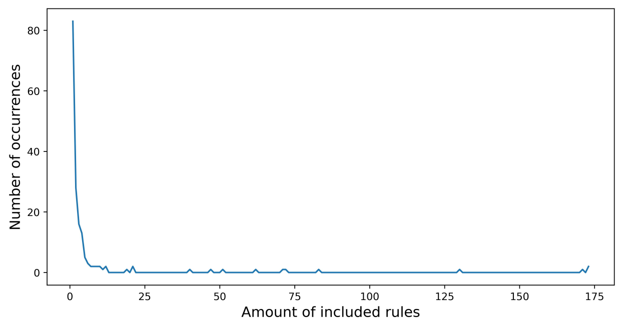


Figure 5: Amount of rules that include n other rules. Rules that include 0 rules were excluded.

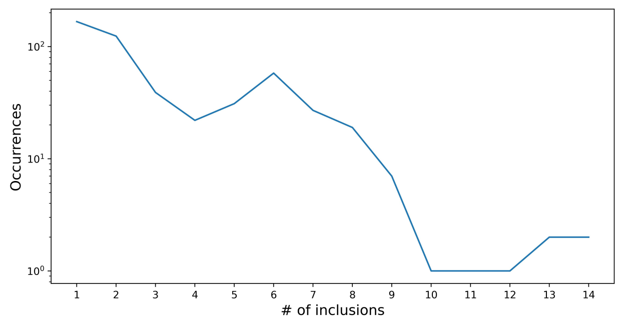


Figure 6: Number of times a rule was included by other rules. The y-axis uses a logarithmic scale. 166 rules were included once, while only one rule was included 13 and one 14 times.

distribution, and second, ANOVA without an r-value. In the original implementation of STEREO, all ANOVA tests that did not contain a r-value were seen as non-APA. However, upon further research we found, that APA guidelines do allow ANOVA to be reported with out an r-value. Therefore, when referencing the percentage of APA conform statistics in a corpus, we mention both including and excluding ANOVA tests without an r-value. For both the Z-Test and ANOVA without an r-value, only APA conform extraction rules were added.

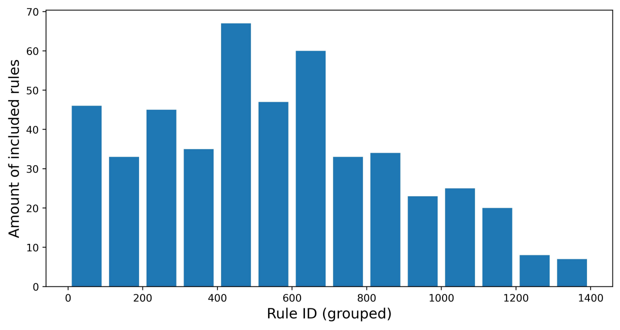


Figure 7: Amount of rules absorbed by rule ID, grouped by the hundreds. The rule ID reflects the order in which a rule was added to the rule set using STEREO's [8] wrapper induction (see Section 4.1.1). When a new rule is added the ID is incremented by +1. See Appendix B.1 for a detailed view, with no grouping.

PDF files. For the PDF files, 9 R^- rules and no R^+ rules were added. We did need to add rules for page numbers and citations. In LaTeX files and in the CORD-19 dataset, page numbers, as well as citations, were automatically removed or never generated. However, converted PDF files contained citations, which in turn included pages of an article in a journal or ACM identifiers. These had a high variance of representation, making finding rules to capture them difficult. Some examples of these variations can be seen in the following:

- `Human Factors in Computing Systems. dl.acm.org, 2853-2859.`
- `ACM, New York, NY, USA, 285-296.`
- `Computer Graphics 19, 12 (2013), 2713-2722.`
- `Virtual Environ. 7(3), 225- 240 (1998).`
- `Thousand Oaks, CA, 508-510 (2007) [49]`

In the end, we added the rule `\),\s\d{1,4}[--]\d{2,4}[.)]` to capture most, if not all, cases. Furthermore, tables could not be removed as previously, which led to some rules being added, but too specific cases were ignored. In general, most numbers were matched by the previously added R^- and R^+ rules.

All in all, learning rules in the HCI domain added 99 rules.

5.3 Evaluation of New Rule Set

HCI Results. Our R^+ rules extract the original statistic types used in STEREO [8] (Pearson's Correlation, Spearman Correlation, Student's t-test, ANOVA, Mann-Whitney U Test, Wilcoxon Signed-Rank Test, and Chi-Square Test) as well as Z-Test and ANOVA without an r-value, added by the previous step.

In the HCI domain, the 4, 023 HCI papers, 6, 321 and 7, 669 sentences matched by an R^+ rule were extracted using the PDF and LaTeX files, respectively. These are about 3% of all sentences in the dataset for both file types. Table 4 shows all reported statistics categorized by type and whether the statistics matched APA style or not. For every statistic type, more statistics were extracted from LaTeX files than from PDF files. The only exceptions are Spearman Correlation (APA) and ANOVA (non-APA), but these exceptions can be attributed to formatting errors (Spearman Correlation was marked as non-APA using LaTeX and ANOVA was marked as ANOVA without r-value APA). 'Other statistics' is a unspecified statistic type that includes a wide range of statistics (e. g., interquartile range (IQR) or Kolmogorov-Smirnov tests).

Table 4: Number of extracted statistics for APA and non-APA conform reporting on HCI papers. Separately considering into extraction from PDF and LaTeX files.

	APA conform		non-APA conform	
Statistic Type	PDF	LaTeX	PDF	LaTeX
Student's t-test	440	634	38	69
Pearson Correlation	48	65	76	94
Spearman Correlation	2	1	59	64
ANOVA	0	0	2	0
ANOVA without r-value	1, 059	1, 097	0	0
Mann-Whitney U	0	0	270	425
Wilcoxon Signed-Rank	0	0	0	0
Chi-Square	53	85	14	718
Z-Test	66	195	0	0
Other statistics	N/A	N/A	4, 194	4, 184
Total number of statistics	1, 668	2, 077	4, 653	5, 554

Using PDF files, about 26% of the extracted statistics were APA conform (9% when treating ANOVA without r-value as non-APA). With LaTeX files, 27% of extracted statistics were APA conform (13% without ANOVA without r-value).

Using the extracted HCI sentences, we evaluate the extraction manually using a sample of 200 sentences for every statistic type. If fewer than 200 sentences of a statistic were extracted, we use all extracted sentences. For statistic types without any samples, we cannot calculate precision.

On the PDF files, the extraction precision for APA statistics was 100% and ranged from 90% to 100% for non-APA statistics (see Table 5). Besides that, 'Other statistics' only had a precision of 54.5%. Similarly, using the LaTeX files, we achieved 100% precision for APA conform statistics and precision ranging from 89% to 100% otherwise. Additionally, 'Other statistics' also only achieved a precision of 60.5%.

The large deviation of the precision (Table 5) for 'Other statistics' compared to other statistic types can be traced back to a single R^+ rule. The rule in question is `\([P|p] \s? <?=? \s? \d (\.\d+)?\)`. This rule also captures the string `(P1)`, for example, which is not a reported statistic. When changing the rule to `\([P|p] \s? [<=]+ \s? \d (\.\d+)?\)` and re-running the evaluation, only 2, 337 'Other statistics' were extracted. However, the precision goes up to 97.5% for the PDF files. For the LaTeX files, 'Other statistics' extractions were reduced to 2, 254, with a precision increase to 98.5%.

For both PDF and LaTeX files, the falsely classified Mann-Whitney U tests were mostly Wilcoxon Signed-Rank tests and the falsely classified Pearson Correlations were mostly Spearman Correlations and vice versa.

CORD-19 Results. We ran the same evaluation on CORD-19, see Table 6 and 7. As expected, we achieve similar results as the original STEREO paper. The extracted sentences divided by statistic type can be seen in Table 6. Besides 'Other Statistics', non-APA conform Pearson Correlations were extracted the most, with a large margin. Of the extracted statistics, 1.8% were APA conform (0.8% when treating ANOVA without r-value as non-APA).

We evaluate the precision, again using 200 randomly selected sentences, see Table 7. As for the HCI dataset, the APA conform extractions achieved a precision of 100%. For non-APA conform statistics, the precision ranged from 94.5% to 100%.

Table 5: Precision of extracted statistics using HCI papers. We calculate the precision on 200 samples for every statistic type (or all extracted samples, if there are less). Statistic types with no extracted samples could not be calculated.

	APA conform		non-APA conform	
Statistic Type	PDF	LaTeX	PDF	LaTeX
Student's t-test	100%	100%	97.4%	100%
Pearson Correlation	100%	100%	96%	96.8%
Spearman Correlation	100%	100%	90.7%	89%
ANOVA	N/A	N/A	100%	N/A
ANOVA without r-value	100%	100%	N/A	N/A
Mann-Whitney U	N/A	N/A	92%	94%
Wilcoxon Signed-Rank	N/A	N/A	N/A	N/A
Chi-Square	100%	100%	100%	100%
Z-Test	100%	100%	N/A	N/A
Other statistics	N/A	N/A	54.5%	60.5%

Table 6: Number of extracted statistics for APA and non-APA conform reporting on CORD-19 papers. Note: The number of reported statistics deviate slightly from the original paper, due to untracked dataset changes.

Statistic Type	APA conform	non-APA conform
Student's t-test	662	210
Pearson Correlation	113	5, 034
Spearman Correlation	1	551
ANOVA	0	2
ANOVA without r-value	1, 239	0
Mann-Whitney U	2	419
Wilcoxon Signed-Rank	0	0
Chi-Square	69	58
Z-Test	103	0
Other statistics	N/A	114, 242
Total number of statistics	2, 189	120, 516

Table 7: Precision for extracted statistics using CORD-19 papers. We calculate the precision on 200 samples for every statistic type (or all extracted samples, if there are less). Statistic types with no extracted samples could not be calculated.

Statistic Type	APA conform	non-APA conform
Student's t-test	100%	97%
Pearson Correlation	100%	98.5%
Spearman Correlation	100%	100%
ANOVA	N/A	100%
ANOVA without r-value	100%	N/A
Mann-Whitney U	100%	94.5%
Wilcoxon Signed-Rank	N/A	N/A
Chi-Square	100%	100%
Z-Test	100%	N/A
Other statistics	N/A	98.5%

R^- rule evaluation. Moreover, we evaluate the R^- rules on both HCI as well as CORD-19, to test if any statistics were falsely matched by R^- rules. For this, we sample 200 sentences matched by an R^- rule from the original STEREO rule set in randomly selected papers from the respective paper corpus. In all scenarios, all 200 extracted sentences were correctly identified as non-statistics. Rerunning the experiment with the reduced R^- rule set, the 200 newly extracted sentences were also correctly identified as non-statistics. During this step, we also measure the runtime using the HCI dataset and compare the minimized regular expression set with the complete rule set. As every sentence in a paper is checked against all R^- rules (until a match is found), this evaluation provides a good baseline for future tasks involving the rule set. The full rule set takes 122.4 seconds (averaged over 5 runs), while the reduced rule set takes 84.5 seconds (averaged over 5 runs). This is only decrease of about 40 seconds and is not that relevant in the case of sampling 200 sentences. However, when extending this to a much larger test or using the reduced set on a larger corpus, this 30% difference is not negligible.

Table 8: Evaluation of sentences not covered by R^- or R^+ rules. Evaluated on a sample of 200 sentences taken from the respective datasets.

Dataset / File Type	False Positives	True Positives	parse error
CORD-19 / JSON	17	174	9
HCI / LaTeX	14	186	0
HCI / PDF	5	192	3

Lastly, we extract 200 sentences that contain a number but were not matched by any R^- or R^+ rules. We assess whether these uncaptured sentences report a statistic or not, or whether they contain a parsing error regardless of the content (see Table 8). 92% of uncaptured sentences did not contain any uncaught statistics. The most missed statistics (8.5%) were in CORD-19, while using the PDF files in the HCI domain missed the fewest (2.5%). Using the CORD-19 dataset also resulted in the most parsing errors. Using the LaTeX files did not result in any parsing errors.

6 DISCUSSION

6.1 Main Results

Inspecting reduced R^- rules. In our experiments, we apply the rule set inclusion algorithm to reduce our rule set and thus also improve the runtime performance on tasks using the new rule set by about a third. In Figure 7, we see that the later a rule was added, the likelier it was to include one or more other rules. As before, "later" references the point in time a rule was added during the active wrapper learning process and that a higher rule ID was assigned to it. Furthermore, in a deeper analysis (see Appendix B), we found that almost all inclusions were rules included by a later rule, suggesting no unnecessary rules were added when a match was already present. Some exceptions exist where a lower ID rule includes a rule with a higher ID. These few exceptions can be attributed to human error (e. g., adding a rule to the rule set explicitly and not due to a found sentence), as the rules in question are mostly duplicates. This finding makes sense, as later rules were added with more background knowledge and thus were more generalized. An example of a rule superseded by a rule created with more background knowledge is the most included rule **`figure \d{1,2}`**. Later rules like **`[fF]igure?\s\d+(\s?\s?\d+)*`** and **`(...| fig | figure | Table |...)\s*\d+(\s*[\.\,]\s*\d+)*`** (shortened) include the first rule and do not only match a figure, but also tables and equations.

The structure of rules, which include many other rules, is also mostly similar. We see that every rule, which included more than 100 other rules leveraged numbers being preceded or followed by a word (e. g., **`[a-zA-Z]{3,}`**). Other common patterns used often in rules were **`\d(\.\d+)?`** for a number with an optional decimal, **`\s?=\s?`** for optional spaces around a symbol, **`[mM]`** for the choice of the same letter capitalized or not (e. g., **`m`** for meters), and physical units being preceded by all possible SI [13] prefixes in a character class (e. g., **[µkmndcpfazyhMGTPEZY]?m** to capture meters with any (or none) prefixes).

LaTeX vs. PDF. In total, using LaTeX files yielded more statistics than using PDF files. In specific cases (Chi-Square Test and Spearman Correlation), using LaTeX files even extracted more samples than the CORD-19 dataset, even though CORD-19 is a much larger dataset. Regardless of the file format used for the input, the precision is generally satisfactory. Since all APA-conform statistics follow a very strict and well-defined pattern, it makes sense that they have a precision of 100%. However, non-APA Mann-Whitney U test rules need refinement, as in all scenarios, some Wilcoxon Signed-Rank tests were falsely identified as Mann-Whitney U tests.

Using the HCI dataset, of the extracted statistics, about 26% were APA conform. This is a large improvement to the 1.8% of APA conform statistics in the CORD-19 dataset, however, CORD-19 is a much larger dataset. Nonetheless, this means that the remaining 74% for HCI and 98.2% for CORD-19 of reported statistics are non-APA conform.

While using LaTeX as an input source did miss 14 statistics, we did not encounter any parsing errors, which were the case in both JSON and PDF. Due to this better performance and the option to easily remove environments, e. g., tables, figures and captions, or tikzpictures, as well as the simple parsing of math formulas and special symbols, LaTeX files are the best format to use for our statistic extraction pipeline.

6.2 Limitations and Threats to Validity

We acknowledge that when transforming the given regular expression, not all Python features for regular expressions can be transformed exactly. For example, lookahead assertions[7] or line beginning and ending symbols ($\wedge$ and \$) could not be represented in our formal definition exactly and were simply removed during preprocessing. However, all rules which ignored some parts and therefore were not exact transformations, were marked as such, and any potential inclusion pertaining to such a rule was checked manually. Not many inclusions involved such rules, however, of these inclusions, many had to be removed during the manual check, as the matches did not hold with the original Python regular expression.

Other than that, we extend the statistics extraction to the HCI domain, but our HCI paper corpus is very small ($4,023$ papers used) in comparison to CORD-19 ($> 110,000$ papers). Nonetheless, we were able to add some new statistic types and capture rules using a random sample of 200 of papers. Following this, one could criticize the active wrapper process itself, as only a very small portion of the corpus is used to learn rules. However, both in the original study as well as in our experiments, we achieve high precision on the datasets and formats.

6.3 Impact and Future Work

In future studies, the R^+ rules could be refined more. During our evaluation, we found many Wilcoxon Signed-Rank tests were falsely captured as Mann Whitney U tests. While Wilcoxon Signed-Rank and Mann Whitney U do have similar reporting styles, a clear distinction should be made, for example, by using the surrounding words.

Furthermore, some statistic types captured by 'Other statistics' (e. g., Kolmogorov-Smirnov tests or odds-ratio) could be separated and adapted to their own statistic type. This would more clearly show the distribution of reported statistics and potentially further improve precision and robustness.

Of course, the same procedure we used can be applied to more domains to find new statistic types, improve existing statistic types, and extend the R^- rules. In Appendix A, we offer some recommendations for future rule creation. Expanding the statistic types is also motivated by the 2-8% of missed statistics in a sample of 200 sentences (see Table 8). Now that the implementation has been modified, it can handle papers submitted in JSON, PDF, and LaTeX, which should account for the majority of input formats.

Moreover, one could adapt our regular expression inclusion algorithm to additionally check for sub-expression inclusion (i. e., if a regular expression is wholly included as a sub-expression of another regular expression). Using a simple example, the expression **abc** includes **bc** as a sub-expression. Although this does not directly help us to further reduce our rule set, we can use this for a similar extension. One could develop an algorithm that, given two regular expressions, automatically generates the smallest regular expression that accepts both expressions. With these modifications, sub-expression inclusion can be used to further reduce the rule set size. Our example with **abc** and **bc** can be merged as **a?bc**.

7 CONCLUSION

In this paper, we practically apply an algorithm to determine the inclusion of regular expressions. We employ the STEREO statistics extraction tool which extracts statistics from articles using a set of regular expressions. One third of the rules are removed when our inclusion algorithm is applied to the list of 1,510 regular expressions used by STEREO. Furthermore, we extend the rule set to the HCI domain. We repeat the active wrapper learning on a sample of 200 papers, adding 13 R^+ and 86 R^- rules. This is only a small fraction of newly required rules compared to the $1,510$ original rules in STEREO. We apply the extended statistics extraction rule set to the whole HCI dataset. We find that only 26% of extracted statistics were APA conform in the HCI domain. This is a large improvement compared to the $< 2\%$ achieved on CORD-19. However, still the majority of reported statistics in HCI are not reported in accordance with the APA style guide. Additionally, we compare the use of PDF and LaTeX files as an input. Due to better extraction precision and fewer parsing errors, we advise using LaTeX files.

For repeatability and further insights we provide our source code: https://github.com/Tobi2K/BachelorThesis

ACKNOWLEDGMENTS

We thank the authors of STEREO [8] for providing the source code, rule set, and their expertise. The original source code of STEREO can be found here: https://github.com/Foisunt/STEREO.

We also thank Carolin Schindler, Simon Birkholz, and Danial Podjavorsek for providing the crawler (https://github.com/Data-Science-2Like/arXive-crawler) we used to gather the arXiv papers.

REFERENCES

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Boston, Massachusetts. https://www.worldcat.org/oclc/12285707

[2] Madison Bentley, CA Peerenboom, FW Hodge, Edward B Passano, HC Warren, and MF Washburn. 1929. Instructions in regard to preparation of manuscript. *Psychological Bulletin* 26, 2 (1929), 57.

[7] https://docs.python.org/3/howto/regex.html#non-capturing-and-named-groups

[3] Ingmar Böschen. 2021. Evaluation of JATSdecoder as an automated text extraction tool for statistical results in scientific reports. *Scientific Reports* 11, 1 (2021), 1–12.

[4] Anne Brüggemann-Klein and Derick Wood. 1998. One-Unambiguous Regular Languages. *Inf. Comput.* 142, 2 (1998), 182–206. https://doi.org/10.1006/inco.1997.2695

[5] Haiming Chen and Zhiwu Xu. 2020. Inclusion algorithms for one-unambiguous regular expressions and their applications. *Sci. Comput. Program.* 193 (2020), 102436. https://doi.org/10.1016/j.scico.2020.102436

[6] Colin B. Clement, Matthew Bierbaum, Kevin P. O'Keeffe, and Alexander A. Alemi. 2019. On the Use of ArXiv as a Dataset. arXiv:1905.00075 [cs.IR]

[7] Edsger Wybe Dijkstra. 1961. Algol 60 translation: An Algol 60 translator for the x1 and Making a translator for Algol 60. *Stichting Mathematisch Centrum. Rekenafdeling* -, MR 34/61 (1961).

[8] Steffen Epp, Marcel Hoffmann, Nicolas Lell, Michael Mohr, and Ansgar Scherp. 2021. STEREO: A Pipeline for Extracting Experiment Statistics, Conditions, and Topics from Scientific Papers. In *The 23rd International Conference on Information Integration and Web Intelligence* (Linz, Austria) *(iiWAS2021)*. Association for Computing Machinery, New York, NY, USA, 340–349. https://doi.org/10.1145/3487664.3487712

[9] Leonard P Freedman, Gautham Venugopalan, and Rosann Wisman. 2017. Reproducibility2020: Progress and priorities. *F1000Research* 6 (2017), 21 pages.

[10] Dag Hovland. 2012. The inclusion problem for regular expressions. *J. Comput. Syst. Sci.* 78, 6 (2012), 1795–1813. https://doi.org/10.1016/j.jcss.2011.12.003

[11] Sree Sai Teja Lanka, Sarah Michele Rajtmajer, Jian Wu, and C. Lee Giles. 2021. Extraction and Evaluation of Statistical Information from Social and Behavioral Science Papers. In *Companion of The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, Ljubljana, Slovenia, 426–430. https://doi.org/10.1145/3442442.3451363

[12] Simon M. Lucas and T. Jeff Reynolds. 2005. Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.* 27, 7 (2005), 1063–1074. https://doi.org/10.1109/TPAMI.2005.143

[13] David Newell and Eite Tiesinga. 2019. The International System of Units (SI), 2019 Edition. https://doi.org/10.6028/NIST.SP.330-2019

[14] Tobias Nipkow and Dmitriy Traytel. 2014. Unified Decision Procedures for Regular Expression Equivalence. In *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8558)*, Gerwin Klein and Ruben Gamboa (Eds.). Springer, Vienna, Austria, 450–466. https://doi.org/10.1007/978-3-319-08970-6_29

[15] Michèle B Nuijten, Chris HJ Hartgerink, Marcel ALM Van Assen, Sacha Epskamp, and Jelte M Wicherts. 2016. The prevalence of statistical reporting errors in psychology (1985–2013). *Behavior research methods* 48, 4 (2016), 1205–1226.

[16] Michele B Nuijten, Marcel A L M van Assen, Chris H J Hartgerink, Sacha Epskamp, and Jelte M Wicherts. 2017. The Validity of the Tool "statcheck" in Discovering Statistical Reporting Inconsistencies. https://doi.org/10.31234/osf.io/tcxaj

[17] PsychOpen. 2017. PsychOpen uses Statcheck tool for quality check. https://www.psychopen.eu/news/article/psychopen-uses-statcheck-tool-for-quality-check/

[18] M. O. Rabin and D. Scott. 1959. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development* 3, 2 (1959), 114–125. https://doi.org/10.1147/rd.32.0114

[19] Rohit Rastogi, Pinki Mondal, and Kritika Agarwal. 2015. An exhaustive review for infix to postfix conversion with applications and benefits. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, India, 95–100.

[20] John K Sakaluk and Cynthia A Graham. 2018. Promoting transparent reporting of conflicts of interests and statistical analyses at the Journal of Sex Research. , 6 pages.

[21] Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Cambridge University Press, England. http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521844253

[22] Thomas Schmidt. 2017. Statcheck does not work: All the numbers. Reply to Nuijten et al. (2017). https://doi.org/10.31234/osf.io/hr6qy

[23] Ken Thompson. 1968. Regular Expression Search Algorithm. *Commun. ACM* 11, 6 (1968), 419–422. https://doi.org/10.1145/363347.363387

[24] Dominika Tkaczyk, Pawel Szostek, Mateusz Fedoryszak, Piotr Jan Dendek, and Lukasz Bolikowski. 2015. CERMINE: automatic extraction of structured metadata from scientific literature. *Int. J. Document Anal. Recognit.* 18, 4 (2015), 317–335. https://doi.org/10.1007/s10032-015-0249-8

[25] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Chris Wilhelm, Boya Xie, Douglas Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. CORD-19: The Covid-19 Open Research Dataset. *CoRR* abs/2004.10706 (2020), 11 pages. arXiv:2004.10706 https://arxiv.org/abs/2004.10706

[26] Bruce Watson. 1993. *A Taxonomy of Finite Automata Construction Algorithms*. Citeseer, Pennsylvania, United States.

Supplementary Materials

A LESSONS LEARNED

On the basis of our analysis of the rule set, we offer recommendations on how rules should be created in the future.

This section is not a complete set of guidelines one should follow, but rather a collection of common patterns and recommendations we learned during rule creation and analysis.

A.1 General

In general, it is important that rules are not too generic, as these will produce many false positives. This applies to both R^+ and R^- rules. For example, an R^+ rule capturing any equals sign followed by a number is too broad. Almost all statistics would be captured by this rule. However, this prevents correct value extraction and a differentiation between statistic types is not applicable, defeating the purpose of extracting specific statistic types. However, the rules should not be too specific either. Although this does not produce false positives, rules that only capture a single case worsen maintainability and usability, as every case would need a new rule. We show some patterns in the following, with which one can easily create good rules.

A.2 Leverage Contextual Information

As can be seen in Table 3, most of the rules that include many others, leverage being preceded or followed by a word. This context can be used to rule out a sentence or number as a statistic, as statistics usually use a maximum of 2 letters as an identifier (e.g. `p` or `t`). So capturing a number following a word, especially without an operator, rules out that the sentence contains a statistic.

On the other hand, the context can also be used to capture (non-APA) statistics. For example, some existing R^+ rules capture a sentence containing "significance" followed by a p-value. Significance values usually only report a p-value, however the simple pattern `p=\d(\.\d+)?` also captures all p-values reported in any other statistic. To prevent this, rules can be limited by only capturing the p-value following the word "significance". An example of such a rule is `significantly higher[\sa-zA-Z]+ \(\s?[pP]\s?<\s?0?\.\d+\s?\)`. Here, "significantly higher" can be followed by some other letters and then the p-value in parentheses afterwards.

A.3 Use Optional Sequences

We already mentioned some of the following patterns which use optional sequences:

- `\d(\.\d+)?` for a number with an optional decimal
- `\s?=\s?` for optional spaces around a symbol
- `[mM]` for the choice of the same letter capitalized or not
- `[µkmndcpfazyhMGTPEZY]?m` to capture physical units being preceded by all possible SI [13] prefixes
- `[<>=≤≥]` (or a subset) to capture mathematical operators
- `[a-zA-Z\s]+` to capture a word or words. Can also be modified:
  - `[a-zA-Z\s]{,20}` to capture up to 20 letters or white spaces
  - `[a-zA-Z\s]{20,}` to capture at least 20 letters or white spaces

These patterns can combat different formatting or spelling mistakes and should always be used, unless there is an explicit reason not to. Using these patterns also prevents excess rule creation.

A.4 Grouping Similar Structures

Numbers are often presented in a similar context. A simple example is the numbering of tables, figures, algorithms, etc. Following Section A.2, we can use this context and further combine these into a single rule. A shortened example is `(Fig. | Figure | fig | figure | Table | table | TABLE | FIGURE) -? \s? \d+ (\s [.,] \s \d+)?`. This rule captures figure and table references in multiple formats. For example, `Table 1`, `FIGURE 2,1`, `fig-3.6`, and more, are all captured. Another non-statistic example of this being applied to are page number references. Here we group the options `(pages|page|pp\.|pp)` in one rule.

This grouping can also be applied to capturing statistics. For example, STEREO uses a rule to capture Mann-Whitney U tests that requires the test statistics to be preceded by the name "Mann-Whitney U". To be more lenient, many options are possible: `(Mann Whitney U test|MWU test| mwu test|Mann-Whitney test)`.

A.5 Greek Letters and PDF Formatting

Furthermore, it is advisable to keep different formats in mind when creating rules. An example used in STEREO is the Chi-Square test. In CORD-19, the used documents always converted a χ to an x or X. Therefore, rules capturing Chi-Square tests only used the capture group `[xX]`. However, LaTeX files use χ to create the symbol χ symbol. The LaTeX conversion we use can interpret this command directly and convert the symbol as a Unicode symbol (U+03C7), instead of an X. Thus, we advise to add Greek letters as an option, if applicable, especially for R^+ rules.

Moreover, many scientific papers are written using LaTeX and converted to PDF. Some characters undergo specific formatting during this conversion. A common occurrence is using a hyphen. Depending on the formatting and font, a hyphen can be written as - or – (Unicode: U+002D vs. U+2013). Other less common hyphens are — (U+2014) and − (Math-mode minus, U+2212). When writing a rule containing a hyphen, e. g., ranges of numbers, we recommend using a capture group containing at least the first two (`[-–]`), if not all four (`[-–—−]`) alternatives.

B DEEPER ANALYSIS

This section presents some further research conducted.

B.1 Inclusion comparison by rule ID

In Figure 7, we show the amount of rules removed, grouped by rule ID. Figure 8 shows a more detailed view, where we split rules. We see, that almost all rule inclusion were a higher rule ID including a lower rule ID. Some exceptions exist, however these are mostly duplicates.

B.2 Extraction with original rules

To understand how well the original rules transfer to a new domain, we reran the extraction on HCI without the rules we added during active wrapper learning. During active wrapper learning we added rules for t-test (APA and non-APA), Pearson Correlation (APA and non-APA), Spearman Correlation (non-APA), Mann-Whitney U (non-APA), and Chi-Square (APA and non-APA). These differences can be seen in Table 9 in comparison to Table 4.

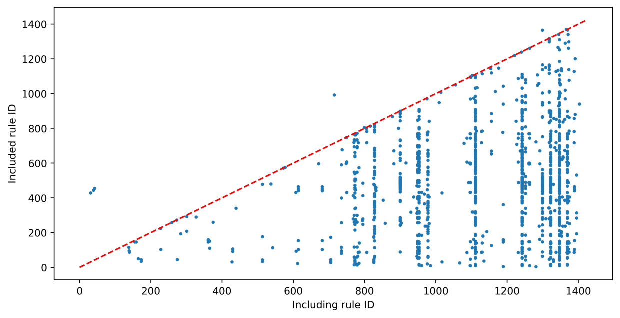


Figure 8: A comparison of rules with their respective inclusions by rule ID. The rule ID represents the order in which rules were added in STEREO [8]. The red straight line shows the angle bisector, where every rule would include themselves. The x-axis defines the including rule ID, i. e., the rule that included others, while the y-axis shows the rule IDs of rules that were included.

Table 9: Number of extracted statistics for APA and non-APA conform reporting on HCI papers using only original rules. Split into extraction from PDF and LaTeX files.

	APA conform		non-APA conform	
Statistic Type	PDF	LaTeX	PDF	LaTeX
Student's t-test	377	634	18	38
Pearson Correlation	42	65	67	94
Spearman Correlation	2	1	51	64
ANOVA	0	0	5	0
Mann-Whitney U	0	0	34	42
Wilcoxon Signed-Rank	0	0	0	0
Chi-Square	1	0	0	0
Other statistics	N/A	N/A	4, 124	4, 141
Total number of statistics	422	700	4, 299	4, 379

C THOMPSON'S CONSTRUCTION

See Figure 9. The colored circles represent the default automata used in Thompson's construction, cf. Figure 1.

D UNOPTIMIZED AUTOMATON INCLUSION

Algorithm 3 is the original, unoptimized version of Algorithm 2 presented by Chen and Xu [5].

Algorithm 3: Automaton inclusion as proposed by Chen and Xu [5]

```
Data: Two DFAs $A1$ and $A2$
Result: Boolean whether $A2 \subseteq A1$
1 $A' \leftarrow computeComplement(L(A1))$ ;          // Step 2
2 $B \leftarrow createByLanguage(L(A2) \cap L(M'))$ ;  // Step 3
  /* Step 4 */
3 Construct Graph $G = (Q_B \times \delta_B)$;
  /* $Q_B$ is equivalent to $Q_{A2} \times Q_{A'}$ */
4 if $G$ contains path from start state to accepting state then
5     return $FALSE$;
6 else
7     return $TRUE$
8 end
```

E INFORMATION ON ARXIV PAPERS

Of the original 9, 730 papers tagged with "cs.HC", 6, 110 papers had "cs.HC" as the primary tag, of which 5 were inaccessible on the date of access (2022-07-11). This usually means that papers have been withdrawn by the authors or were removed from arXiv. Of these 6, 105 papers, 2, 037 were only available as PDF, 4, 067 had source code, i. e., LaTeX files, and one paper was only available as a `.docx` document. We only use papers that were provided as both LaTeX and PDF files. This is needed for a fair comparison of the statistics extraction on both file types. 44 papers given as source code did not have PDF files, which means 4, 023 papers remain.

F CHANGED RULES

While learning new rules for the HCI domain, we changed six previous R^- rules. These changes were made, as we had knowledge of previously added rules. We only broadened rules, allowing more cases, for example additional white spaces. The following changes have been made:

- Allow *2D* and *3D* with both capital and lowercase letter
  `\s[2|3]D\s`
  $\longrightarrow$
  `\s[2|3][d|D]\s`
- Allow *2D* and *3D* with both capital and lowercase letter
  `\s[2|3]D\d\s`
  $\longrightarrow$
  `\s[2|3][d|D]\d\s`
- Allow any white space instead of only spaces
  `\s\d\) [a-zA-Z]+`
  $\longrightarrow$
  `\s\d\)\s{1,2}[a-zA-Z]+`
- Allow additional operators
  `[a-zA-Z]\s?[a-zA-Z]\s?[a-zA-Z]\s?[a-zA-Z]  \s? [a-zA-Z]\s?[≈=]\s?\d+`
  $\longrightarrow$
  `[a-zA-Z]\s?[a-zA-Z]\s?[a-zA-Z]\s?[a-zA-Z]  \s? [a-zA-Z]\s?[≈=>]\s?\d+`
- Allow additional white spaces
  `\d+(\.\d+)?,\s[a-zA-Z]{3,}`
  $\longrightarrow$
  `\d+(\.\d+)?,\s+[a-zA-Z]{3,}`
- Allow optional negative values in lists
  `\)\s?(,|or)?\s?\d\s?\(`
  $\longrightarrow$
  `\)\s?(,|or)?\s?-?\d{1,2}\s?\(`

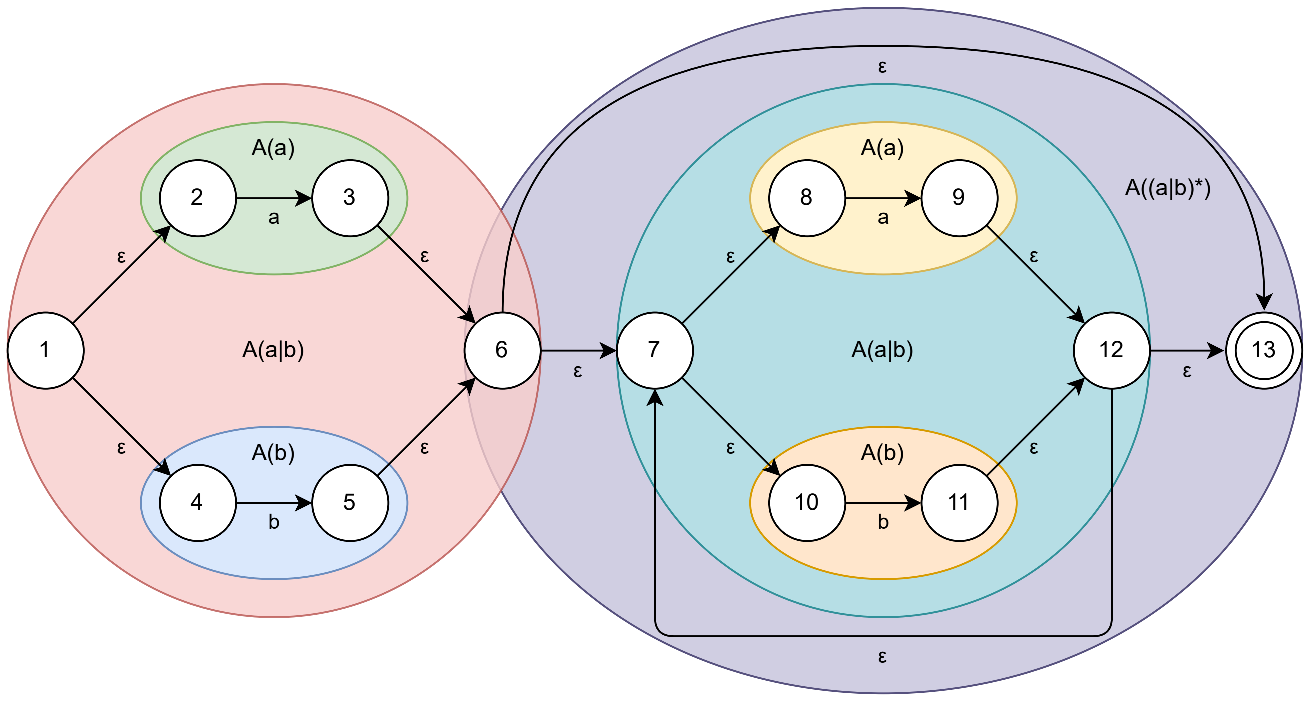


Figure 9: Larger image of Thompson's construction for `ba|ab|*&` (originally `[a-b](a|b)*`)