An Approach Towards Merging Grammars

Isaac D. Griffith and Rosetta Roberts
Empirical Software Engineering Laboratory
Informatics and Computer Science
Idaho State University
Pocatello, Idaho, 83208
Email: {grifisaa@isu.edu, roberose@isu.edu}

***Abstract*—Introduction: Since the introduction of Island Grammars, they have been successfully used for a variety of tasks, including impact analysis, multi-lingual parsing, and source code identification. However, there has been no attempt to automate the generation of Island Grammars. Objective: This research considers the development of a method to automate the merging of Island Grammar components. The goal of this is to facilitate the development of an approach to automate the creation of Island Grammars. The result of this is the reduction in initial effort and maintenance effort required for Island Grammar engineering. Methods: We develop an automated approach to merge the components of grammars. To evaluate this approach, we conducted two experiments, each using a factorial design of five replications each. We randomly selected pairs of grammars from each of three size categories to assess the effects of the merging process on the maintenance effort and complexity of the generated grammars. Results: We found that in nearly all cases, the application of this merging approach reduces the maintenance effort and complexity of the grammars. Limitations: The primary limitation of this research is that this approach is currently limited to grammars written in the Antlr4 grammar format. Conclusions: This work presents the initial steps towards the automated construction of Island and Tolerant Grammars. We have shown that this approach to merging grammar components follows suit with the expectations of Island and Tolerant grammars (reduction in maintenance effort and complexity).**

***Index Terms*—Island Grammars, Automated Grammar Formation, Software Language Engineering**

I. Introduction

Modern software development practice has led to an increasing number of software systems created using multiple languages [1]. As an example, a modern web application typically consists of five or more languages (e.g. SQL, Java, TypeScript, HTML, CSS). Such multi-lingual codebases present a difficult challenge to the development and maintenance of source code analysis tools [1]. These tools typically address this challenge using a combination of multiple parsers (one per supported language). Constructing and integrating each parser for each language can be both difficult and time-consuming, and thus it is better to use existing parsers [2].

An alternate solution to this problem is Island Grammars, which allow the creation of multi-lingual parsers [3]. These parsers provide an elegant, more efficient solution than the application of existing parsers. Currently, they require the manual combination of selected components from source grammars [3], a process that can be cumbersome to maintain as these grammars evolve. To overcome this challenge, we develop a crucial part of the automated construction of Island Grammars through an approach to merging multiple grammars.

Underlying this approach is the following hypothesis: *Automated grammar merging is an important and necessary step in the evolution of Island Grammar research. It provides the capability to address difficulties in the initial construction and further maintenance of Island Grammars.* To evaluate this hypothesis we construct Goal-Question-Metric (GQM) [4] hierarchy starting with the following research goal (RG):

RG Evaluate an automated approach for the purpose of automating the merging of grammar rules with respect to the maintenance effort and complexity from the point of view of software language engineers in the context of the creation of Tolerant and Island Grammars.

This goal is further refined in Sec. IV.

This paper is organized as follows. Sec. II discusses the theoretical foundations and alternative approaches related to this work. Sec. III details our approach to automate the merging of grammars. Sec. IV details the design of experiments that evaluate the proposed approach. Sec. V presents the results of the experiments and their analysis and interpretation. Sec. VI details the threats to the validity of this study. Finally, Sec. VII concludes this paper with a summary and description of future work.

II. Background and Related Work

A context free grammar, G, can be described as $G = (V, \Sigma, P, S)$ [5]. Where, V is the set of non-terminal symbols, Σ is the set of terminal symbols, $P \subseteq V \times (V \cup \Sigma)^*$ is the set of productions describing how the symbols of V can be substituted for other symbols, and $S \in V$ is the starting symbol. Each production is written as $a \rightarrow b$ with $a \in V$ and $b \in (V \cup \Sigma)^*$. When b is the empty string, the production is denoted by $a \rightarrow \varepsilon$. A string, s, is called a *valid sentence* for a grammar if it can be created by repeated application of the productions of that grammar [6]. $L(G)$ denotes the set of all valid sentences, or language, of grammar G.

Island Grammars are a specialized form of context free grammar, which includes the addition of a set of interest productions. These focus the grammar on the set of language components of interest to the grammar developers. An Island Grammar is formally defined as the following tuple $G' = (V, \Sigma, P, S, I)$ [6]. Where I is the set of interests or

978-1-7281-4291-3/20/$31.00 ©2020 IEEE

islands. The remaining components of a language reduce to one or more catchall productions referred to as water [6], thus having the effect of reducing the complexity of the grammar. An Island Grammar, G', derived from a grammar, G, has a language $L(G')$ which satisfies the following property: $L(G') \supset L(G)$.

These properties of Island Grammars offer several advantages. These advantages include faster development time, lower complexity, and better error tolerance necessary for several applications, including: documentation extraction and processing [7], impact analysis [8], and extracting code embedded in natural language documents [9], [10]. Of particular interest to our research is their use for creating multilingual parsers [3], which inspired this research, and research into the development of Tolerant Grammars [11]–[13].

III. Approach

The following describes the approach for merging similar productions of a grammar. Initially, this approach assumes that the grammar has been normalized such that each production in the grammar is one of the following two forms:

Form$_1$: A production composed of a rule containing at least one symbol, and where all symbols are concatenated with one another. An example is: $\langle A \rangle ::= \langle B \rangle$ ‘a’ . . ..

Form$_2$: A production composed of an alternation of one or more symbols. An example is: $\langle B \rangle ::= \langle A \rangle \mid$ ‘b’ $\mid \ldots \mid \varepsilon$.

Algorithm 1 embodies this approach. The algorithm initially separates productions of the normalized grammar, $\mathcal{N}$, based on production form into two disjoint sets (line 2). Within each set, pairs form to compare production similarity (line 3). The algorithm compares pair similarity to a known threshold value, t. If the similarity is less than t, then the pair is disregarded. Otherwise, the algorithm offers it to a priority queue (sorted on similarity). The following describes the pair similarity calculation defining the “Measure” function.

For each pair of productions, we measure a form-dependent similarity score. For *Form*$_1$ production, equation (1) measures similarity as the ratio of the cardinality of aligned production symbols to the total size of the productions. For *Form*$_2$ productions, equation (2) measures similarity as the ratio of the cardinality of the intersection of production symbols to the total size of productions and accounts for common symbols.

$$S_1 = \frac{2|LCS(A,B)|}{|A|+|B|} \tag{1}$$

$$S_2 = \frac{2|A \cap B|}{|A|+|B|} \tag{2}$$

In (1) the function “LCS” aligns rules using a dynamic programming solution to the longest common subsequence problem [14]. This function then returns the length of the longest common subsequence. Finally, the algorithm doubles this value to account for the size across both productions.

Algorithm 1 Merge Algorithm

```
 1: procedure MergeProductions($\mathcal{N}, t$)
 2:     $f_1 \leftarrow collect(\text{‘form1’}, \mathcal{N})$; $f_2 \leftarrow collect(\text{‘form2’}, \mathcal{N})$
 3:     $p_1 \leftarrow$ Pairs($f_1$); $p_2 \leftarrow$ Pairs($f_2$)
 4:     $\mathcal{Q}_1 \leftarrow$ Measure($p_1, t, S_1$)
 5:     $\mathcal{Q}_2 \leftarrow$ Measure($p_2, t, S_2$)
 6:     Process($\mathcal{Q}_1, f_1, S_1$); Process($\mathcal{Q}_2, f_2, S_2$)
 7: function Process($\mathcal{Q}, data, func$)
 8:     while $\mathcal{Q} \neq \varnothing$ do
 9:         $p \leftarrow$ Poll($\mathcal{Q}$)
10:         $\mathcal{V} \leftarrow$ Merge($p.pair$)
11:         $\mathcal{Q} \leftarrow$ Update($\mathcal{Q}, \mathcal{V}, p, data, func$)
12: function Merge($pair$)
13:     $rule \leftarrow \varnothing$
14:     if $pair$ is of form1 then
15:         $list \leftarrow$ alignPair($pair$)
16:         for all $p \in list$ do
17:             if $p.left = p.right$ then
18:                 $rule \leftarrow rule \cdot p.left$
19:             else
20:                 $rule \leftarrow rule \cdot (p.left | p.right)$
21:     else
22:         $v \leftarrow \varnothing$
23:         $v \leftarrow pair.right.symbols \cup pair.left.symbols$
24:         $rule \leftarrow concat(v)$
25:     $\mathcal{V} \leftarrow$ Normalize($rule$)
26:     return $\mathcal{V}$
```

Once the “Measure” function returns the priority queue containing the pairs of productions to merge (lines 4 and 5), the algorithm process each queue. This process polls the most similar pair from the queue to merge and form a new set of normalized productions (lines 7–11). These normalized productions update the queue based on the appropriate similarity scoring function. This process continues until the priority queue is empty. The following describes the “Merge” and “Update” functions.

This “Merge” function defines a form-dependent process used to merge a production pair. $Form_2$ pairs, the simplest case, merge by forming a new production containing an alternation composed of the union of the two productions’ symbols (lines 21–24). $Form_1$ pairs merge using a process similar to $Form_1$ similarity scoring (lines 14–15). The “alignPair” function aligns the symbols from each production using the LCS function from (1), reusing the original table created during measurement. This produces an optimal alignment, but in cases with multiple optimal alignments one is arbitrarily selected.

Once aligned, the algorithm forms a new production forms with an empty rule. The rule forms by concatenating aligned pairs (lines 16–20). In the case that symbols differ, the differing portions in each list are concatenated together, and the contents from each list are joined to form an alternation. Otherwise, it uses the symbol from only one list. Once merged, the new production is normalized (which may produce multiple new productions in the case of $Form_1$ productions), and the normalized results returned.

Once a pair of productions have been merged and normalized, the priority queue and grammar must be updated (line 11). The update process works as follows — initially, the data set of merged productions updates by removing the merged pair. Next, the priority queue updates by removing all pairs in the queue containing either of the merged productions — next, the grammar updates to replace all uses of the merged productions with the newly merged production. Finally, the merged production pairs with other productions of the same form, and the similarity between pairs is measured. Each pair, with similarity above the threshold, are added to the priority queue.

The outcome of this algorithm is a normalized grammar composed of the input grammars, where those productions with a similarity of at least t combine into single productions. Where the resulting language, $L_{G'}$, of the merged and normalized grammar, G' has the following property: $L_{G'} \supseteq L_G$. Where L_G is a language of input grammar G, and this property holds across all input grammars.

IV. Experimental Design

This section describes the overall experimental design used to evaluate the grammar merging approach presented in this paper. Initially, we refine the research goal, defined in Sec. I, into a set of actionable research questions (**RQ**) and metrics (**M**), according to the GQM process, as follows:

RQ1 What effect does merging grammars have on maintenance effort?

RQ2 What effect does merging grammars have on complexity?

M1 ΔHAL – the change in Halstead Effort measured, using the Halstead Effort measure for grammars defined by Power and Malloy [15], after the normalization phase and after the final merge phase.

M2 ΔMCC – the change in complexity measured, using the McCabe Cyclomatic Complexity metric for grammars defined by Power and Malloy [15], after normalization phase and after the final merge phase.

This decomposition leads to the identification of the experiments' dependent and independent variables. The dependent variables are ΔHAL and ΔMCC. The independent variables are:

- Similarity Threshold – the parameter that guides the similarity measurements used in the merging process. The values used in the experiments are 0.001 (control), 0.25, 0.5, 0.75, and 1.0.
- Size – the category of the grammar as defined by a statistically thresholded measure of the number of productions (PROD) [15]. Possible values are Small, Medium, and Large.

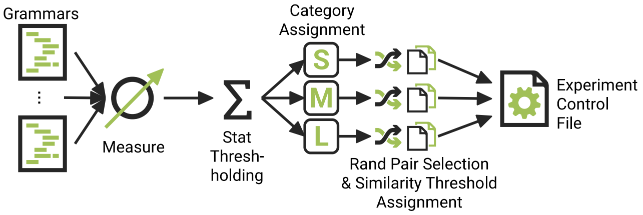


Fig. 1. Experimental unit selection process.

To evaluate the approach, we conduct two experiments. The first evaluates the effect of merging on the maintenance effort required for a merged grammar as compared to its combined source grammars. The second evaluates the effect merging has on the complexity of the merged grammar as compared to its combined source grammars. Both experiments use a Factorial Design, with a single dependent variable (ΔHAL and ΔMCC, respectively), a single treatment factor $SimilarityThreshold$ and the grouping factor $Size$.

The experimental units are pairs of grammars selected from the Antlr4 [16] grammar repository[1]. At the time of this writing, the repository contained 198 individual grammars from a variety of general-purpose and domain-specific languages. Grammar pair selection, for each experiment, is depicted in Fig. IV and works as follows. Initially, for each grammar in the repository, we collected a combination of metadata and metric measurements. The metadata collected consists of the language represented by the grammar, the version of that language (if applicable) and the following metrics (selected from the metrics suite by Power and Malloy [15]): TERM – the number of terminals, VAR – the number of defined nonterminals, PROD – the number of productions, and MCC – McCabe's Cyclomatic Complexity.

The resulting metrics measures subdivided the grammar dataset into three categories (Small, Medium, and Large) based on the logarithm of PROD (as the values are lognormal distributed), using statistically constructed thresholds [17]. Category threshold values are defined as: Small-Medium: 20.1084 productions and Medium-High: 238.4995 productions. These categories form the groups from which we select experimental units.

The experiments use a 3×5 factorial design and require 15 grammar pairs (5 per size category) per replication. A replication analysis identified a need for 5 replications, therefore a total of 25 grammar pairs per experiment. Thus, we selected (without replication) 12 grammars, yielding $\binom{12}{2}$ = 66 combinations of which we randomly select 5 pairs per replication per experiment. Tab. I shows the selected grammars.

The metadata and selected grammars combine into an experiment control file (one per replication). This file, containing a randomized set of triples (a grammar pair, the similarity threshold value, and size category), directs the experimental execution system and ensures process validity.

The experimental execution system executes the data collection process across each experimental unit, as depicted in Fig. IV, as follows. Initially, the system reads in the control

[1] https://github.com/antlr/grammars-v4

TABLE I
GRAMMARS RANDOMLY SELECTED FROM EACH SIZE CATEGORY USED IN THE EXPERIMENTS.

Category	Grammars
S	brainfuck, cmake, csv, inf, lcc, pdn, tsv, url
	quakemap, sexpression, properties, useragent
M	cto, dart2, flatbuffers, fusion-tables, lua
	pascal, python2, romannumerals, stacktrace
	webidl, sgf, z-ops
L	cql3, edif300, fortran77, idl, informix, java9
	kotlin, rexx, sharc, swift2, objc-two-step
	powerbuilder

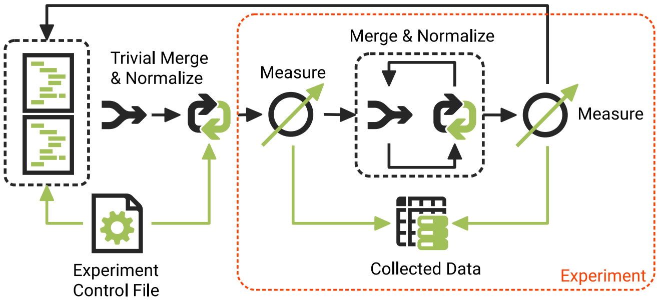


Fig. 2. Data collection procedure.

file. For each file entry (experimental unit), the following occurs. 1.) The selected similarity threshold value is applied. 2.) The grammars are located, read in, and trivially merged. 3.) The combined grammar's effort or complexity is measured and recorded. 4.) The final merging process is applied, resulting in the final grammar. 5.) The resulting grammar's effort or complexity is measured and recorded.

Once the data collection process completes, the system exports the results to a data table. This cycle repeats for each replication of each experiment. Finally, the results of each replication combine into a single data table used in the analysis phase.

As described in Sec. IV, both experiments utilize a Factorial design [18]. Typically, a factorial design uses an ANOVA, but due to significant violations of the ANOVA assumptions, we instead use a permutation F-test [19]. The statistical model for the permutation F-Test is as follows:

$$y_{ijk} = \mu + st_i + size_j + (st * size)_{ij} + \epsilon_{ijk}$$

Where, y_{ijk} is the kth value of the observation (either ΔMCC or ΔHAL) associated with the ith similarity threshold level and jth size level, μ is the baseline mean, st_i is the ith level of similarity threshold effect, $size_j$ is the jth level of size effect, $(st * size)\_ij$ is the interaction effect due to similarity threshold and size, and ϵ_{ijk} is the random error of the kth observation from the (i, j)th cell. We test the following null hypotheses using this model:

$H_{1,0}$: The effects of interaction term levels are equal.
$H_{2,0}$: The effects of similarity threshold levels are equal.
$H_{3,0}$: The effects of size levels are equal.

For each of these tests, we have selected an α threshold of 0.95. In the case that we reject $H_{1,0}$ (at an α level of 0.95), we conduct a multiple-comparison procedure to compare the individual effects of each level of the similarity threshold factor. We have selected to use Steel's [20] non-parametric multiple comparison procedure. This procedure corrects for multiple comparisons and compares each treatment against a control (similarity threshold = 1.0). For this test, we will be evaluating the following null hypothesis (at α threshold of 0.95):

$H_{4,0}$: There is no difference between the median effects of similarity threshold effects and control effect.

Additionally, we are interested if there is a strict order of the effect on ΔHAL or ΔMCC for the levels of the similarity threshold. To evaluate this, we have selected to utilize the Jonhckheer's trend test [21]. Jonhckheer's trend test is a non-parametric test to determine if there is an *a priori* ordering within independent samples. The null hypothesis to be tested is as follows:

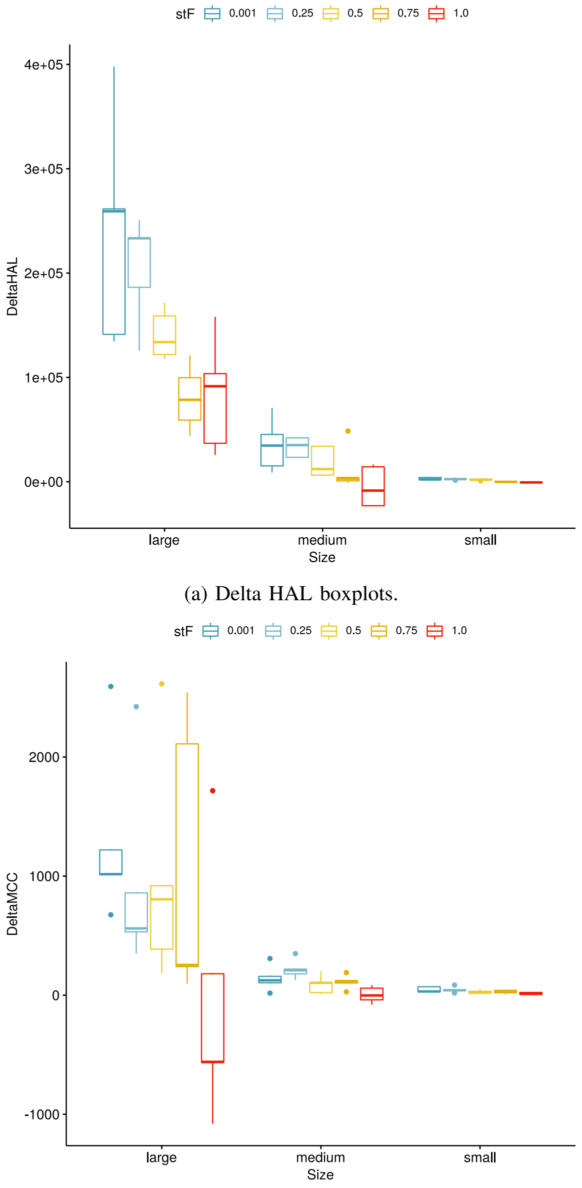
$H_{5,0}$: There is no difference in the median effects of the similarity threshold levels.

The following section discusses the results of these experiments and statistical tests.

V. RESULTS AND DISCUSSION

This section describes the results of the ΔHAL and ΔMCC experiments. Initially, we explored the dispersion of both ΔHAL and ΔMCC across size and similarity threshold values, a displayed in Fig. 3a and Fig. 3b, respectively. Both graphs show very little variability in either ΔHAL or ΔMCC when the size is small. Of note, is the apparent decreasing trend across size and within the size, as similarity threshold increases for ΔHAL, a trend not apparent within the plot for ΔMCC. With this in mind, we now turn to the results of the experiments.

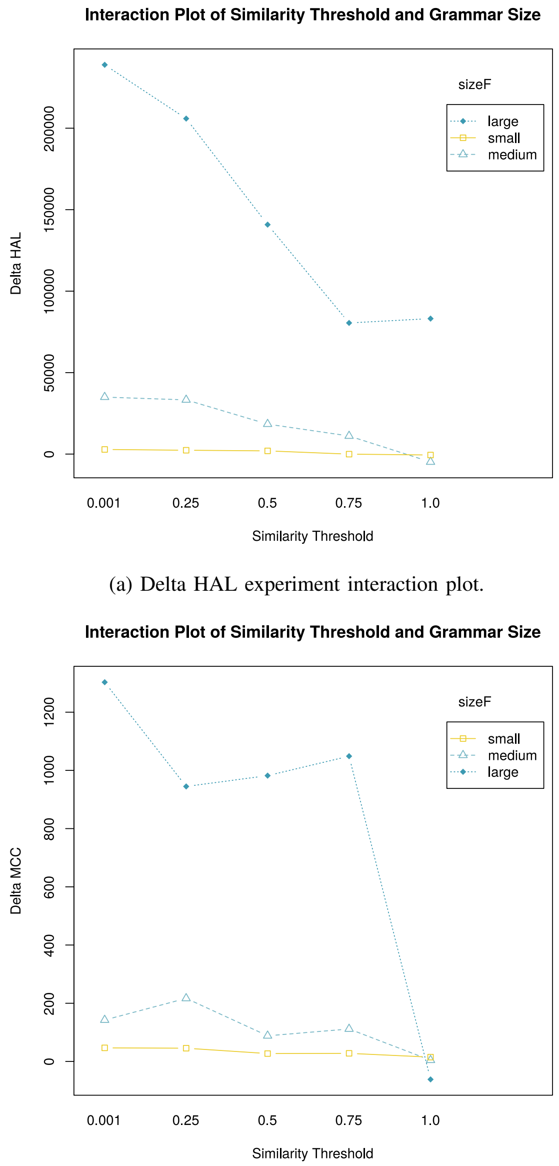
The results of the ΔHAL experiment are as follows. There is strong evidence (p-val < 2.2e-16) indicating an interaction between the grammar size and similarity threshold, as depicted in Fig. 4a, rejecting $H_{1,0}$. Furthermore, there is strong evidence (p-val < 2.2e-16) that not all levels of the similarity threshold have an equal effect when controlling for other sources of variability, rejecting $H_{2,0}$. Additionally, there is strong evidence (p-val < 2.2e-16) that not all levels of size have the same effect, rejecting $H_{3,0}$. Upon further comparison to control, all levels of the similarity threshold showed strong evidence of a greater effect on ΔHAL, rejecting $H_{4,0}$. Finally, there is strong evidence (JT-stat = 767, p-val = 6e-04) of a decreasing order in effect on median ΔMCC coinciding with an increase in the similarity threshold level, rejecting $H_{5,0}$. This evidence suggests that as the similarity threshold increases, the value of ΔHAL decreases. Such a result is indicative that the merging process reduces the maintenance effort as the similarity threshold increases.



(a) Delta HAL boxplots.

(b) Delta MCC boxplots.

Fig. 3. Boxplots for the ΔHAL and ΔMCC experiments



(a) Delta HAL experiment interaction plot.

(b) Delta MCC experiment interaction plot.

Fig. 4. Interaction plots for the ΔHAL and ΔMCC experiments

The results of the ΔMCC experiment are as follows. There is marginal evidence (p-val = 0.2534) indicating an interaction between the grammar size and similarity threshold, as depicted in Fig. 4b, a failure to reject $H_{2,0}$, thus the interaction was removed from the model. Furthermore, there is moderate evidence (p-val = 0.141) that there is a difference in the effects of the levels of similarity threshold when controlling for other sources of variability, reject $H_{2,0}$. Additionally, there is strong evidence (p-val < 2e-16) that there is a difference in the effects of the levels of size, reject $H_{3,0}$. Upon further comparison to control, similarity thresholds 0.25, 0.5, and 0.75 show strong evidence of a greater effect on complexity than control, but there is no evidence for this at the 1.0 level. Finally, there is strong evidence (JT-stat = 742, p-val = 3e-04) of a decreasing order in effect on median ΔMCC coinciding with an increase in the similarity threshold level. This evidence, overall, suggests that as the similarity threshold increases, the value of ΔMCC decreases. Such a result is indicative that the merging process reduces the maintenance effort as the similarity threshold increases. We note the result comparing the similarity threshold level of 1.0 compared to control will require further evaluation.

The results indicate that the merge process reduces the Halstead Effort and McCabe Cyclomatic Complexity of a combined grammar at each threshold below the control threshold, regardless of size. The results also indicate there is an increasing order to the amount of change in Halstead Effort and Cyclomatic Complexity caused by the algorithm as the similarity threshold decreases. These results imply that smaller values of the similarity threshold produce better results.

Since this was a randomized experiment, one may infer that the difference in similarity threshold caused the difference in Halstead Effort and McCabe Cyclomatic Complexity. Because the subjects were selected randomly from the population of Antlr grammars, we can extend this inference to that population. However, extending this inference to the population of grammars as a whole is speculative at best. This deficiency, however, is minor; the causal relationship is strong even though it applies only to Antlr grammars.

VI. Threats to Validity

In this work, we focused on threats to the conclusion, construct, internal, and external validity, as detailed by Wohlin et al. [22]. We have identified that there are no threats to the conclusion or construct validity. Nevertheless, there is a threat to the internal validity due to the under-representation of the domain, since we selected grammars only from Antlr. Additionally, there is a threat to external validity due to the restriction of representations to BNF and Antlr4 precludes the ability to evaluate TXL, SDF, or other grammar formats.

VII. Conclusions and Future Work

In this paper, we develop an algorithmic approach to merge programming language grammars. The goal is to facilitate the automatic creation of Island Grammars to aid in the development of multi-lingual software analysis tools. To evaluate this approach, we examined its effect on Halstead Effort and McCabe Cyclomatic Complexity. The experimental results showed that the merging approach reduces the merged grammar's Halstead Effort and Cyclomatic Complexity when controlling grammar size and similarity threshold. These results present a promising avenue toward automated Island Grammar generation and, therefore, multi-lingual analysis tools.

There are several avenues for future work. Initially, we intended to conduct further studies to improve the results herein by expanding the study to grammars selected from the GrammarZoo [23] collection. As part of this, we intend to extend the capabilities of this approach to incorporate tree-based grammars such as TXL and SDF, which will also reduce threats to validity. Additionally, we are currently integrating this approach into a more extensive process for the automated construction of Island Grammars for use in static analysis and quality measurement of software systems.

Acknowledgements

This research is supported by funding from the Ronald E. McNair Post Baccalaureate Achievement Program at Idaho State University, which is sponsored by the Department of Education (P217A170169).

References

[1] Z. Mushtaq, G. Rasool, and B. Shehzad, "Multilingual Source Code Analysis: A Systematic Literature Review," *IEEE Access*, vol. 5, pp. 11 307–11 336, 2017, bibtex: mushtaqMultilingualSourceCode2017.

[2] A. Janes, D. Piatov, A. Sillitti, and G. Succi, "How to Calculate Software Metrics for Multiple Languages Using Open Source Parsers," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 404, pp. 264–270. [Online]. Available: http://link.springer.com/10.1007/978-3-642-38928-3_20

[3] N. Synytskyy, J. R. Cordy, and T. R. Dean, "Robust multilingual parsing using island grammars," in *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2003, pp. 266–278.

[4] G. Caldiera, V. R. Basili, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[5] M. Haoxiang, *Languages and Machines: An Introduction to the Theory of Computer Science*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co. Inc., 1988.

[6] L. Moonen, "Generating robust parsers using island grammars," in *Proceedings Eighth Working Conference on Reverse Engineering*, Oct. 2001, pp. 13–22.

[7] A. V. Deursen and T. Kuipers, "Building documentation generators," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, Aug. 1999, pp. 40–49.

[8] L. Moonen, "Lightweight Impact Analysis using Island Grammars." in *IWPC*. Citeseer, 2002, pp. 219–228.

[9] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 308–318.

[10] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci, "Extracting structured data from natural language documents with island parsing," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*. IEEE, 2011, pp. 476–479.

[11] S. Klusener and R. Lammel, "Deriving tolerant grammars from a base-line grammar," in *International Conference on Software Maintenance*. IEEE, 2003, pp. 179–188.

[12] A. Goloveshkin and S. Mikhalkovich, "Tolerant parsing with a special kind of «Any» symbol: the algorithm and practical application," *Proceedings of the Institute for System Programming of the RAS*, vol. 30, no. 4, pp. 7–28, 2018. [Online]. Available: http://www.ispras.ru/en/proceedings/isp_30_2018_4/isp_30_2018_4_7/

[13] J. Kurš, M. Lungu, R. Iyadurai, and O. Nierstrasz, "Bounded seas," *Computer languages, systems & structures*, vol. 44, pp. 114–140, 2015.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge Massachusetts: The MIT Press, 2001.

[15] J. F. Power and B. A. Malloy, "A metrics suite for grammar-based software," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 16, no. 6, pp. 405–426, Nov. 2004. [Online]. Available: http://doi.wiley.com/10.1002/smr.293

[16] T. Parr, *The definitive ANTLR 4 reference*, ser. The pragmatic programmers. Dallas, Texas: The Pragmatic Bookshelf, 2012, oCLC: ocn802295434.

[17] M. Lanza and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Berlin; London: Springer, 2011, oCLC: 750954916.

[18] D. C. Montgomery, *Design and analysis of experiments*, eighth edition ed. Hoboken, NJ: John Wiley & Sons, Inc, 2013.

[19] J. J. Higgins, *An introduction to modern nonparametric statistics*. Pacific Grove, CA: Brooks/Cole, 2004.

[20] R. G. D. Steel, "A multiple comparison rank sum test: Treatments versus control," *Biometrics*, vol. 15, no. 4, p. 560, Dec. 1959. [Online]. Available: http://www.jstor.org/stable/2527654?origin=crossref

[21] A. R. Jonckheere, "A Distribution-Free k-Sample Test Against Ordered Alternatives," *Biometrika*, vol. 41, no. 1/2, p. 133, Jun. 1954. [Online]. Available: https://www.jstor.org/stable/2333011?origin=crossref

[22] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29044-2

[23] V. Zaytsev, "Grammar Zoo: A corpus of experimental grammarware," *Science of Computer Programming*, vol. 98, pp. 28–51, Feb. 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167642314003347