SIGMA: Systematic Island Grammar forMation Approach – Merging Grammars

Isaac 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've been successfully used for a variety of tasks. This includes impact analysis, multilingual parsing, source code identification, and other tasks. 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 fully automate the creation of island grammars. The end result of which is the reduction in initial effort and maintenance effort required for island grammar engineering.

Methods: We developed an automated approach to merge components of a grammar. To evaluate this approach we conducted three experiments, using a randomized complete block approach, across a selection of 30 grammars from 3 different size groups.

Results: What are the main findings? Practical implications?

Limitations: What are the weaknesses of this research?

Conclusions: What is the conclusion?

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

I. Introduction

Modern software development practice has led to the creation of software systems using multiple languages. As an example, the modern web application might use 5 or more languages (e.g. SQL, Java, TypeScript, HTML, CSS). Such multilingual codebases present a difficult challenge to the development and maintenance of code analysis tools [1]. To address this challenge current source code analysis tools utilize multiple parsers (one per supported language supported).

Island grammars have been shown to be a solution to the problem of developing multilingual parsers [2]. Though useful, the technique requires the manual combination of selected components from source grammars, a process which can be cumbersome when dealing with evolution of these grammars. To overcome this, we propose an automated method to reduce the initial time-consuming manual process and the further difficulty of maintaining the constructed island grammar in the face of source grammar evolution. In this paper we discuss a key to the component to the automated construction of island grammars. Specifically, the capability to correctly combine grammar productions together without reducing the grammar's ability to define key aspects of interest.

Towards this end, we believe that the automated merging of grammars is an important and necessary step in the evolution of island grammar research. The capability to automate grammar merging addresses the key issues found in the initial construction and further maintenance of island grammars. To evaluate this hypothesis, we used the Goal-Question-Metric (GQM) paradigm [3] to form the following research goal:

RG Develop an automated approach for merging components of a grammar to facilitate the construction of both the island and water components in an island grammar in order to reduce both the overall initial effort required in creating such grammars and the effort required to maintain such grammars.

The solution proposed will allow tool designers to develop source code analysis tools which support multiple programming languages in a less difficult and more maintainable way.

Code analysis tools have become an essential part of modern coding [X]. Using these tools, software engineers find bugs, identify security flaws, increase product quality, and comply with business rules and regulations. These tools have been integrated into software build processes and integrated into their pipelines for quality assurance and other services (e.g. SonarQube™[1]).

Organization

The remainder of this paper is organized as follows. Sec. II discusses the theoretical foundations of this work while also discussing other related studies. Sec. III details our approach to automate the merging of grammars which forms the foundation of an initial approach to automating the creation of multilingual parsers, which we call SIGMA. Sec. IV details the design of experiments which evaluate the proposed approach and its parameterization. Sec. V presents the results of the experiments. Sec. VI presents a discussion of the results and their interpretation in light of other studies. Sec. VII details the threats to the validity of this study. Finally, this paper is concluded in Sec. VIII.

II. Background and Related Work

A. Theoretical Foundations

A context free grammar can be described as $G = (V, \Sigma, P, S)$ [4]. Where, V is the set of non-terminal symbols,

[1] https://sonarqube.org

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. 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 $a \rightarrow \varepsilon$. $S \in V$ is the starting symbol. 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 [5]. $L(G)$ denotes the set of all valid sentences, or language, of grammar G.

Island Grammars are a specialized form of context free grammar which include the addition of a set of interests. These interests are used to focus the grammar on the set of language components most of interest to the grammar developers. An island grammar is formally defined as the following tuple $G' = (V, \Sigma, P, S, I)$ [5]. Where, I is the set of interests. These interests, as noted, define the components to which the grammar is focused, these are also known as islands. The remaining components of a language are reduced down into one or more catchall productions referred to as water [5]. This has 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: $L(G') \supset L(G)$. Island grammars offer several advantages over regular grammars for many applications including faster development time, lower complexity, and better error tolerance. Due to this island grammars have been used for many applications including documentation extraction and processing [6], impact analysis [7], and extracting code embedded in natural language documents [8], [9]. Of particular interest to our research is their use for creating multilingual parsers [2], which inspired this research, and research into the development of tolerant grammars [10], [11], [12].

B. Related Studies

Using an intermediate representation (IR), such as Java™ bytecode or Microsoft IL, is a common method for performing operations on multilingual code bases. This allows code analysis tools to be built around the language(s) defined by the IR rather than having to deal with each of the different languages individually. However, using an IR doesn't work when analyzing systems build from languages that don't have a common IR. Using an IR also doesn't work if performing manipulations of source code or if there isn't a clear correspondence between a source language and its IR. An example of this can be seen for PIT [13], a Java mutation testing framework. Even though their system is designed around transforming and manipulating java bytecode, the system struggles with handling bytecode generated by other JVM languages. However, they still have had limited success with other JVM languages.

Another approach that's been developed for handling multilingual systems is to use a separate parser for each language. Writing such a parser for each language is can be difficult, and thus it is better to use existing parsers [14]. Using existing parsers, one can modify the abstract syntax trees into a language independent form. [X] The problem with this approach is you have to use existing parsers, which might be written in a variety of languages.

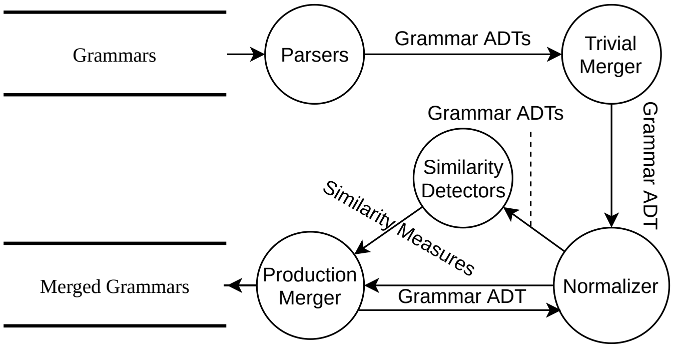

Fig. 1. Diagram of Approach

$\langle S \rangle$ ::= <A> | <B>
$\langle Y \rangle$ ::= <A> | 'y'
$\langle A \rangle$ ::= 'a' ε <B> <C>
$\langle C \rangle$ ::= 'c'
$\langle B \rangle$ ::= 'b' 'd'

(a) Grammar G_1.

$\langle S \rangle$ ::= <C> | <D>
$\langle Z \rangle$ ::= <S> | 'z'
$\langle C \rangle$ ::= 'c'
$\langle D \rangle$ ::= 'a' 'd' ('e' | 'c')
| (<C> | 'b')

(b) Grammar G_2.

Fig. 2. Example grammars.

Another approach for multilingual systems focuses on using island grammars. Using an Island Grammar approach, one can build a multilingual parser specially designed for code composed of interleaved languages [2]. This particular work inspired this research. However, island grammar's tend to be specific to their particular problem and their construction is currently a time-consuming manual process.

III. Approach

In order to generate an island grammar from one or more source grammars, there is a need to merge components of the source grammar(s) to reduce or deduplicate the final grammar. Specifically, this technique is to be utilized in combining and reducing the island components. Thus, the focus of this paper (as noted in Sec. I) is the development of a technique to merge grammars together. This technique is depicted in Fig. 1 and utilizes the following step:

1. Parse the input grammars.
2. Trivially merge the grammars into a single grammar.
3. Normalize the merged grammar.
4. Calculate the similarity between each pair of rules.
5. Merge the two most similar rules.
6. Repeat steps 3–5 until the two most similar rules are no longer similar.
7. Output the merged grammar.

The remainder of this section presents the details of each of these steps and utilizes a running example based on the the two grammars depicted in Fig. 2a and Fig. 2b.

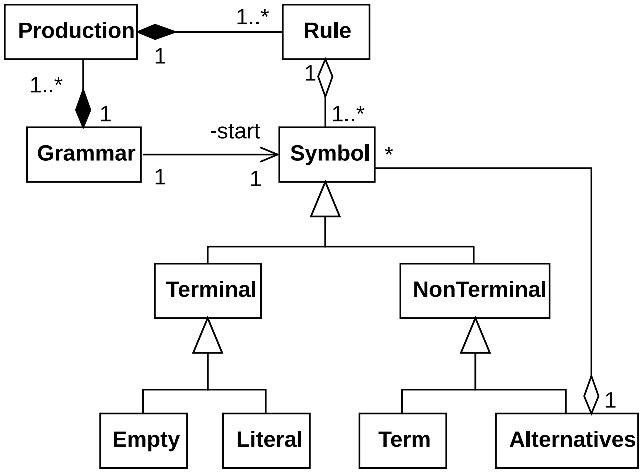

Fig. 3. Grammar domain model.

A. Parsing

The initial step of the process is the translation of the textual representations of the grammars into an in-memory representation useful for manipulation. There are numerous forms of grammar representation, but in this paper we focus on the Antlr4 textual representation and an in-memory domain model based on BNF. The latter model is depicted in Fig. 3.

To create instances of our model, we extracted information from the grammars using an ANTLR grammar parser generated by ANTLR. The parser creates an abstract syntax tree which is then converted into an instance of the domain model.

The ANTLR representation is not fully compatible with the domain model our approach depends upon. To resolve this, we removed or converted all non-bnf features except for dot, character range, character class, and not rules. The dot rule is a single character wildcard. Character ranges allow you to specify a range of characters. A character class only matches characters that fall into certain Unicode classes. The not rules match anything that isn't one of a set of items. For dot, character range, and character class rules, we converted them to special terminal symbols. We retained not rules, but made sure to extract them out as their own rules and their contents as well. For example

```
A: 'a' . ~('b' | 'c')
```

would be converted to
$\langle A\rangle$ ::= ‘a’ DOT $\langle$*Generated-1*$\rangle$
$\langle$*Generated-1*$\rangle$ ::= ~$\langle$*Generated-2*$\rangle$
$\langle$*Generated-2*$\rangle$ ::= ‘b’ | ‘c’
The not rules are ignored completely by our technique except for when detecting and merging duplicate rules.

B. Trivial Merge

A trivial merge between the start nodes of grammars is done rather easily. We begin by create a fresh start node in the new grammar. The production rule for this new start production is then an alternatives list containing each of the source grammars' start nodes. The trivial merging of grammars G_1 and G_2 yields the grammar G_3 depicted in Fig. 4a. Where the original source grammar start productions are renamed S1 and S2 representing the start productions for both G_1 and G_2, respectively.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle Y\rangle$ ::= <A> | ‘y’
$\langle A\rangle$ ::= ‘a’ ε <B> <C_1>
$\langle C_1\rangle$ ::= ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= <C_2> | <D>
$\langle Z\rangle$ ::= <S> | ‘z’
$\langle C_2\rangle$ ::= ‘c’
$\langle D\rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| (<C_2> | ‘b’)

(a) Grammar G_3.

Fig. 4. Grammar G_3.

C. Normalization

For normalization, we normalized to a unique normal form where every production is either of the following two forms:
$\langle A\rangle$ ::= $\langle B\rangle$ ‘a’ ...
$\langle B\rangle$ ::= $\langle A\rangle$ | ‘b’ | ... | ε
We chose these forms so that we would be able to easily compare similar productions. We refer to the two forms as form 1 and form 2 respectively.

In both forms, the rules on the right hand side must not expand to the same form as on the left hand side. We implemented this so that if productions are nested in different ways, they normalized to the same form. Here is an example: Given grammar G1
$\langle A\rangle$ ::= ‘a’ $\langle B\rangle$
$\langle B\rangle$ ::= ‘b’ ‘c’
and grammar G2
$\langle A\rangle$ ::= $\langle B\rangle$ ‘c’
$\langle B\rangle$ ::= ‘a’ ‘b’
they both normalize to the same grammar.
$\langle A\rangle$ ::= ‘a’ ‘b’ ‘c’

To normalize to the above form, we repeat six processes until the grammar stopped changing. These processes are to eliminate unused rules, simplify the productions, merge equivalent rules, eliminate unit rules, expand productions, and collapse compatible productions.

1) Eliminate Unused Rules: To eliminate unused rules, we removed all rules that cannot be eventually produced from the start rule. We did this by enumerating all used rules via a depth first search and then retaining only those rules. When applied to G_3 the grammar is transformed into grammar G_4 depicted in Fig. 5a.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle A\rangle$ ::= ‘a’ ε <B> <C_1>
$\langle C_1\rangle$ ::= ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= <C_2> | <D>
$\langle C_2\rangle$ ::= ‘c’
$\langle D\rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| (<C_2> | ‘b’)

(a) Grammar G_4.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle A\rangle$ ::= ‘a’ <B> <C_1>
$\langle C_1\rangle$ ::= ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= <C_2> | <D>
$\langle C_2\rangle$ ::= ‘c’
$\langle D\rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| (<C_2> | ‘b’)

(b) Grammar G_5.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle A\rangle$ ::= ‘a’ <B> <C>
$\langle C\rangle$ ::= ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= <C> | <D>
$\langle D\rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| (<C> | ‘b’)

(c) Grammar G_6.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle A\rangle$ ::= ‘a’ <B> ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= ‘c’ | <D>
$\langle D\rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| (‘c’ | ‘b’)

(d) Grammar G_7.

$\langle S\rangle$ ::= <S_1> | <S_2>
$\langle S_1\rangle$ ::= <A> | <B>
$\langle A\rangle$ ::= ‘a’ <B> ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle S_2\rangle$ ::= ‘c’ | <D>
$\langle D\rangle$ ::= <D_1> | <D_3>
$\langle D_1\rangle$ ::= ‘a’ ‘d’ <D_2>
$\langle D_2\rangle$ ::= ‘e’ | ‘c’
$\langle D_3\rangle$ ::= ‘c’ | ‘b’

(e) Grammar G_8.

$\langle S\rangle$ ::= <A> | <B> | ‘c’ |
<D>
$\langle A\rangle$ ::= ‘a’ ‘b’ ‘d’ ‘c’
$\langle B\rangle$ ::= ‘b’ ‘d’
$\langle D\rangle$ ::= <D_1> | ‘c’ | ‘b’
$\langle D_1\rangle$ ::= ‘a’ ‘d’ <D_2>
$\langle D_2\rangle$ ::= ‘e’ | ‘c’

(f) Grammar G_9.

Fig. 5. Transformed grammars produced during the normalization of grammar G_3.

2) Simplify: The simplify step was added to simplify our object representation. We removed ε items embedded inside terms and replaced objects containing only one object with the object inside them. When this step is applied to grammar G_4, it is transformed into grammar G_5, as depicted in Fig. 5b

3) Merge equivalent Rules: In this step, we replace rules that have identical productions with a single rule. The new rule is given a name derived from the rules that were merged to create it. In the following example of this step, rules a and b are merged into the rule a+b.

$\langle s\rangle$ ::= $\langle a\rangle$ | $\langle b\rangle$
$\langle a\rangle$ ::= ‘a’ ‘b’ $\langle a\rangle$
$\langle b\rangle$ ::= ‘a’ ‘b’ $\langle a\rangle$

Here it is after the rules are merged together.

$\langle s\rangle$ ::= $\langle a{+}b\rangle$
$\langle a{+}b\rangle$ ::= ‘a’ ‘b’ $\langle a{+}b\rangle$

When this step is applied to grammar G_5, it is transformed into grammar G_6, as depicted in Fig. 5c.

4) Eliminate Unit Rules: This part of the normalization process is taken from the Chomksy Normal Form. In this process, we eliminate all rules of one of the following two forms by replacing each usage of one of these rules with their production.

$\langle a\rangle$ ::= $\langle b\rangle$
$\langle a\rangle$ ::= ‘a’

The Chomsky Normal Form differs in its unit rule removal in that it does not eliminate rules of the second form. The reason we chose to eliminate the second form is so that the simplification process can simplify rules of the following form.

$\langle a\rangle$ ::= $\langle b\rangle$ ‘a’ ‘b’
$\langle b\rangle$ ::= ε

When this step is applied to grammar G_6, it is transformed into grammar G_7, as depicted in Fig. 5d.

5) Expand Productions: In this step, we expand out rules that have any nesting in their productions. We do this by replacing all nested content with a rule pointing to their content.

When this step is applied to grammar G_7, it is transformed into grammar G_8, as depicted in Fig. 5e.

6) Collapse Compatible Productions: In the final step of the normalization process, we collapse productions that are compatible with each other. This is to make sure that any non-terminal symbols referenced in a form does not produce a production of the same form. For example, the following productions

$\langle A\rangle$ ::= ‘a’ $\langle B\rangle$
$\langle B\rangle$ ::= ‘b’ ‘c’
$\langle C\rangle$ ::= ‘c’ | $\langle D\rangle$
$\langle D\rangle$ ::= ‘d’ | ‘e’

would collapse to

$\langle A\rangle$ ::= ‘a’ ‘b’ ‘c’
$\langle C\rangle$ ::= ‘c’ | ‘d’ | ‘e’

When this step is applied to grammar G_8, it is transformed into grammar G_9, as depicted in Fig. 5f.

D. Measuring Rule Similarity

The fourth step of SIGMA measures the similarity between every pair of productions. This measure is used by the fifth step to determine which two rules to merge. In this section, we lay out the two measures of similarity that we used for each of the two forms of rules that our normalization step produces. If two productions are different forms, we measure their similarity as 0.

1) Term Similarity: The first measure of similarity that we used is for measuring the similarity between two productions of form 1. For explaining our process, we use the following two productions.

$\langle P_a\rangle$ ::= ‘a’ $\langle A\rangle$ ‘b’ ‘c’
$\langle P_b\rangle$ ::= $\langle A\rangle$ $\langle B\rangle$ ‘b’ ‘a’

In the first step of measuring the similarity between two productions like this, SIGMA aligns the productions as best as possible. This can be performed using the Longest Common Subsequence (LCS) algorithm [15].

```
'a' <A>      'b'      'c'
```

```
<A> <B> 'b' 'a'
```

Once we have aligned the two productions as best as possible, we count the number of aligned terms and divide it by the total number of terms. The following formula also describes this:

$$S_1 = \frac{2|\text{LCS}(P_a, P_b)|}{|P_a| + |P_b|}$$

Applying the above formula to our example, `<A>` and `'b'` are each counted twice because they occur in both sequences of terms. The total number of terms across both productions is 8. The total similarity score is $S_1 = \frac{4}{8} = .5$.

2) Alternatives Similairty: The second measure of similarity that we use is for measuring the similarity between two productions of form 2. For explaining our process, we use the following two productions.

$\langle P_a \rangle$ ::= ‘a’ | $\langle A \rangle$ | ‘b’ | ‘c’
$\langle P_b \rangle$ ::= $\langle A \rangle$ | $\langle B \rangle$ | ‘b’ | ‘a’

Like the previous measure, we calculate the number of common alternatives before dividing by the total number of alternatives. However, since the order of the alternatives doesn’t matter, we use a different approach to measure the common alternatives. We simply count an alternative as common to both if it is both productions. The following formula describes how to calculate the similarity score using this method:

$$S_2 = \frac{2|P_a \cap P_b|}{|P_a| + |P_b|}$$

In our particular example, the common elements are `<A>`, `'a'`, and `'b'`. The total similarity is then calculated as $\frac{2*3}{8} = .75$.

E. Merging Similar Rules

Once we have detected the two most similar productions using the above method, we merge them together. Just like for measuring similarity, we have two different processes for merging similar productions. These measures also differ for the forms that we merge together.

In our process, we have to define a minimal similarity M_s score before we merge similar productions. We only merge productions if their similarity score is above or equal to this threshold.

1) Merging Similar Terms: To merge two productions of the first form, we use the LCS alignment produced while measuring the similarity between the terms. We identify each pair of subsequences that do not align. In the previous example, these pairs of subsequences are (`'a'`, ε), (ε, `<B>`), and (`'a'`, `'c'`). We then replace each subsequence with a term that produces to either subsequence. For example, the two sequences of terms mentioned earlier would merge to

$\langle P_{a+b} \rangle$ ::= (‘a’ | ε) $\langle A \rangle$ (ε | $\langle B \rangle$) ‘b’ (‘c’ | ‘a’)

2) Merging Similar Alternatives: Merging two productions of the second form is simpler than the first form. The merged production produced simply contains all alternatives that are in either constituent production. Our previous example would be merged to the following

$\langle P_{a+b} \rangle$ ::= ‘a’ | $\langle A \rangle$ | $\langle B \rangle$ | ‘b’ | ‘c’

F. Grammar Output

At the end of this process, we output the merged and transformed grammar. To do this, we simply visit every production in the grammar and print it out.

IV. Experimental Design

This section describes the overall experimental design used to evaluate the grammar merging approach presented within this paper. This section is further divided into subsections detailing the separate aspects of our experimental design.

A. Goals, Hypotheses, and Variables

This subsection describes the refinement of our initial research goal, defined in Sec. I, into a set of actionable research questions and metrics. Based on this set of research questions we also identified the variables used in statistical models driving our analysis procedures. We begin with the research questions and metrics.

We deconstructed RG, following the GQM paradigm, into the following set of research questions:

RQ1 What is the effect that this process has on the effort between the source grammars and the grammar produced by this approach?
Rationale: *It is expected that the merging of grammar components will reduce the maintenance effort required.*

RQ2 What is the effect that this process has on the complexity between the source grammars and the grammar produced by this approach?
Rationale: *It is expected that the merging and reduction of grammar components will reduce the complexity of the grammar, thus making the grammar easier to understand and read.*

In addition to these research questions we have selected the following metrics to assess the results of the approach used:

M1 Effort – To assess the effort required to maintain a grammar, we utilize the Halstead Effort measure for grammars as defined by Power and Malloy [16].

M2 Complexity – To assess the complexity of a grammar, we utilize McCabe’s Cyclomatic Complexity metric for grammars defined by Power and Malloy [16].

The dependent variables in the experiments, as indicated by the above hypotheses, are effort and complexity. Specifically, we are concerned with the change between the trivial merge state and the final grammar in terms of the effort and complexity of the grammar. Thus, the dependent variables of concern are $\Delta Effort$ and $\Delta Complexity$, representing the change in effort from trivial merge to final grammar and the change in complexity from trivial merge to final grammar, respectively. The independent variables we are concerned with are:

- Similarity Threshold – the parameter guiding the similarity measurements used in the merging process. The values used in the experiments are 0.001, 0.25, 0.5, 0.75, and 1.0.

- Size – the size of the grammar as defined by measuring its number of productions (PROD) [16], and thresholding this value into three distinct categories: Small, Medium, and Large.

B. Design

To evaluate the approach we elected to utilize two experiments. The first experiment evaluates the effect the approach has on the effort necessary to maintain a merged grammar as compared to its combined source grammars. The second experiment evaluates the effect the approach has on the complexity of the merged grammar as compared to its combined source grammars. These experiments utilize a Randomized Complete Block Design, with a single dependent variable (Effort, Complexity), a single treatment factor (Similarity Threshold) and the blocking variable size.

C. Experimental Units

This section describes the experimental units and the selection process used to select them. In these experiments, the experimental units are pairs of grammars selected from the Antlr4 [17] grammar repository[2]. At the time of this writing, the repository contained 198 individual grammars from a variety of general purpose and domain specific languages. The selection process used to select the grammar pairs for each experiment is depicted in Fig. 6 and works as follows. Initially, for each of the grammars in the repository we collected a combination of metadata and metric measurements. The metadata collected consisted of the language represented by the grammar and the version of that language (if applicable). We also measured the following metrics for each grammar based on the metrics suite by Power and Malloy [16]:

- TERM – the number of terminals.
- VAR – the number of defined non-terminals.
- PROD – the number of productions.
- MCC – McCabe's cyclomatic complexity.

Using the results of these calculations, we subdivided the dataset into three categories based on the logarithm of the number of productions (as a measure of size of the grammar). The logarithm was chosen because the distribution of the number of productions in the grammars is similar to a log-normal distribution. This subdivision was based on statistically construction thresholds [18]. This process utilizes the calculation of the mean of the logarithm of the size, $\mu = \text{MEAN}(\log(\text{size}))$, for the population of grammars and the standard deviation of the size, $\sigma = \text{STDDEV}(\log(\text{size}))$, for the population of grammars. Using these values we define the threshold values between the categories Small, Medium, and High as: Small-Medium: $e^{\mu-\sigma}$ and Medium-High: $e^{\mu+\sigma}$. Using these thresholds each grammar is then grouped into one of the three categories.

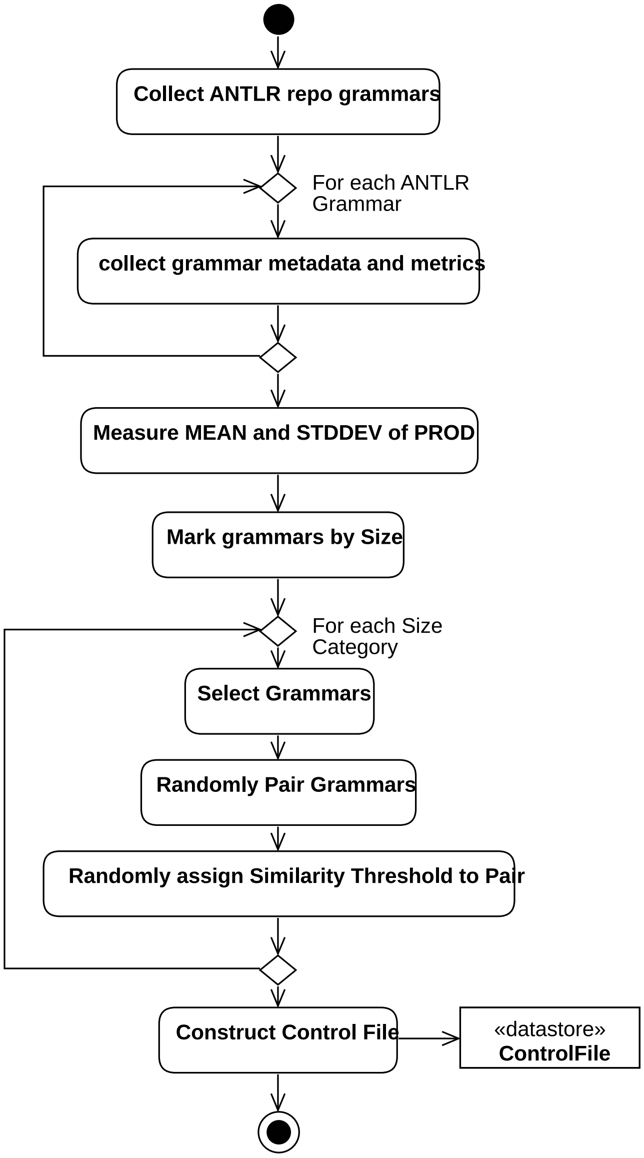

Fig. 6. Experimental unit selection process.

Using these categories as the blocking variable in the experiments, we can then begin the sampling process. For each size category we randomly select (without replication) 10 grammars. From these 10 grammars there are $\binom{10}{2} =$ 45 combinations of which we randomly select 5, for each experiment. In the case that replications are necessary, this process repeats for each replication or until the number of grammars are exhausted. In this case we have selected to complete two replications of each experiment, and the selected grammar pairs for both replications of both experiments are shown in Tab. I.

After the grammar pairs have been selected, they are assigned a treatment value for use during the experiment execution and data collection phase. For each size category in each replication of each experiment, the set of five grammar-

[2] https://github.com/antlr/grammars-v4. The sys-verilog grammar was excluded because of errors while parsing it.

TABLE I
EVALUATION GRAMMARS AND THEIR PROPERTIES.

		Replication 1		Replication 2	
Experiment	Category	Grammar-Pair	Similarity-Threshold	Grammar-Pair	Similarity-Threshold
	S	fen, cookie	1.0	url, tnt	.25
		propcalc, cmake	.75	fen, dice	.001
		metric, cmake	.5	dice, propcalc	.5
		telephone, metric	.25	telephone, cmake	1.0
		telephone, cookie	.001	telephone, dice	.75
	M	graphql, cool	.75	algol60, asm8080	.25
		javadoc, unicode-classify	.5	scss, golang	.001
		javadoc, golang	1.0	algol60, graphql	.5
		algol60, javadoc	.25	scss, graphql	.75
		javadoc, snobol	.001	xdr, golang	1.0
	L	cobol85, cql3	1.0	objc-one-step, tsql	.25
		objc-one-step, kotlin	.25	kotlin, csharp	.5
		asn_3gpp, cql3	.75	swift2, tsql	1.0
		asn_3goo, csharp	.001	objc-one-step, csharp	.001
		swift2, kotlin	.5	cobol85, kotlin-formal	.75

pairs are assigned, at random, a value for the similarity threshold treatment. The possible values that can be assigned are 0.001, 0.25, 0.5, 0.75, and 1.0. It should be noted that a value of 1.0 is the control value.

Finally, the data generated as part of the selection process is used to construct an experiment control file. This file is used to direct the experimental execution system and to ensure the validity of the process. Each control file is simply an ordered set of triples. Where, each triple consists of the following information:

- Grammar Pair - the pair of grammars to be merged together, as selected during the selection phase.
- Treatment - the similarity threshold value assigned to the grammar pair during the selection phase.
- Size - value of the size category, for use in constructing the data table.

Each replication of each experiment has a separate control file to control its execution. Thus, for each replication of each experiment, the control file triples are created and put into a list which is then randomized. This randomized list is then output to a file which is later read in by the experimental execution system.

D. Data Collection Procedures

Data collection is performed via the experiment execution system specifically developed for this study. This system is depicted in Fig. 7 and is controlled via the control files created during the experimental unit selection process.

The data collection process follows the activity diagram depicted in Fig. 7, as follows. First, the control file is read into the system and stored in an ordered list construct. Next, for each of the triples in the list, the following occurs: 1.) The selected similarity threshold value is applied. 2.) The grammars located, read in, and trivially merged together. 3.) The combined grammar's effort or complexity is then measured and recorded. 4.) The normalization process is executed. 5.) The grammar is then normalized using our technique. 6.) The final merging process is applied resulting in the final grammar. 7.) The resulting grammar's effort or complexity is then measured and recorded.

Once all experimental units have been processed the results are exported to a data table. The results from multiple replications of each experiment are then combined into a single data table. Finally, the data table is then used during the analysis phase.

E. Analysis Procedures

Both experiments utilize a Randomized Complete Block Design (RCBD) [19] with the size category of the grammar-pairs acting as the blocking variable. An assumption of the RCBD design is that there is no interaction between the blocking variable and the independent variable. We begin by initially validating this assumption via an interaction plot between size and similarity threshold. Upon confirmation of this assumption we select the appropriate analysis techniques. Prior to conducting the experiments, we conduct a sample size estimation to determine the number of repetitions necessary to achieve the power level of the tests we intend to use.

Typically, a single-factor RCBD experiment will utilize ANOVA to determine if there is a difference between the effects of the treatments. In this case, the statistical model used is as follows:

$$y_{ij} = \mu + \tau_i + \beta_j + \epsilon_{ij}$$

Where:

- y_{ij} is the value of the observation (either $\Delta Complexity$ or $\Delta Effort$) associated with the ith treatment level and jth block level.

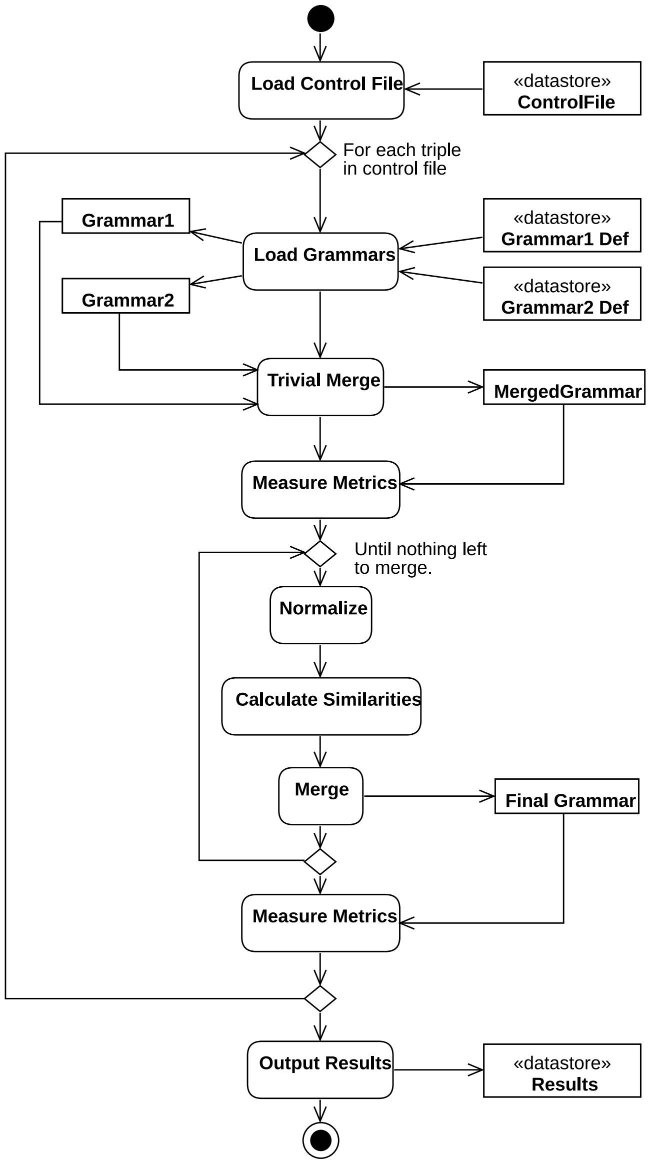

Fig. 7. Data collection procedure.

- μ is the baseline mean
- τ_i is the effect of the ith treatment level.
- β_j is the effect of the jth block level.

The hypotheses to be tested in this case are as follows:

- $H_{1,0}$: there is no difference between treatment effects.
- $H_{1,A}$: there is at least one difference between treatment effects.

ANOVA has several assumptions that must first be verified. First, is the assumption of homogeneity of variance. To evaluate this assumption, we will use Levene's Test [20]. The next assumption is that both the treatments and errors are normally distributed. To verify this assumption, we will use the Anderson-Darling Test for Goodness of Fit [21] to the normal distribution for both the errors and treatment effects. Finally, there is the assumption of independence of the observations, which is valid due to the nature of the process.

In the case of any violations of these assumptions we will attempt to correct the violation. If the homogeneity of variance assumption has been violated, we will attempt to correct this via either by using a Box-Cox [22] transformations or via the application of a weighted least squares. Similarly, to correct issues with the normality of the errors we will also attempt a transformation of the data. Finally, in the case that the assumptions are violated beyond the capability to correct, we will be forced to utilize a non-parametric approach.

In the case that the ANOVA assumptions have been seriously violated and cannot be reasonably corrected, we will utilize the Friedman Test [23] instead. The Friedman Test also has a set of assumptions that must be met prior to use. First, the blocks must be mutually independent of one another and must contain the same number of experimental units as there are treatments. Second, the observations both within and among blocks must be independent. Third, the variable of interest must be continuous. Finally, the measurement scale of the block variable must be at least ordinal. In the case of both experiments each of these assumptions are met.

The model for the Friedman Test is similar to that of ANOVA, but focuses on the median rather than the mean. Thus, the hypotheses to be tested in this case are as follows:

- $H_{1,0}$: there is no difference between median values
- $H_{1,A}$: there is at least one difference between median values

Regardless of whether ANOVA or the Friedman test is used, the next step is to conduct the hypothesis test and make a decision. In this case we have selected an α threshold of 0.95. In the case that we reject $H_{1,0}$ we will then conduct a multiple-comparison procedure to compare the individual effects of each level of the similarity threshold treatment. Specific to these experiments, we consider the similarity threshold level of 1.0 to be the control (as noted in Sec. IV-B). We have selected to pair Dunnett's [24] multiple comparison procedure if ANOVA is used, and Steel's [25] multiple comparison procedure (a non-parametric technique analogous to Dunnett's) if using the Friedman test. Both of these comparison procedures control the error for multiple comparisons and allow the ability to compare against a control. In the case of Dunnett's Test we will be evaluating the following hypotheses:

- $H_{1,0}$: $\mu_i = \mu_{control}$ for some treatment i.
- $H_{1,A}$: $\mu_i < \mu_{control}$ for some treatment i.
- Where, μ is the mean value of the effect.

In the case of Steel's Test we will be evaluating the following hypotheses:

- $H_{1,0}$: $M_i = M_{control}$ for some treatment i.
- $H_{1,A}$: $M_i = M_{control}$ for some treatment i.
- Where, M is the median value of the effect.

For either of these tests we will use an α threshold of 0.95.

Finally, for either approach we will conduct a power analysis to ensure the Type-II error rate is within tolerance. This final analysis also ensures that the original sample size and repetition analysis was correct.

F. Evaluation of Validity

This subsection details the procedures put in place as part of the experimental design to ensure the validity of the results. As per Wohlin et al. [26] we are concerned with conclusion, internal, construct, and external validity. The following sub-subsections detail the procedures/processes used to ensure the validity of our experiments.

Conclusion Validity: The following characteristics of our experiments and data help to ensure the conclusion validity of this study. First due to the nature of the data we have put into place techniques which validate the assumptions of our statistical tests, and in the case of that we are unable to correct violations of these assumptions we have selected alternative analysis procedures. As part of this study several metrics are used to evaluate the selected grammars and those generated by our proposed technique. To ensure that the accuracy of the implementation of these metrics, they have been thoroughly tested prior to use. To ensure the accurate operation of the experiments we have implemented and tested an experimental execution system. For each experiment we are utilizing multiple comparison procedures, both of which have been selected such that they utilize proper error correcting procedures. Finally, due to the nature of the experiments there are no issues in the experimental setting that will cause random irrelevancies.

Internal Validity: The following characteristics of our experiments and data help to ensure the internal validity of this study. First, due to the nature of the experimental units and experimental system the timing of experiment execution has no effect on the outcome. Second, due to the nature of the experimental unites there are no social concerns affecting the internal validity. Third, the nature of the experiment which utilizes grammars generated by the combination of existing grammars and which does not affect the source grammars there is no effect on the outcome when repeating the same experiment. Fourth, We have selected grammars that are representative for population of ANTLR grammars and for different sizes, but which may be under-representative of grammars in general. Fifth, the selection, assignment, and analysis procedures used in these experiments have been selected to ensure that there is no ambiguity regarding the direction of causal influence.

Construct Validity: The following characteristics of our experiments and data help to ensure the construct validity of this study. First, the nature of the experimental units social issues affecting construct validity. Second, the design of the experiments and the technique being evaluated prevents both mono-operation and mono-method biases. Third, the implementation of the constructs underlying the technique being evaluated, there are no issues due to inadequate preoperational explication or confounding constructs. Fourth, the use of size categories for source grammars is designed to evaluate the spectrum of grammars rather than simply testing whether the technique works on grammars. Fifth, the design of the experiments and the experiment execution system prevents interaction of treatments. Sixth, due to the nature of the technique and the experimental process the source grammars are left unaffected and thus cannot unintentionally affect due to the application of the treatment.

TABLE II
DESCRIPTIVE STATISTICS.

Size Category	**Mean Delta HAL**	**Mean Delta MCC**
Small	1290.2	32.32
Medium	18573	112.7
Large	149854	842.4

External Validity: The following characteristics of our experiments and data help to ensure the internal validity of this study. The selection of simple languages (such as the group of small languages) not currently in use in industry would pose a threat to external validity, but this is offset by the inclusion of the other languages from both the medium and large groups. Second, we restrict our internal representations to BNF, but we do include the ability to utilize ANTLR grammars which are currently used in practice. Yet, because we do not include grammars in other formalisms (i.e., SDF [27] and TXL [28]) used in practice a threat to external validity still remains. Finally, the nature of the experiment precludes an effect due to the data or time of the application of the treatment.

V. RESULTS

A. Descriptive Statistics

This subsection describes the data collected via a series of descriptive statistics. Tab. II shows the mean values of Delta HAL and Delta MCC for each value of the size category in the respective experiments. Along with this table the dispersion of the both DeltaHAL and DeltaMCC across size and similarity threshold values is displayed in Fig. 8 and Fig. 9, respectively. We can also see the distribution and correlations between these for both the HAL and MCC experiments in Fig. 10 and Fig. 11, respectively.

As we can see from the Residuals vs. Fitted plot, depicted in Fig. 12, the homogeneity of variance assumption is not met. To validate this observation we used Levene's Test. In this test, the null-hypotheses (H_0) asserts that the variance is equally distributed. As expected, from our observations of the Residuals vs Fitted Values plot, the test produced an F-value of 19.962 and a p-value of 1.267e-07 indicating that we should reject the null hypothesis that there is a homogeneity of variance.

The Normal Q-Q plot, depicted in Fig. 12, indicates that the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test and a Shapiro-Wilks GOF test against the Normal distribution. In both cases the null-hypothesis (H_0) is that the empirical distribution is Normal. In the case of the Anderson-Darling test the A statistic has a value of 6.8257 and an associated p-value of < 2.2e-16 while the Shapiro-Wilk

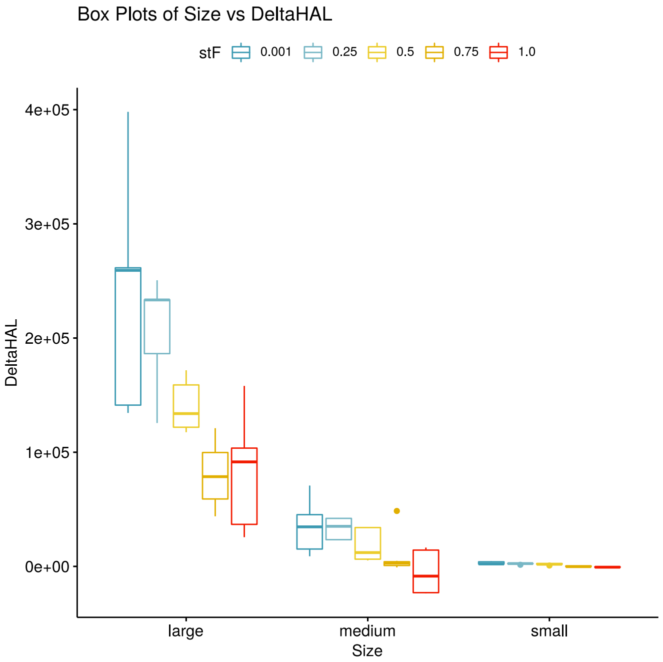

Fig. 8. HAL Boxplot.

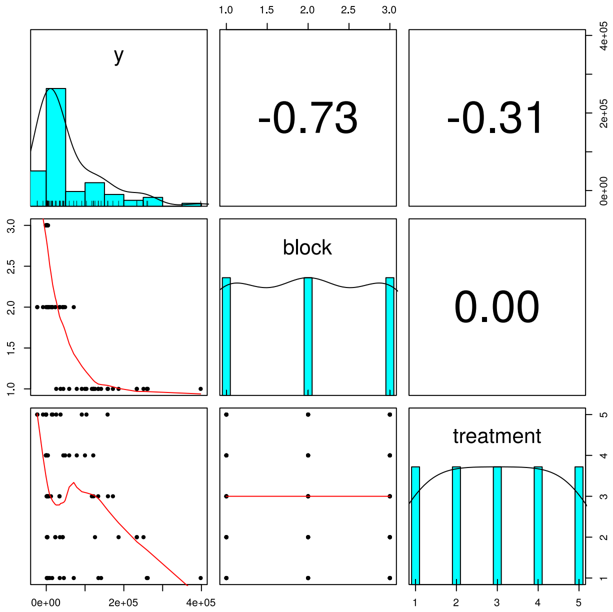

Fig. 10. HAL Scatterplots.

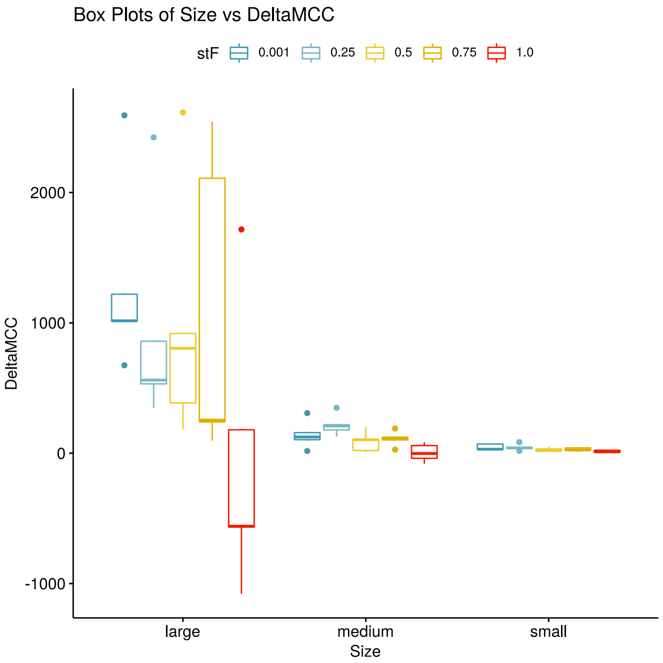

Fig. 9. MCC Boxplot.

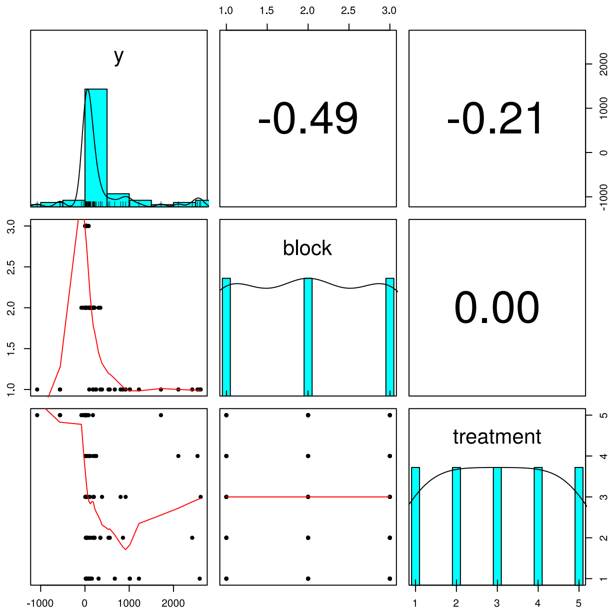

Fig. 11. MCC Scatterplots.

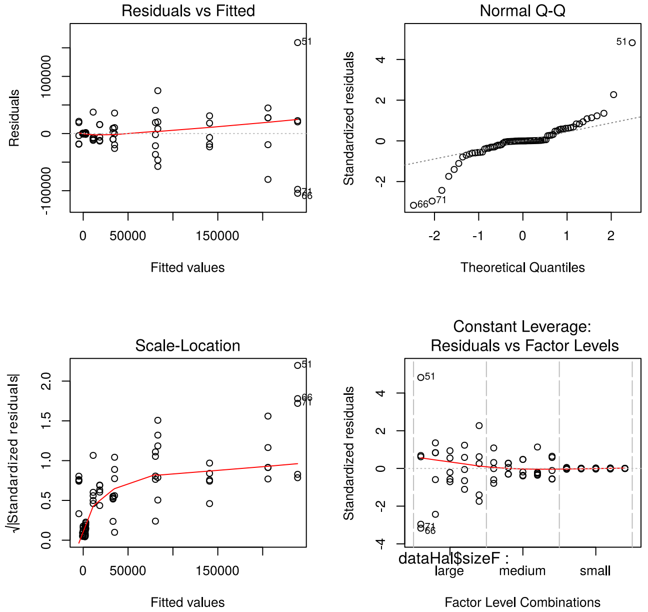

Fig. 12. Experiment 1 Q-Q plots.

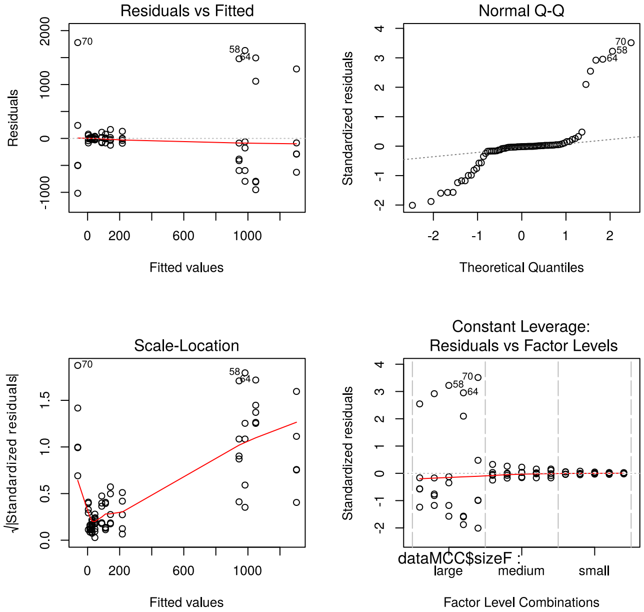

Fig. 14. Experiment 2 Q-Q plots.

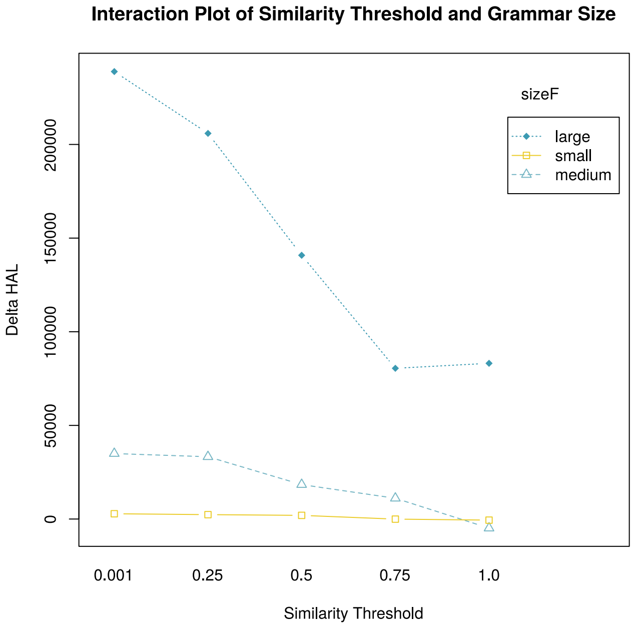

Fig. 13. Experiment 1 interaction plot.

normality test has a W statistic of 0.75365 with an associated 6.876e-10 both of which indicate that we are to reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Fig. 13, indicates that their is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0. This indicates that the assumption that the block and treatment do not interact and that an RCBD design is inappropriate and rather a factorial design would be correct.

As we can see from the Residuals vs. Fitted plot, depicted in Fig. 14, the homogeneity of variance assumption is not met. To validate this observation we used Levene's Test. In this test, the null-hypotheses (H_0) asserts that the variance is equally distributed. As expected, from our observations of the Residuals vs Fitted Values plot, the test produced an F-value of 29.856 and a p-value of 3.61e-10 indicating that we should reject the null hypothesis that there is a homogeneity of variance.

The Normal Q-Q plot, depicted in Fig. 14, indicates that the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test and a Shapiro-Wilks GOF test against the Normal distribution. In both cases the null-hypothesis (H_0) is that the empirical distribution is Normal. In the case of the Anderson-Darling test the A statistic has a value of 9.9558 and an associated p-value of < 2.2e-16 while the Shapiro-Wilk normality test has a W statistic of 0.68204 with an associated 1.852e-11 both of which indicate that we are to reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Fig. 15, indicates that their is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0. This indicates that the assumption that the block and treatment do not interact and that an RCBD design is inappropriate and rather a factorial design would be correct.

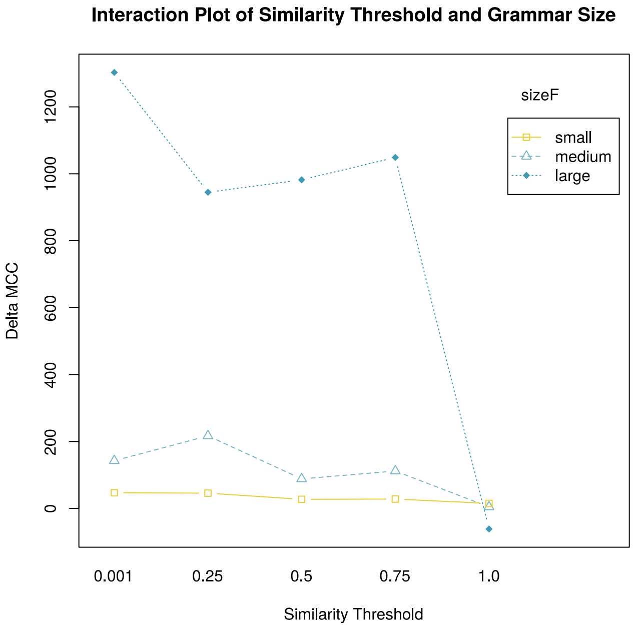

Fig. 15. Experiment 2 interaction plot.

B. *Hypothesis Testing*

This subsection details the results of the HAL experiment and MCC experiment. In both cases we initially executed a sample size analysis to determine the number of replications to execute to achieve a power level of .95. The results of this analysis indicate the need to execute a total of 5 replications of each example. To achieve this we would need a total of 12 grammars randomly selected from each size category to form the necessary pairs of grammars used in the experiments.

1) Experiment 1: We conducted an experiment using a factorial design based on the levels of the factors size and similarity threshold. Our goal was to determine if the algorithm when applied had an effect on the change in Halstead Effort for the combined grammars, as measured by DeltaHAL. Because of the violations to the assumptions of ANOVA, we were opted to utilize non-parametric methods in our analysis. The initial analysis utilize a permutation F-test approach to evaluate if there is any difference among treatment levels. The results was an F-Value of 9.569 and a p-value of 4.73e-06 indicating that we could reject H_0. We also note that the interaction indicated in Fig. 13 was shown to be significant with an F-Value of 5.098 and a p-value of 7.31e-05.

We were interested in determining if there was any difference between the control level (1.0) and other levels of the similarity threshold. To evaluate this we used Steel's multiple comparison against a control. The results of this test indicated for each level that there was a significant difference and that each level the mean DeltaHAL was lower than that of the control value. Finally, we were interested in evaluating the selected levels of the similarity threshold (excluding control) and if there was an implied order (that is there is a decreasing order to the change in DeltaHAL) as the level value is lowered in the similarity threshold. To evaluate this we used Jonchkheere-Terpstra test for ordered differences among classes, which yields a JT statistic value of 767 and a p-value of 6e-04, when executed using 10000 permutations.

2) Experiment 2: We conducted an experiment using a factorial design based on the levels of the factors size and similarity threshold. Our goal was to determine if the algorithm when applied had an effect on the change in Halstead Effort for the combined grammars, as measured by DeltaHAL. Because of the violations to the assumptions of ANOVA, we were opted to utilize non-parametric methods in our analysis. The initial analysis utilize a permutation F-test approach to evaluate if there is any difference among treatment levels. The results was an F-Value of 9.569 and a p-va.ue of 4.73e-06 indicating that we could reject H_0. We also note that the interaction indicated in Fig. 13 was shown to be significant with an F-Value of 5.098 and a p-value of 7.31e-05.

We were interested in determining if there was any difference between the control level (1.0) and other levels of the similarity threshold. To evaluate this we used Steel's multiple comparison against a control. The results of this test indicated for each level that there was a significant difference and that each level the mean DeltaHAL was lower than that of the control value. Finally, we were interested in evaluating the selected levels of the similarity threshold (excluding control) and if there was an implied order (that is there is a decreasing order to the change in DeltaHAL) as the level value is lowered in the similarity threshold. To evaluate this we used Jonchkheere-Terpstra test for ordered differences among classes, which yields a JT statistic value of 767 and a p-value of 6e-04, when executed using 10000 permutations.

VI. Interpretation

The results indicate that proposed merge process embodied in our algorithm 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 that in the case of Halstead Effort, there is a increasing order to the amount of change in halstead effort and cyclomatic complexity caused by the algorithm as the similarity threshold decreases. This implies that smaller values of the similarity threshold produce better results.

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

VII. Threats to Validity

This section details the known threats to the validity of this study. Specifically, we focused on threats to the conclusion, construct, internal, and external validity as detailed by

Wohlin et al. [26]. Given the evaluation of validity detailed in Sec. IV-F we have identified no threats to the conclusion and construct validity of this study. But, as noted in Sec. IV-F there is a threat to the internal validity of this study. Specifically, we selected grammars from the ANTLR repository which limits the grammars available as well as the formalism in which they are implemented. This is a threat due to the under-representation of the domain. Additionally, since we restrict our internal representations to BNF, but we do not include the ability to utilize TXL or SDF grammars there is a threat to external validity, as well.

VIII. Conclusions and Future Work

In this paper we have developed an algorithmic approach to merging programming language grammars together. The goal of this approach is to facilitate the automatic creation of Island Grammars for use in the development of multi-lingual software analysis tools. We evaluate this approach through two experiments on combined grammars selected from the Antlr grammar repository. These experiments evaluated the effects of this approach on changes to Halstead Effort and McCabe Cyclomatic Complexity of the grammar pairs as they are merged. The results show that the merging approach reduces the halstead effort and cyclomatic complexity when taking both grammar size and similarity threshold into consideration.

The results of this study are promising and present a potentially fruitful avenue towards the development of an automtated approach to Island Grammar generation. Island Grammars have been shown to be an extremely useful approach for the development of multi-lingual analysis tools, but can be time-consuming to develop.

A. Impact

Immediate avenues for future work are as follows. We intended to conduct further studies to improve the results herein by expanding the study to grammars selected from the GrammarZoo [29] collection. As part of this we intend to extend the capabilities of this approach to incorporate TXL and SDF grammars which will also reduce threats to validity identified in Sec. VII. The results herein indicate the promise of this approach and we intend to integrate it with ongoing work to develop tools which will support 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] 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.

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

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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/

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

[13] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "PIT: a practical mutation testing tool for Java (demo)," in *Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016*. Saarbr&#252;cken, Germany: ACM Press, 2016, pp. 449–452. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2931037.2948707

[14] 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

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

[16] 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

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

[18] 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.

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

[20] H. Levene, "Robust tests for equality of variances1," *Contributions to probability and statistics: Essays in honor of Harold Hotelling*, vol. 2, pp. 278–292, 1960.

[21] T. W. Anderson and D. A. Darling, "A Test of Goodness of Fit," *Journal of the American Statistical Association*, vol. 49, no. 268, p. 765, Dec. 1954. [Online]. Available: http://www.jstor.org/stable/2281537?origin=crossref

[22] G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*,

vol. 26, no. 2, pp. pp. 211–252, 1964. [Online]. Available: http://www.jstor.org/stable/2984418

[23] M. Friedman, "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, Dec. 1937. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522

[24] C. W. Dunnett, "A Multiple Comparison Procedure for Comparing Several Treatments with a Control," *Journal of the American Statistical Association*, vol. 50, no. 272, pp. 1096–1121, Dec. 1955. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1955.10501294

[25] 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

[26] 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

[27] J. Heering, P. R. H. Hendriks, P. Klint, and J. Rekers, "The syntax definition formalism SDF—reference manual—," *ACM SIGPLAN Notices*, vol. 24, no. 11, pp. 43–75, Nov. 1989. [Online]. Available: http://portal.acm.org/citation.cfm?doid=71605.71607

[28] J. R. Cordy, "TXL - A Language for Programming Language Tools and Applications," *Electronic Notes in Theoretical Computer Science*, vol. 110, pp. 3–31, Dec. 2004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S157106610405217X

[29] 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