

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, robepea2@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 across a selection of 9 grammars from 3 different size groups. The first experiment evaluate the effects of different similarity thresholds when merging a grammar with itself. The second experiment compared the effects of merging pairs of grammars when considering the size of the grammar and the change in similarity threshold. Finally, the third experiment evaluates the effect of a change in similarity threshold on the approach when considering grammars that are very similar using a grammar mutation approach.

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:

RG1 Develop and 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™ [X]). Improvements in the underlying technologies that these tools use will allow them to rise to the challenges we are facing today.

Organization

The remainder of this paper is organized as follows. Sec. II describes the theoretical foundations of this work and relates this work to prior studies. Sec. III explains how SIGMA, our proposed tool for merging grammars, functions and the design decisions behind it. Sec. ?? explains how we evaluated SIGMA on a small selection of grammars. Sec. V shows the results of our analysis.

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, 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 empty, 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 repeatedly applying productions of that grammar [5]. $L(G)$ denotes the set of all valid sentences, or language, of grammar G .

Island grammars are specialized grammars defined as $G = (V, \Sigma, P, S, I)$ [5]. It is a context free grammar with the addition of a set of interests, I . This set of interests is utilized to construct the grammar such that it matches only those productions defined in I , which are called the islands, while reducing all other aspects of the language to a common components called water. They are also designed to blindly match surrounding content that is not of interest as water [5]. This has the effect of reducing the complexity of the grammar. Furthermore an island grammar, G' , derived from a grammar, G , has a language $L(G')$ which satisfies the following property: $L(G') \supset L(G)$. They offer several advantages over regular grammars for many applications including faster development time, lower complexity, and better error tolerance. 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 the development of tolerant grammars [10], [11], [12].

B. Related Studies

Using an intermediate representation (IR) is a common method for performing operations on multilingual code bases. This allows code analysis tools to be built around the language 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 difficult, especially for languages like C++, so it is better to use existing parsers

[14]. Using existing parsers, one can modify the abstract syntax trees into a language independent form. The problem with this approach is you have to use existing parsers, which might be written in a variety of languages.

Another approach for multilingual systems focuses on using island grammars. Island grammars are a kind of grammar specially designed for extracting constructs of interest while ignoring everything else [5]. They're known for their robustness. Using an Island Grammar approach, one can build a multilingual parser especially designed for code composed of interleaved languages [2]. This particular work inspired this research. However, currently this relies on building the island grammar by hand. move to intro

C. Contributions

In this paper, we contribute an automatic method for creating multilingual grammars. Using these grammars, one can create multilingual parsers and source code analysis tools. Current multilingual tools mostly either use an intermediate representation such as Java™ bytecode which doesn't easily allow manipulations of the original source, rely on using multiple existing parsers which are difficult to hook up together, or by handwriting grammars for each language combination the tool supports. Our method still allows manipulation of the original source code whilst providing an automatic way of building automatic multilingual grammars and parsers. Applications that we expect to be able to use our tool include quality analysis, mutation testing, static code analysis, graph based testing, and refactoring.

III. APPROACH

In this section, we go in depth into each step of our approach. Our approach is presented in Fig. 1. Sec. III-A explains how SIGMA parses input grammars. Sec. III-B explains how SIGMA trivially merges the input grammars. Sec. III-C goes into depth on how SIGMA normalizes the trivially merged grammar. Sec. III-D explains how SIGMA calculates the similarity between two rules. Sec. III-E explains how SIGMA merges similar rules. Our approach for merging grammars uses the following 7 steps:

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.

Throughout this section we describe each of these steps while presenting a running example based on the the following two grammars. The first grammar, G_1 , is depicted in Fig. 2a. The second grammar, G_2 , is depicted in Fig. 2b. The following subsections describe the steps of this approach in detail.

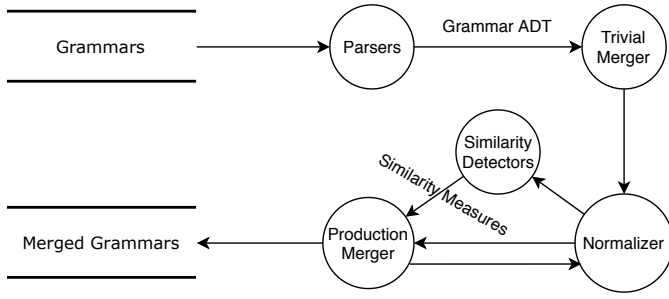


Fig. 1. Diagram of Approach

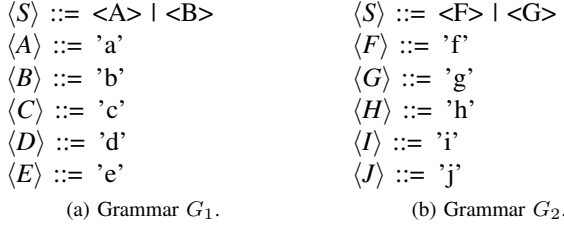


Fig. 2. Example grammars.

A. Parsing

The parsing process transforms the textual form of each grammar into an **object oriented** representation easier for our code to process. A **class diagram** of this representation is in Fig. 3. To do this transformation, **SIGMA uses parsers generated by ANTLR to extract an abstract syntax tree for each grammar. Each abstract syntax tree is then converted into our representation. Finally, rules with the same name are linked together within each grammar.**

Their ANTLR representation wasn't fully compatible with the BNF format 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

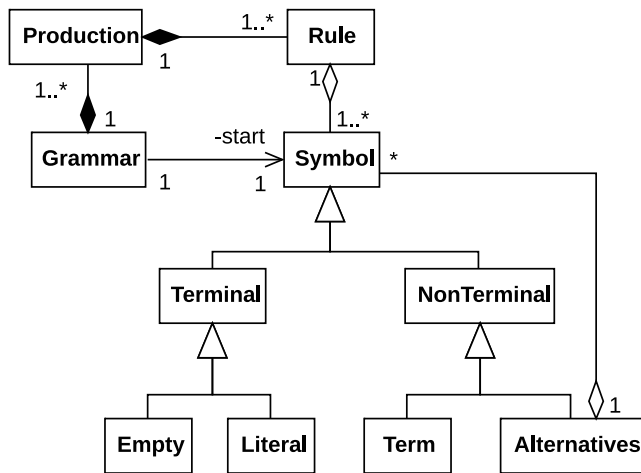


Fig. 3. Class Diagram of Grammar

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= 'a'$
 $\langle B \rangle ::= 'b'$
 $\langle C \rangle ::= 'c'$
 $\langle D \rangle ::= 'd'$
 $\langle E \rangle ::= 'e'$
 $\langle F \rangle ::= 'f'$
 $\langle G \rangle ::= 'g'$
 $\langle H \rangle ::= 'h'$
 $\langle I \rangle ::= 'i'$
 $\langle J \rangle ::= 'j'$

Fig. 4. Grammar G_3 .

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 kept 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

$\langle A \rangle ::= 'a' \text{ DOT } \langle \text{Generated-1} \rangle$
 $\langle \text{Generated-1} \rangle ::= \sim \langle \text{Generated-2} \rangle$
 $\langle \text{Generated-2} \rangle ::= 'b' \mid 'c'$

The not rules are ignored completely by our algorithm 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. 4. Where the original source grammar start productions are renamed S1 and S2 representing the start productions for both G_1 and G_2 , respectively.

C. Normalization

For normalization, we normalized to a unique normal form where every production is either of the following two forms:

$\langle A \rangle ::= \langle B \rangle 'a' \dots$
 $\langle B \rangle ::= \langle A \rangle \mid 'b' \mid \dots \mid \epsilon$

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 G_1

$\langle A \rangle ::= \text{'a'} \langle B \rangle$
 $\langle B \rangle ::= \text{'b'} \text{'c'}$

and grammar G_2

$\langle A \rangle ::= \langle B \rangle \text{'c'}$
 $\langle B \rangle ::= \text{'a'} \text{'b'}$

they both normalize to the same grammar.

$\langle A \rangle ::= \text{'a'} \text{'b'} \text{'c'}$

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.

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.

$\langle s \rangle ::= \langle a \rangle \mid \langle b \rangle$
 $\langle a \rangle ::= \text{'a'} \text{'b'} \langle a \rangle$
 $\langle b \rangle ::= \text{'a'} \text{'b'} \langle a \rangle$

Here it is after the rules are merged together.

$\langle s \rangle ::= \langle a+b \rangle$
 $\langle a+b \rangle ::= \text{'a'} \text{'b'} \langle a+b \rangle$

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 Chomsky 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.

$\langle a \rangle ::= \langle b \rangle$
 $\langle a \rangle ::= \text{'a'}$

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.

$\langle a \rangle ::= \langle b \rangle \text{'a'} \text{'b'}$
 $\langle b \rangle ::= \varepsilon$

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

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(a) Grammar G_4 .

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(b) Grammar G_5 .

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(c) Grammar G_6 .

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(d) Grammar G_7 .

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(e) Grammar G_8 .

$\langle S \rangle ::= \langle S1 \rangle \mid \langle S2 \rangle$
 $\langle S1 \rangle ::= \langle A \rangle \mid \langle B \rangle$
 $\langle S2 \rangle ::= \langle F \rangle \mid \langle G \rangle$
 $\langle A \rangle ::= \text{'a'}$
 $\langle B \rangle ::= \text{'b'}$
 $\langle C \rangle ::= \text{'c'}$
 $\langle D \rangle ::= \text{'d'}$
 $\langle E \rangle ::= \text{'e'}$
 $\langle F \rangle ::= \text{'f'}$
 $\langle G \rangle ::= \text{'g'}$
 $\langle H \rangle ::= \text{'h'}$
 $\langle I \rangle ::= \text{'i'}$
 $\langle J \rangle ::= \text{'j'}$

(f) Grammar G_9 .

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

5) **Expand Productions**: 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**: 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.

$$\begin{aligned}\langle P_a \rangle &::= \text{'a'} \langle A \rangle \text{'b'} \text{'c'} \\ \langle P_b \rangle &::= \langle A \rangle \langle B \rangle \text{'b'} \text{'a'}\end{aligned}$$

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].

$$\begin{array}{ccccc}\text{'a'} & \langle A \rangle & & \text{'b'} & & \text{'c'} \\ & \langle A \rangle & \langle B \rangle & \text{'b'} & \text{'a'} & \end{array}$$

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, $\langle A \rangle$ 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 Similarity*: 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.

$$\begin{aligned}\langle P_a \rangle &::= \text{'a'} \mid \langle A \rangle \mid \text{'b'} \mid \text{'c'} \\ \langle P_b \rangle &::= \langle A \rangle \mid \langle B \rangle \mid \text{'b'} \mid \text{'a'}\end{aligned}$$

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 $\langle A \rangle$, 'a' , and 'b' . The total similarity is then calculated as $\frac{2 \times 3}{8} = .75$.

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 $(\text{'a'}, \epsilon)$, $(\epsilon, \langle B \rangle)$, and $(\text{'a'}, \text{'c'})$. 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 $\langle P_{a+b} \rangle ::= (\text{'a'} \mid \epsilon) \langle A \rangle (\epsilon \mid \langle B \rangle) \text{'b'} (\text{'c'} \mid \text{'a'})$

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

$$\langle P_{a+b} \rangle ::= \text{'a'} \mid \langle A \rangle \mid \langle B \rangle \mid \text{'b'} \mid \text{'c'}$$

F. Grammar Output

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 further refines our initial research goal, defined in Sec. I, into a set of actionable research questions and metrics. Using this set of research questions, we also develop an initial set of statistical hypotheses and describe the variables used in statistical models driving our analysis procedures. We begin with the research questions and metrics.

We deconstructed RG1, 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:

RQ2 What is the effect that this process has on the complexity between the source grammars and the grammar produced by this approach?

Rationale:

RQ3 What is the effect that this process has on the size between the source grammars and the grammar produced by this approach?

Rationale:

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].

M3 Size – To assess the size of a grammar, we utilize the PROD metric defined by Power and Malloy [16].

These questions and metrics then lead us to the following set of initial hypotheses:

$H_{1,0}$ The approach produces no effect on mean change in effort.

$H_{2,0}$ The approach produces no effect on mean change in complexity.

The dependent variables in the experiments, as indicated by the above hypotheses, are effort and complexity. 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

This section details the experimental designs used to evaluate the approach described in Sec. III. 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 an *island* grammar as compared to its source grammars. The second experiment evaluates the effect the approach has on the complexity of the generated *island* grammar as compared to its source grammars. In order to fairly compare different grammars, we used size as a blocking variable. Thus both experiments utilize a Randomized Complete Block Design, with a single dependent variable (Effort, Complexity), a single Treatment (Similarity Threshold) and the blocking variable Size.

C. Subjects

We utilized the Antlr4 [17] grammar repository¹ as our population of grammars. At the time of this writing, the repository contained 198 individual grammars from a variety of general purpose and domain specific languages. For each grammar we collected a combination of metadata and metrics. The metadata collected consisted of the language represented by the grammar, 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.

¹<https://github.com/antlr/grammars-v4>

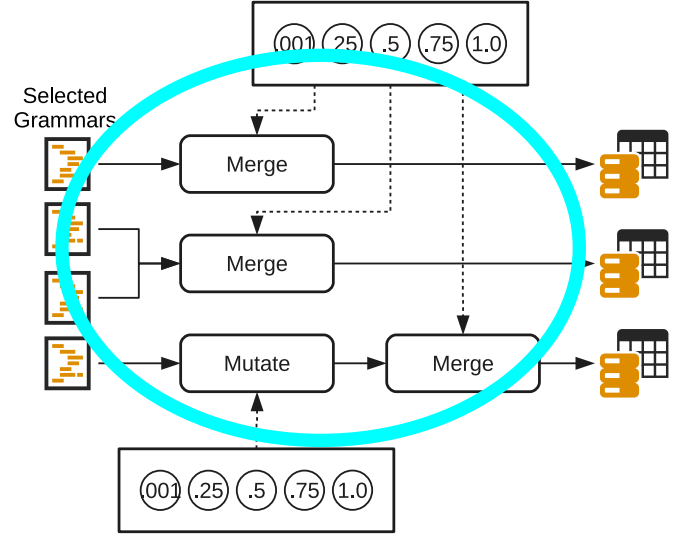


Fig. 6. Data collection procedure.

Using the results of these calculations, we subdivided the dataset into three categories based on the number of productions (as a measure of size of the grammar). This subdivision was based on statistically construction thresholds as follows: ... For each experiment, we randomly selected 5 grammars from each size category. The grammars selected using this process, their metadata, and their associated metrics are shown in Tab. I.

D. Data Collection Procedures

The data collection process is executed as follows, and as depicted in Fig. 6. Using the experimental sets defined in Sec. IV-B we conduct the three experiments as follows: In the case of all three experiments the grammars selected are read into the system. Next, in the case of self-merge, or merge-with-others, the merge process is conducted using the selected similarity threshold; in the case of merge-with-mutants the a set of mutants are first generated using the associated mutant control parameter and then these grammars are paired and merged using the merge process and selected similarity threshold. Finally, the data from the experiment is collected for later analysis.

E. Analysis Procedures

This subsection describes the analysis techniques that will be used in the validation and evaluation of the experiments described in previous subsections.

Both of these experiments utilize a Randomized Complete Block Design (RCBD) [18] with the size category of the grammar 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 Similarity Threshold. We begin by initially validating this assumption via an interaction plot. Upon confirmation of this assumption we select the appropriate analysis techniques. Prior to conducting the experiments, we will conduct a sample size estimation

TABLE I
EVALUATION GRAMMARS AND THEIR PROPERTIES.

Experiment	Category	Grammar	Version	TERM	VAR	PROD	MCC
1	S						
	M						
	L						
2	S						
	M						
	L						

procedure to evaluate 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 i th treatment level and j th block level.
- μ is the baseline mean
- τ_i is the effect of the i th treatment level.
- β_j is the effect of the j th 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, there is the homogeneity of variance assumption. To validate this, we will utilize Levene's Test [19]. Next is the assumption that both the treatments and errors are normally distributed. To verify this assumption, we will use the Anderson-Darling Test for Goodness of Fit [20] to the Normal distribution for both the errors and treatment effects. Finally, there is the assumption of independence of the observations.

In the case of violations of these assumptions we will attempt to correct the violation. In the case that the homogeneity of variance has been violated, we will attempt to correct this via either Box-Cox [21] transformations or via the application

of a weighted least squares approach. Similarly, to correct issues with the normality of the errors we will attempt a transformation of the data. 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 corrected, we will utilize the Friedman Test [22] instead. Though a non-parametric technique, 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 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 then conduct a multiple-comparison procedure to compare the individual effects of each level of the Similarity Threshold. Specific to these experiments, we consider the value 0.001 to be the control, and we have selected to pair Dunnett's [23] multiple

include hypotheses

comparison procedure if ANOVA is used, and Steel's [24] 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.

Finally, for either approach we will conduct a power analysis to ensure the Type-II error rate is within tolerance. This final analysis is to ensure 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. [25] we are concerned with conclusion, internal, construct, and external validity threats. The following sub-subsections detail the procedures/processes used to ensure the validity of our experiments.

Conclusion Validity: To ensure the conclusion validity we have implemented the following as part of our experimental design:

- Due to the nature of the data, we cannot guarantee that the assumptions for a typical ANOVA test associated with each experiment can be met, thus we have selected to utilize non-parametric variants of these tests.
- During the execution of the analysis we have selected to evaluate the power of the tests.
- Each of the selected metrics' implementations have been evaluated via unit and integration testing of their components prior to the execution of the experiments to ensure accurate results.
- The application of the treatments are controlled by the experimental system and this system has been thoroughly tested to ensure proper operation.
- The error rate is controlled when conducting multiple analyses using either techniques which include adjustment procedures or by applying 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: To ensure the internal validity of the experiments we have implemented the following:

- Due to the nature of the experimental system the timing of execution has no effect on the outcome.
- There is no effect on the outcome when repeating the same experiment.
- We have selected grammars that we feel are representative for the size, but this may pose a threat due to under-representation.
- The analysis procedures used in these experiments ensure that there is no ambiguity regarding the direction of causal influence.
- There are no issues regarding differences in behavior between the groups.
- Finally, due to the nature of these experiments there are no social threats to internal validity.

Construct Validity: To ensure the construct validity of the experiments we have implemented the following procedures:

- Due to the nature of the experiments there are no social threats to construct validity.
- The design of the experiments prevents both mono-operation and mono-method biases.
- As per the underlying implementation of the approach and its normalization procedures there is no threats due to inadequate preoperational explication or confounding constructs.
- Additionally, the levels associated with the constructs are designed to evaluate the spectrum rather than whether the construct exists or not. Although each subject is in each experiment, the nature of the experiments prevents interaction of treatments.
- Similarly, there is no issue in with the application of treatments to the grammars and the grammars improving due to the treatment.
- The treatments do not directly modify the actual grammars and thus, cannot unintentionally affect other subjects not currently understudy.

External Validity: To ensure the construct validity of the experiments we have implemented the following procedures:

- The selection of simple languages (such as the group of small languages) not currently in use in industry could pose a threat to external validity, but this is offset by the inclusion of the other languages from both the medium and large groups.
- We do restrict our internal representations to BNF, but we do include the ability to utilize Antlr grammars which are currently used in practice. Though a threat to external validity still exists as we did not include grammars in SDF or TXL formats as well, thus this threat remains.
- There is no effect of time or date of treatment on the grammars.

V. RESULTS

A. Data Set Reduction

B. Descriptive Statistics

This subsection describes the data collected via a series of descriptive statistics.

1) Self-Merge:

2) Other-Merge:

3) Mutant-Merge:

C. Hypothesis Testing

1) Self-Merge:

2) Other-Merge:

3) Mutant-Merge:

VI. INTERPRETATION

A. Evaluation of Results

B. Limitations of Study

In each experiment conducted as part of this study the following limitations apply. The selection of the subjects was

not a random sample and thus we are unable to generalize, statistically, to the greater population of grammars. Yet, the assignment of subjects to treatment groups was conducted at random and thus provides the capability for causal analysis.

C. Inferences

D. Lessons Learned

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 and their specific types of threats as detailed by Wohlin et al. [25]. Given the processes and procedures detailed in Sec. IV-F there are 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 based their size and availability, but this poses a threat due to potential underrepresentation of the domain. Additionally, since we restrict our internal representations to BNF, but we do not include the ability to utilize TXL [26] or SDF [27] grammars (which are extensively used in software language engineering) there is a threat to external validity.

VIII. CONCLUSIONS AND FUTURE WORK

A. Summary of Findings

B. Relation to Existing Evidence

C. Impact

D. Limitations

E. Future Work

Immediate avenues for future work are as follows. We intended to conduct further studies to improve the results herein by randomly selecting grammars from the Grammar-Zoo. 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. Synytsky, 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 baseline 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. Nierstras, "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*. Saarbrücken, Germany: ACM Press, 2016, pp. 449–452. [Online]. Available: <http://dl.acm.org/citation.cfm?doi=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] D. C. Montgomery, *Design and analysis of experiments*, eighth edition ed. Hoboken, NJ: John Wiley & Sons, Inc, 2013.
- [19] 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.
- [20] 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>
- [21] 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. 211–252, 1964. [Online]. Available: <http://www.jstor.org/stable/2984418>
- [22] 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>
- [23] 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>

- [24] 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>
- [25] 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>
- [26] 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>
- [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>