XRESE/2019/001

Multilingual Parsing

Idaho State University
Computer Science

Work-Package 1: “Tech Report”

SIGMA – Systematic Island Grammar forMation Approach

Isaac Griffith

Oct 2019

XR|ESE

SIGMA – Systematic Island Grammar forMation Approach

Isaac Griffith

eXtended Reality and Empirical Software Engineering Lab
Department of Informatics and Computer Science
Idaho State University

Preliminary Report

Prepared for XRESE

Table of Contents

Figures and Tables

Figures

Tables

List of Algorithms

1 Introduction

The primary paradigm of software application development has been moving steadily away from the homogenous single language and single technology approach and towards a heterogenous multilingual and multi-technology approach for sometime now [@http://zotero.org/users/1776655/items/B8D2E2AR
Simultaneosly the use of static and dynamic analysis tools as a standard part of a developer's and organizations toolkit has seen rapid growth. Yet, such tools such as SpotBugs[1], PMD[2], Parasoft dotTEST[3], Pylint[4], etc. have been designed with a single language or family of languages in mind. Services and software issue accumulation tools such as SonarQube[5] have shown that there is a need for static and dynamic analysis tools to be capable of crossing the boundaries of languages. The typical design paradigm for such tools involves the inclusion of multiple languages via plugins (or some other technique) which requires the use of one or more (language specific) parsers to extract knowledge from the code. Unfortunately, this causes two main issues: 1.) A maintenance nightmare for the inclusion and integration of multiple parsers. 2.) A maintenance headache as languages continue to update and change.

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

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[TM6]).

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 to maintain as these grammars evolve. 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 this paper we discuss a key 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.

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):

[1] https://spotbugs.github.io
[2] https://pmd.github.io
[3] https://www.parasoft.com/products/dottest
[4] https://www.pylint.org
[5] https://www.sonarqube.org
[6] https://sonarqube.org

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

The solution proposed will allow tool designers to develop source code analysis tools which support multiple programming languages in an efficient and maintainable way.

1.1 Organization

The remainder of this paper is organized as follows. **??** discusses the theoretical foundations and alternative approaches related to this work. **??** 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. **??** details the design of experiments which evaluate the proposed approach and its parameterization. **??** presents the results of the experiments. **??** presents a discussion of the results and their interpretation in light of other studies. **??** details the threats to the validity of this study. Finally, this paper is concluded in **??**.

2 Background

2.1 Theoretical Foundations

2.1.1 Grammars

A context free grammar, G, 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, and $S \in V$ is the starting symbol. Each production is written as $a \rightarrow b$ with $a \in V$ and $b \in (V \cup \Sigma)^*$. When b is the empty string, the production is denoted by $a \rightarrow \varepsilon$. A string, s, is called a *valid sentence* for a grammar if it can be created by repeated application of the productions of that grammar [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].

2.1.2 Grammar Metrics

2.2 Alternative Approaches

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 created from languages that do not share a common IR [X]. 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 analyzing multilingual systems is through the use of a separate parser for each language [A-Z]. Constructing and integrating each parser for each language can be both difficult and time consuming, 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 one contends with in this approach is these parsers may be written in a variety of languages. The use of an existing Island Grammar[2] may alleviate these issues, but typically they tend to be specific to their particular problem and their construction is currently a time-consuming manual process.

2.3 Research Contributions

In this paper we develop the following contributions:

1. The proposal of an algorithmic approach to the generation of island grammars from multiple grammars.
2. Case study results concerning island grammar generation and grammar measurement.
3. A tool which provides the capability to automate the proposed processes.

3 Algorithms

3.1 Normalization Algorithm

The following algorithm defines the approach for normalizing a given grammar. The normalization process defined here facilitates the ability to merge productions, in pursuit of the overarching goal of automated generation of Island [X], Tolerant [X], Bridge [X], and Bounded Seas [X] grammars.

This algorithm assumes that the source grammar, G, was initially in some defined formalism such as Antlr [X], EBNF [X], BNF [X], SDF [X], TXL [X], etc. The grammar was then read in and processed to conform to the metamodel depicted in Figure 1. Assuming that the grammar meets this condition, the goal of this algorithm is then to reformat the grammar such that each production is of one of the following forms:

- $Form_1$: A production composed of a rule containing at least one symbol, and where all symbols are concatenated with one another. An example is: $\langle A \rangle$ ::= $\langle B \rangle$ 'a' . . ..
- $Form_2$: A production composed of an alternation of one or more symbols. An example is: $\langle B \rangle$ ::= $\langle A \rangle$ | 'b' | . . . | ε.

The normalization process, as defined in Algorithm 1, repeatedly executes six processes until the grammar stabilizes. These six processes are: i) eliminating unused rules, ii) simplifying productions, iii) merging equivalent rules, iv) eliminating unit rules, v) expanding productions, and vi) collapsing compatible productions.

3.1.1 Eliminating Unused Rules

This process removes all productions that are not produced, directly or indirectly, from the start production. This is accomplished by marking all productions enumerated via a depth first search (see Algorithm 3), and then dropping unmarked productions, as shown in Algorithm 2. When applied to G_3 the grammar is transformed into grammar G_4, as depicted in Figure 2a.

3.1.2 Simplifying Productions

Algorithm 1 Normalization Algorithm

```
1:  procedure NORMALIZE(G)
2:      repeat
3:          G ← ELIMINATEUNUSEDPRODUCTIONS(G)
4:          G ← SIMPLIFYPRODUCTIONS(G)
5:          G ← MERGEEQUIVPRODUCTIONS(G)
6:          G ← ELIMINATEUNITPRODUCTIONS(G)
7:          G ← EXPANDPRODUCTIONS(G)
8:          G ← COLLAPSEPRODUCTIONS(G)
9:      until UNCHANGED(G)
10: end procedure
```

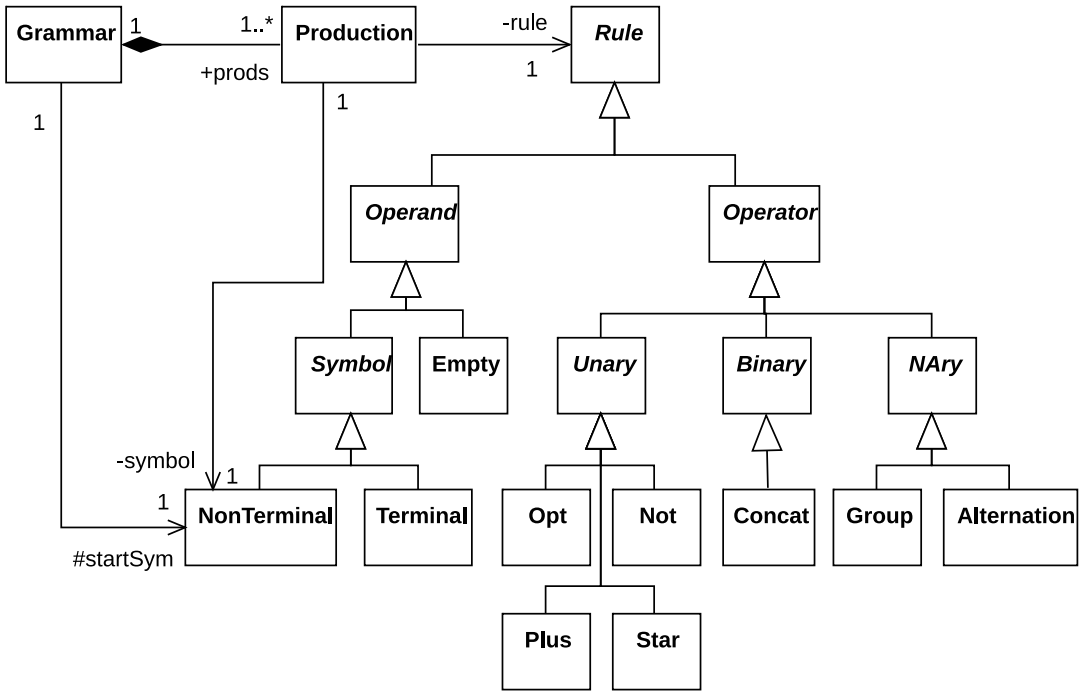


Figure 1. Grammar metamodel.

This process aims to simplify productions. This is achieved by first removing ε's embedded inside rules. Once this is complete productions containing a single term are replaced with that term. This process is embodied in Algorithm 4. When to grammar G_4, it is transformed into grammar G_5, as depicted in 2b.

3.1.3 Merging Equivalent Productions

Productions that have identical rules are replaced by a single production. This new production is given a name derived from the productions that were merged to create it. In the following example of this step, rules a and b are merged into the rule a+b:

$$\begin{aligned} \langle \mathrm{s} \rangle &\models \langle \mathrm{a} \rangle \mid \langle \mathrm{b} \rangle \\ \langle \mathrm{a} \rangle &\models \mathsf{a}\ \mathsf{b}\ \langle \mathrm{a} \rangle \\ \langle \mathrm{b} \rangle &\models \mathsf{a}\ \mathsf{b}\ \langle \mathrm{a} \rangle \end{aligned}$$

Which after equivalent rules are merged yields:

$$\begin{aligned} \langle \mathrm{s} \rangle &\models \langle \mathrm{a+b} \rangle \\ \langle \mathrm{a+b} \rangle &\models \mathsf{a}\ \mathsf{b}\ \langle \mathrm{a+b} \rangle \end{aligned}$$

When this step is applied to grammar G_5, it is transformed into grammar G_6, as depicted in Figure 2c.

3.1.4 Eliminating Unit Productions

Algorithm 2 Eliminate Unused Productions

```
1:  function EliminateUnusedProductions(G)
2:      H ← (V, E)
    ▷ Create empty graph
3:      for all p ∈ G.Σ do
4:          H.V ← H.V ∪ {p}
5:      end for
6:      for all p ∈ G.Σ do
7:          for all nt ∈ p.rule do
    ▷ Create directed edge between non-terminals
8:              H.E ← H.E ∪ {(p, nt)}
9:          end for
10:     end for
11:     DFSMark(G.S)
12:     for all v ∈ H.V do
13:         if !isMarked(v) then
14:             G.Σ ← G.Σ − {v.value}
15:         end if
16:     end for
17: end function
```

Algorithm 3 Depth First Marking

```
1:  function DFSMark(start)
2:      S ← [start]
3:      while S ≠ ∅ do
4:          p ← pop(S)
5:          Mark(p)
6:          for all s ∈succ(p) do
7:              if !isMarked(s) then
8:                  push(S, s)
9:              end if
10:         end for
11:     end while
12: end function
```

All non-terminals with productions of one of the following two forms will have their non-terminal symbols replaced by their rules, and their productions eliminated.

$$\langle \mathrm{a} \rangle \models \langle \mathrm{b} \rangle$$
$$\langle \mathrm{a} \rangle \models \mathsf{a}$$

Elimination of productions of the first form, is derived from Chomsky Normal Form (CNF) [X]. Eliminations of productions of the second form, a derivation from CNF, allows the simplification process to simplify rules of the following form:

$$\langle \mathrm{a} \rangle \models \langle \mathrm{b} \rangle \ \mathsf{a}\ \mathsf{b}$$
$$\langle \mathrm{b} \rangle \models \epsilon$$

Algorithm 4 Simplify Productions

```
1:  function SimplifyProductions(G)
    ▷ Remove embedded ε's
2:      for all p ∈ G.Σ do
3:          if contains(p, concat(ε)) then
4:              remove(p, ε)
5:          end if
6:      end for
    ▷ Replace one-term productions
7:      for all p ∈ G.Σ do
8:          if |p.rule| = 1 then
9:              replace(uses(p), p.rule)
10:         end if
11:     end for
12: end function
```

Algorithm 5 Merge Equivalent Productions

```
1:  function MergeEquivProductions(G)
2:      pairs ← ∅
3:      for i ∈ [0, |G.Σ|) do
4:          for j ∈ (i, |G.Σ|) do
5:              if i ≠ j then
6:                  pairs ← pairs ∪ (G.Σ[i], G.Σ[j])
7:              end if
8:          end for
9:      end for
10:     for all p ∈ pairs do
11:         if p.left.rule = p.right.rule then
12:             CombineAndReplace(p.left, p.right)
13:         end if
14:     end for
15: end function
```

When this step is applied to grammar G_6, it is transformed into grammar G_7, as depicted in Figure 2d.

3.1.5 Expanding Productions

Productions that have nested rules have all nested content replaced by with a non-terminal. The new non-terminal defines a production pointing to their content. When this step is applied to grammar G_7, it is transformed into grammar G_8, as depicted in Figure 2e.

3.1.6 Collapsing Compatible Productions

The final step of the normalization process combines productions that are compatible with each other. This ensures that any non-terminal symbols referenced in a rule will not define a duplicate production. The following provides an example:

Algorithm 6 Eliminate Unit Productions

```
1: function EliminateUnitProductions(G)
2:     for all p ∈ G.Σ do
3:         if |p.rule| = 1 then
4:             replace(uses(p), p.rule)
5:         end if
6:     end for
7: end function
```

Algorithm 7 Expand Productions

```
1: function ExpandProductions(G)
2:     repeat
3:         changed ← ⊥
4:         for all p ∈ G.Σ do
5:             if isConcat(p.rule) then
6:                 for all g ∈ p.rule do
7:                     if isGroup(g) then
8:                         createAndReplaceWithProd(g)
9:                         changed ← ⊤
10:                    end if
11:                end for
12:            else if isAlt(p.rule) then
13:                for all a ∈ p.rule do
14:                    createAndReplaceWithProd(a)
15:                    changed ← ⊤
16:                end for
17:            end if
18:        end for
19:    until changed = ⊥
20: end function
```

$$
\begin{aligned}
\langle \mathrm{A} \rangle &\models \mathtt{a}\ \langle \mathrm{B} \rangle \\
\langle \mathrm{B} \rangle &\models \mathtt{b}\ \mathtt{c} \\
\langle \mathrm{C} \rangle &\models \mathtt{c} \mid \langle \mathrm{D} \rangle \\
\langle \mathrm{D} \rangle &\models \mathtt{d} \mid \mathtt{e}
\end{aligned}
$$

would then collapse to form:

$$
\begin{aligned}
\langle \mathrm{A} \rangle &\models \mathtt{a}\ \mathtt{b}\ \mathtt{c} \\
\langle \mathrm{C} \rangle &\models \mathtt{c} \mid \mathtt{d} \mid \mathtt{e}
\end{aligned}
$$

When this step is applied to grammar G_8, it is transformed into grammar G_9, as depicted in Figure 2f.

Algorithm 8 Collapse Productions

```
1: function CollapseProductions(G)
   ▷ Split productions into form1 and form2
2:     f1 ← Collect("form1")
3:     f2 ← Collect("form2")
4:     for all p ∈ f1 do
5:         if OnlyTerminals(p.rule) then
6:             ReplaceF1UsesWithRule(p)
7:         end if
8:     end for
9:     for all p ∈ f2 do
10:        if OnlyTerminals(p.rule) then
11:            ReplaceF2UseWithRule(p)
12:        end if
13:    end for
14: end function
```

Algorithm 9 Merge Algorithm

```
1: procedure MergeProductions(N, t)
2:     f1 ← collect('form1', N)
3:     f2 ← collect('form2', N)
4:     p1 ← Pairs(f1)
5:     p2 ← Pairs(f2)
6:     Q1 ← Measure(p1, t, S1)
7:     Q2 ← Measure(p2, t, S2)
8:     while Q1 ≠ ∅ ∨ Q2 ≠ ∅ do
9:         Process(Q1, f1, S1)
10:        Process(Q2, f2, S2)
11:    end while
12: end procedure
```

3.2 Merging Algorithm

The following algorithms define the approach for merging similar productions of a grammar. This approach assumes that the grammar has been normalized such that each production in the grammar is one of the following two forms:

- *Form*$_1$: A production composed of a rule containing at least one symbol, and where all symbols are concatenated with one another. An example is: $\langle A \rangle$ ::= $\langle B \rangle$ 'a' ....
- *Form*$_2$: A production composed of an alternation of one or more symbols. An example is: $\langle B \rangle$ ::= $\langle A \rangle$ | 'b' | ... | ε.

The algorithm initially separates all existing productions of the normalized grammar, $\mathcal{N}$ into two disjoint sets of productions based on the form the productions take. These sets are then used to compose pairs of productions which will be compared for similarity. This process uses a simple nested loop approach (the function *Pairs* in lines 13–21 in Algorithm 14). These pairs are then used to form two priority queues (one per form type) via the *Measure* function in Algorithm 14.

3.2.1 Measure

Algorithm 10 Form Pairs

```
1: function PAIRS(list)                                   ▷ O(n log n)
2:     x ← 0
3:     for i ∈ [0, length(list)) do
4:         for j ∈ (i, length(list)) do
5:             list[x] ← (list[i], list[j])
6:             x ← x + 1
7:         end for
8:     end for
9: end function
```

Algorithm 11 Construct Priority Queues

```
1: function MEASURE(data, t, func)                        ▷ O(n log n)
2:     Q ← []
3:     for all p ∈ data do
4:         s ← func(p.a, p.b)
5:         if s ≥ t then
6:             OFFER(Q,⟨p, t⟩)
7:         end if
8:     end for
9:     return Q
10: end function
```

The measure function walks through the list of pairs provided to determine which will be considered during the actual merging process. For each pair of productions, a similarity score is measured, depending on the form. For productions of type $Form_1$ the similarity is measured using equation 1. This equation compares the size of the aligned components of two production rules to the total size of the productions.

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

The function *aligned* aligns two rules as follows. A data structure composed of two doubly linked lists, whose elements are the symbols of a rule. Items in each list which contain the same symbol are then linked with alignment links. Initially, any alignment link which crosses another is removed. This continues, until no links can be removed. The value used in this equation is then the number of links remaining in the data structure. This value is doubled in order to account for the number of connected nodes rather than edges.

The similarity of productions of type $Form_2$ are measured using Equation 2. In this equation we simply calculate the size of the intersection of symbols of each production divided by the total

Algorithm 12 Measures S_1 and S_2

```
1: function S1(a, b)
2:     return 2|aligned(a,b)| / |a|+|b|
3: end function

4: function S2(a, b)
5:     return 2|a∩b| / |a|+|b|
6: end function
```

Algorithm 13 Process Productions

```
1: function PROCESS(Q, data, func)
2:     while Q ≠ ∅ do
3:         p ← POLL(Q)
4:         V ← MERGE(p.pair)
5:         Q ← UPDATE(Q, V, p, data, func)
6:     end while
7: end function
```

number of symbols in both productions. We again double the numerator to account for the use of common symbols in both productions.

$$S_2 = \frac{2|A \cap B|}{|A| + |B|} \quad (2)$$

Finally, in either case we compare the similarity of the two productions to a known threshold value, t. If the value is less than the threshold, we disregard the pair, else we offer it to a priority queue (which is sorted on similarity). The priority queues will be utilized during the merging process.

3.2.2 Process

The process function, shown in Algorithm 13, processes a priority queue of pairs of productions. The priority queue is sorted on the similarity between the two productions to be merged. The most similar pair is initially polled from the queue and then merged to form a new set of normalized productions. These normalized productions are then used to update the queue based on the provided similarity scoring function. This process continues until the priority queue is empty.

3.2.3 Merge

The merge function, shown in Algorithm 14, is where the actual merging of productions occurs. This process is dependent on whether the pair of productions to be merged are of $Form_1$ or $Form_2$. In the former case, an alignment list data structure is created containing the symbols from each production. This data structure is two parallel linked lists (used to maintain the order of the original symbols). The elements in common between the two lists are then linked by what are called "alignment edges". These edges form a bipartite graph between the nodes of the lists.

Alignment edges are then processed to determine if they cross any other edges. This is determined by comparing edges with the largest difference in list indexes against all other edges. If it is determined that a crossing has occurred, the largest difference edge is removed. At this point only edges which can be aligned will remain and the alignment process begins.

The alignment process works as follows. For each edge, starting with the edge with the smallest index (on one side), if the indexes on both sides of the list are not equal, an empty string node is inserted into the list with the node of lesser index. This process continues until all edges are aligned. Once all edges are aligned, pairs are formed by combining the nodes from both lists at the same index. If the nodes contain different symbols, they are grouped by alternation and then concatenated to the rule.

In the case of $Form_2$ productions. The pair is merged by the union of symbols between the two productions. This forms an alternation between each of the symbols in the union.

In both cases, once the merge is completed the new production is then normalized (which may produce multiple new productions in the case of $Form_1$ productions). This set of productions is then returned.

3.2.4 Update

Finally, once a pair of productions have been merged and their result normalized the priority queue and grammar must be updated to account for the change. The process for this is defined by the Update function shown in Algorithm 15. Initially, the data set containing the pair of merged productions is updated by removing the pair. Then the priority queue is updated by identifying all pairs in the queue which contain either of the pair that was merged and if found such pairs are removed from the queue. The grammar is updated to replace all uses of the merged productions are replaced with the newly created merged production. Finally, Using the remaining set of productions of the same form as the merged production, the new productions are then paired with all remaining productions in the set and the similarity between pairs is measured. If the similarity is above the similarity threshold, then the pair is added to the priority queue.

3.2.5 Outcome

The outcome of this algorithm is that those productions with a similarity of at least t are combined into single productions, and then re-normalized to one of the two forms noted at the beginning of Section 3.2. It is plain to see that the mechanics of this process will ensure that for a normalized grammar, G for some language L_G, that the resulting grammar, G', will produce a language $L_{G'} \supseteq L_G$. This is due to the fact that in the case of $Form_1$ merges the order of the symbols is maintained, and for those symbols which cannot be paired an alternation with the empty string is injected. This is equivalent to constructing an optional symbol in EBNF which relaxes the resulting grammar, but which also maintains the capability of the previous grammar.

In the case of $Form_2$ productions, it is plain to see that the merging maintains a similar property as well. In this case both productions are simply an alternation of one or more productions and hence the union of two alternations will relax the resulting grammar, but maintains the capabilities of the source grammar.

3.3 Islandization

This section describes the methods we have developed to automate the generation of multilingual island grammars. In the following subsections we detail the algorithms we have designed to complete this task. We then describe the operation of these algorithms on simple grammars. Finally, we detail the design of the tool which implements these algorithms and provides a platform to facilitate data collection associated with experimentation with these algorithms.

3.3.1 Naive Island Grammar Generation

The experiment detailed in **??** is designed to evaluate the results of applying the previous algorithm to a grammar. In order to evaluated its effect we are in need of a control treatment which acts as if nothing was effectively applied. In this case, we need some means by which we can combine multiple grammars together into a single grammar, but leave them intact. Thus, we have developed what we call the *Naive Island Grammar Generation* algorithm, depicted in 17. This algorithm is quite simple in its operation. In lines 2–4 of the algorithm the three set components, V, Σ, P are constructed via a union across the same components of each of the grammars to be combined. In line 5 of the algorithm, a start production is constructed for the new grammar and its body is the union set of start symbols from all combined grammars separated by

alternation symbols. Finally, in lines 6 and 7 the new grammar is constructed as a tuple of initial components and then returned as the results of the algorithm.

3.3.2 Interest Configuration Generation

Key to the creation of an Island Grammar is the set of interested components. As part of the design of the approach to generate island grammars we developed a model to represent these sets. This also included the definition of an XML schema to represent these concepts outside of the code. The underlying model for these is depicted in 3.

As part of the experiment detailed in Section **??** we need the ability to generate interest configurations for each grammar. We have developed a simple algorithm, depicted in 18, to do just that. The algorithm takes two parameters G the initial grammar and n the number of interest components to extract. Using this information, the algorithm constructs an empty list, P which is then filled with the non-terminal symbols of the grammar. This list is then randomly shuffled and the top n are selected and returned.

3.4 Tolerization

3.5 Bounding The Seas

3.6 Bridgization

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle S_1 \rangle \mid \langle S_2 \rangle \\
\langle S_1 \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a} \in \langle \mathrm{B} \rangle \langle C_1 \rangle \\
\langle C_1 \rangle &\models \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle S_2 \rangle &\models \langle C_2 \rangle \mid \langle \mathrm{D} \rangle \\
\langle C_2 \rangle &\models \mathsf{c} \\
\langle \mathrm{D} \rangle &\models \mathsf{a}\ \mathsf{d}\ (\mathsf{e} \mid \mathsf{c}) \mid (\langle C_2 \rangle \mid \mathsf{b})
\end{aligned}
$$

(a) Grammar G_4.

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle S_1 \rangle \mid \langle S_2 \rangle \\
\langle S_1 \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a}\ \langle \mathrm{B} \rangle \langle C_1 \rangle \\
\langle C_1 \rangle &\models \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle S_2 \rangle &\models \langle C_2 \rangle \mid \langle \mathrm{D} \rangle \\
\langle C_2 \rangle &\models \mathsf{c} \\
\langle \mathrm{D} \rangle &\models \mathsf{a}\ \mathsf{d}\ (\mathsf{e} \mid \mathsf{c}) \mid (\langle C_2 \rangle \mid \mathsf{b})
\end{aligned}
$$

(b) Grammar G_5.

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle S_1 \rangle \mid \langle S_2 \rangle \\
\langle S_1 \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a}\ \langle \mathrm{B} \rangle\ \langle \mathrm{C} \rangle \\
\langle \mathrm{C} \rangle &\models \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle S_2 \rangle &\models \langle \mathrm{C} \rangle \mid \langle \mathrm{D} \rangle \\
\langle \mathrm{D} \rangle &\models \mathsf{a}\ \mathsf{d}\ (\mathsf{e} \mid \mathsf{c}) \mid (\langle \mathrm{C} \rangle \mid \mathsf{b})
\end{aligned}
$$

(c) Grammar G_6.

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle S_1 \rangle \mid \langle S_2 \rangle \\
\langle S_1 \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a}\ \langle \mathrm{B} \rangle\ \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle S_2 \rangle &\models \mathsf{c} \mid \langle \mathrm{D} \rangle \\
\langle \mathrm{D} \rangle &\models \mathsf{a}\ \mathsf{d}\ (\mathsf{e} \mid \mathsf{c}) \mid (\mathsf{c} \mid \mathsf{b})
\end{aligned}
$$

(d) Grammar G_7.

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle S_1 \rangle \mid \langle S_2 \rangle \\
\langle S_1 \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a}\ \langle \mathrm{B} \rangle\ \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle S_2 \rangle &\models \mathsf{c} \mid \langle \mathrm{D} \rangle \\
\langle \mathrm{D} \rangle &\models \langle D_1 \rangle \mid \langle D_3 \rangle \\
\langle D_1 \rangle &\models \mathsf{a}\ \mathsf{d}\ \langle D_2 \rangle \\
\langle D_2 \rangle &\models \mathsf{e} \mid \mathsf{c} \\
\langle D_3 \rangle &\models \mathsf{c} \mid \mathsf{b}
\end{aligned}
$$

(e) Grammar G_8.

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\models \langle \mathrm{A} \rangle \mid \langle \mathrm{B} \rangle \mid \mathsf{c} \mid \langle \mathrm{D} \rangle \\
\langle \mathrm{A} \rangle &\models \mathsf{a}\ \mathsf{b}\ \mathsf{d}\ \mathsf{c} \\
\langle \mathrm{B} \rangle &\models \mathsf{b}\ \mathsf{d} \\
\langle \mathrm{D} \rangle &\models \langle D_1 \rangle \mid \mathsf{c} \mid \mathsf{b} \\
\langle D_1 \rangle &\models \mathsf{a}\ \mathsf{d}\ \langle D_2 \rangle \\
\langle D_2 \rangle &\models \mathsf{e} \mid \mathsf{c}
\end{aligned}
$$

(f) Grammar G_9.

Figure 2. Transformed grammars produced during the normalization of grammar G_3.

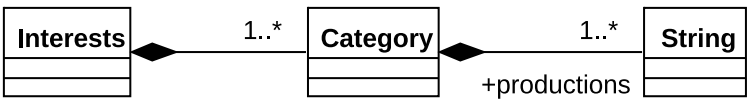


Figure 3. Interest Configuration Model

Algorithm 14 Merge Algorithm

```
1:  function MERGE(pair)
2:      if pair is of form1 then
3:          list ← CREATEALIGNMENTLIST(pair)
4:          for all e ∈ list.alignments do
5:              if ISCROSSING(e) then
6:                  DELETE(e)
7:              end if
8:          end for
9:          repeat
10:             changed ← ⊥
11:             for all e ∈ list.alignments do
12:                 if e.left > e.right then
13:                     INSERT(list.right, ε, e.right)
14:                     changed ← ⊤
15:                 else if e.left < e.right then
16:                     INSERT(list.left, ε, e.left)
17:                     changed ← ⊤
18:                 end if
19:             end for
20:         until changed = ⊥
21:         for all p ∈ list do
22:             if p.left = p.right then
23:                 rule ← rule · p.left
24:             else if p.left = ε then
25:                 rule ← rule · (p.right|p.left)
26:             else
27:                 rule ← rule · (p.left|p.right)
28:             end if
29:         end for
30:     else
31:         v ← ∅
32:         v.symbol ← a.symbol + ' ' + b.symbol
33:         v.rule ← a.rule ∪ b.rule
34:     end if
35:     V ← NORMALIZE(v)
36:     return V
37: end function
```

Algorithm 15 Update Algorithm

```
1:  function UPDATE(Q, v, p, data, func)                    ▷ O(n log n)
2:      data ← data − {p.a, p.b}
3:      for all i ∈ Q do
4:          if contains(i, p.a) ∨ contains(i, p.b) then
5:              REMOVE(Q, i)
6:          end if
7:      end for
8:      for all d ∈ data do
9:          s ← func(v, d)
10:         if s ≥ t then
11:             OFFER(Q, ⟨d, v⟩)
12:         end if
13:     end for
14:     data ← data ∪ {v}
15:     for all p ∈ N do
16:         if CONTAINS(p, a) then
17:             REPLACEANDNORMALIZE(p, a, v)
18:         else if CONTAINS(p, b) then
19:             REPLACEANDNORMALIZE(p, b, v)
20:         end if
21:     end for
22: end function
```

Algorithm 16 Islandization

```
1:  procedure ISLANDIZATION(G, I)
    ▷ for each interest item in I mark corresponding productions in G as interest
2:      for all i ∈ I do
3:          p ← findPRODUCTION(G, i)
4:          markASINTEREST(p)
5:      end for
    ▷ Create fresh Start Production, Water Production, and Island Production in new grammar G'
6:      S ← (W | I)
7:      G' ← (V, Σ, P, S)
    ▷ Construct Island Production I
8:      for all p ∈ MARKED(G) do
9:          G'.Σ ← G'.Σ ∪ {p}
10:         I ← I ∪ {| P}
11:         PROCESS(P, G', G)
12:     end for
    ▷ Construct Water Production W
13:     for all p ∉ MARKED(G) do
14:         W ← W ∪ (| P)
15:         G'.Σ ← G.Σ ∪ {p}
16:     end for
17:     W ← reduceANDMERGE(W)
18:     return G'
19: end procedure

20: function PROCESS(p, G', G)
21:     if p ∈ G' ∨ lexerRULE(p) then
22:         return
23:     else
24:         for all s ∈ p ∧ s ∈ G.Σ ∧ s ∈ MARKED(G) do
25:             G'.Σ ← G'.Σ ∪ {prod(G, S)}
26:             PROCESS(prod(G, S), G', G)
27:         end for
28:     end if
29: end function

30: function REDUCEANDMERGE(G)
31:     G' ← NORMALIZE(G)
32:     MERGEPRODUCTIONS(G')
33: end function
```

Algorithm 17 Naive Island Grammar Generation

1: **procedure** NAIVEGEN($\mathcal{G}$)
▷ Initialize components of Island Grammar to be combinations of components of input grammars
2: $V \leftarrow \bigcup_{g \in \mathcal{G}} V_g$
3: $\Sigma \leftarrow \bigcup_{g \in \mathcal{G}} \Sigma_g$
4: $P \leftarrow \bigcup_{g \in \mathcal{G}} P_g$
▷ Create fresh start production as alternation among input grammar starts
5: $S \leftarrow \bigcup_{g \in \mathcal{G}} S_g$
▷ Construct and return the grammar
6: $G' \leftarrow (V, \Sigma, P, S)$
7: **return** G'
8: **end procedure**

Algorithm 18 Interest Configuration Generation

1: **procedure** GENERATE(G, n)
2: $P \longleftarrow [\,]$
3: $P \longleftarrow G.\Sigma / G.S$
4: SHUFFLE(P)
5: **return** $P[1..n]$
6: **end procedure**

Algorithm 19 Tolerization

1: **procedure** TOLERIZATION(G, I)
2: **end procedure**

Algorithm 20 BoundingTheSeas

1: **procedure** BOUNDINGTHESEAS(G, I)
2: **end procedure**

Algorithm 21 Bridgization

1: **procedure** BRIDGIZATION(G, I)
2: **end procedure**

4 Tool

This chapter describes the design and implementation of the tool to automate the generation of Island, Tolerant, Bridge, and Bounded Seas grammars.

4.1 Design

4.2 Implementation

5 Experimental Designs

5.1 Experiment 1: Evaluating Normalization

5.1.1 Goals, Hypotheses, Variables

Following the GQM paradigm, research goal RG1 can be decomposed into the following set of research questions:

RQ1.1 What is the effect of each step in the normalization process on maintenance effort between the source grammar and the normalized grammar?

Rationale:

RQ1.2 What is the effect of each step in the normalization process on the complexity between the source grammar and the normalized grammar?

Rationale:

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

M1.1 Effort – To assess the effort required to maintain a grammar, we utilize the Halstead Effort measure for grammars as defined by Power and Malloy [15].

M1.2 Complexity – To assess the complexity of a grammar, we utilize McCabe's Cyclomatic Complexity metric for grammars defined by Power and Malloy [15].

5.1.2 Design

5.1.3 Experimental Units

In these experiments, the experimental units are individual grammars selected from the Antlr4 [16] grammar repository[7]. The sys-verilog grammar was excluded because of errors while parsing it. At the time of this writing, the repository contained 198 individual grammars from a variety of general purpose and domain specific languages.

The process used to select the grammars for each experiment is depicted in Figure 4a and works as follows. Initially, for each grammar in the repository we collected a combination of metadata and metric measurements. The metadata collected consists of the following information: the language represented by the grammar, the version of that language (if applicable) and the following metrics (selected from the metrics suite by Power and Malloy [15]):

- TERM – the number of terminals.
- VAR – the number of defined non-terminals.

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

- PROD – the number of productions.
- MCC – McCabe's Cyclomatic Complexity.

Using the resulting measures, the grammar dataset was subdivided into three categories (Small, Medium, and Large) based on a statistical thresholding technique. This subdivision is based on the logarithm of PROD measures and yielded the following thresholds: Small-Medium: $10^{\mu_{PROD}-\sigma_{PROD}} = 10^{1.8404-0.5371} = 20.1084$ and Medium-High: $10^{\mu_{PROD}-\sigma_{PROD}} = 10^{1.8404+0.5371} = 238.4995$. Using these threshold each grammar is then grouped into one of the three categories. The size category then becomes a factor in the experimental model.

5.1.4 Data Collection Procedures

Data collection is performed via the experiment execution system specifically developed for these studies. This system is controlled via the control files created during the experimental unit selection process. The data collection process follows the activity diagram depicted in Figure X, and operates as follows.

Initially, the control file is read into the system. Next, for each of the triples in this file, the following occurs: 1.) The normalizer process is setup for the experiment. 2.) The grammar to be normalized is located, read in, and parsed into an instance of the metamodel. 3.) The initial effort or complexity of the grammar is measured and recorded. 4.) The normalization process is applied, which results in a modified grammar. 5.) The effort or complexity of this normalized grammar is measured and recorded.

Once all experimental units have been processed, the results are exported to a data table. This process is repeated for each replication of each experiment. The results form each replication of each experiment are then combined into a single data table. The combined data table is then used during the analysis phase.

5.1.5 Analysis Procedures

5.1.6 Validity Evaluation

5.2 Experiment 2: Evaluating Merging

5.2.1 Goals, Hypotheses, Variables

This subsection describes the refinement of our initial research goal, defined in **??**, 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 analytical procedures. We begin with the research questions and metrics.

Following the GQM paradigm, research goal RG2 can be decomposed into the following set of research questions:

RQ2.1 What is the effect that this process has on the maintenance 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.*

RQ2.2 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:

M2.1 Effort – To assess the effort required to maintain a grammar, we utilize the Halstead Effort measure for grammars as defined by Power and Malloy [15].

M2.2 Complexity – To assess the complexity of a grammar, we utilize McCabe's Cyclomatic Complexity metric for grammars defined by Power and Malloy [15].

The dependent variables in the experiments, as indicated by the above research questions, are maintenance 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:

- ΔHAL – the change in Halstead Effort as measured after the normalization phase prior to the final merge phase, and after the final merge phase and before generating the output grammar.
- ΔMCC – the change in complexity as measured after the normalization phase prior to the final merge phase, and after the final merge phase and before generating the output grammar.

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) [15], and threshold this value into three distinct categories: Small, Medium, and Large, as defined in Section **??**.

5.2.2 Design

To evaluate the approach we elected to conduct two experiments. The first experiment evaluates the effect the approach has on the maintenance 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 Factorial Design, with a single dependent variable (ΔHAL, ΔMCC), a treatment factor $SimilarityThreshold$ and the grouping factor $Size$.

5.2.3 Experimental Units

This section describes the experimental units and the process used to select them. In these experiments, the experimental units are pairs of grammars selected from the Antlr4 [16] grammar repository[8]. The sys-verilog grammar was excluded because of errors while parsing it. At the

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

Table 1. Grammars randomly selected from each size category used in the experiments.

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

time of this writing, the repository contained 198 individual grammars from a variety of general purpose and domain specific languages.

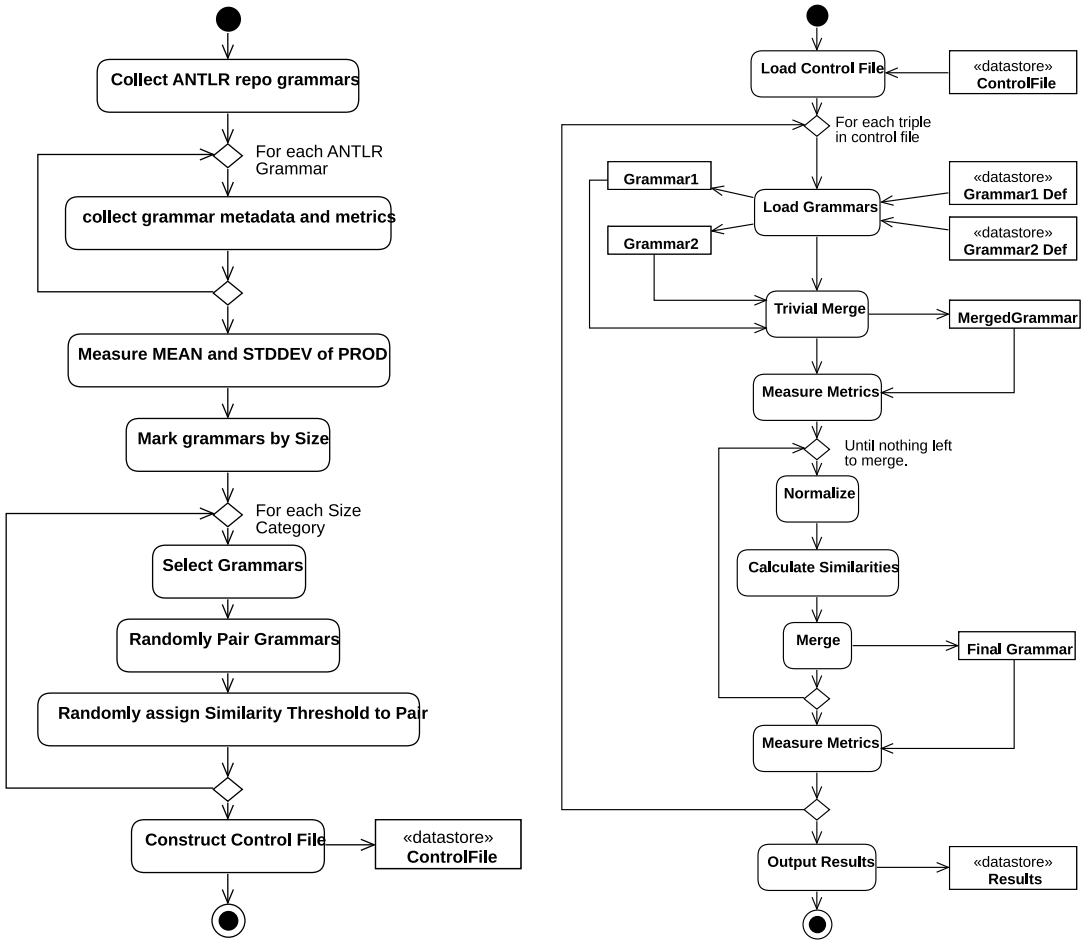
The process used to select the grammar pairs for each experiment is depicted in 4a and works as follows. Initially, for each of the grammars in the repository we collected a combination of metadata and metric measurements. The metadata collected consists of the language represented by the grammar, the version of that language (if applicable) and the following metrics (selected from the metrics suite by Power and Malloy [15]):

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

Using the resulting measures of these metrics, the grammar dataset was subdivided into three categories (Small, Medium and Large) based on the logarithm of PROD (as the values were log-normal distributed). This subdivision was based on statistically construction thresholds, as per Lanza and Marinescu [17]. Category threshold values are defined as: Small-Medium: $10^{\mu_{PROD}-\sigma_{PROD}} = 10^{1.8404-0.5371} = 20.1084$ and Medium-High: $10^{\mu_{PROD}+\sigma_{PROD}} = 10^{1.8404+0.5371} = 238.4995$. Using these thresholds each grammar is then grouped into one of the three categories.

Using these categories as the grouping factor in the experiments, we can then begin the sampling process. As we have selected a 3×5 factorial design, each replication of each experiment requires 15 grammar pairs (5 per size category). Based on our replication analysis (described in Section **??**) we have identified a need for a total of 5 replications. Thus, For each size category we require a total of 50 grammar pairs per experiment yielding a need for 100 total grammar pairs. To meet this requirement we randomly select (without replication) 12 grammars, per size category. From these 12 grammars there are $\binom{12}{2} = 66$ combinations (without replication) of which we randomly select 5 pairs per replication per experiment. The grammars selected and their pairings per experiment/replication are depicted in Table 1.

Once the grammar pairs are 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-pairs are assigned, at random, a level of the similarity threshold treatment. The possible levels that can be assigned are 0.001, 0.25, 0.5, 0.75, and 1.0 (where 1.0 is the control level).



(a) Experimental unit selection process. (b) Data collection procedure.

Figure 4. Activity diagrams for the (a) experimental unit selection process and (b) the data collection process.

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.

5.2.4 Data Collection Procedures

Data collection is performed via the experiment execution system specifically developed for this study. This system is controlled via the control files created during the experimental unit

selection process. The data collection process follows the activity diagram depicted in Figure 4b, as follows.

Initially, the control file is read into the system. Then, 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 final merging process is applied resulting in the final grammar. 5.) 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. This process is repeated for each replication of each experiment. The results from each replication of each experiment are then combined into a single data table. The combined data table is then used during the analysis phase.

5.2.5 Analysis Procedures

As described in Section **??**, both experiments utilize a Factorial design [18]. Typically, a factorial design experiment will utilize ANOVA to determine if there is a difference between the effects of the factors. In this case, the statistical model used is as follows:

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

Where:

- y_{ijk} is the kth value of the observation (either ΔMCC or ΔHAL) associated with the ith similarity threshold level and jth size level.
- μ is the baseline mean
- st_i is the ith level of similarity threshold effect.
- $size_j$ is the jth level of size effect.
- $(st * size)\_ij$ is the similarity threshold * size interaction effect.
- ϵ_{ijk} is the random error of the kth observation from the (i, j)th cell.

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

- $H_{1,0}$: The effect of the levels of the interaction term are equal.
- $H_{1,A}$: There is at least one difference between interaction level effects.
- $H_{2,0}$: The effects of the levels of similarity threshold are equal.
- $H_{2,A}$: There is at least one difference between similarity threshold level effects.
- $H_{3,0}$: The effects of the levels of size are equal.
- $H_{3,A}$: There is at least one difference between size level 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 [19]. The next assumption is that both the factors 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 factor 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. In the case that the assumptions are violated beyond the capability to correct, we will be forced to utilize a non-parametric approach. Specifically, we will use a permutation F-test [21], in place of ANOVA. The permutation F-test also has a set of assumptions that must be met prior to use. ... The model for the permutation F-test is the same as that of ANOVA.

In either case, 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 factor. In these experiments the similarity threshold level of 1.0 to be the control (as noted in Section **??**). We have selected to pair Dunnett's [22] multiple comparison procedure if ANOVA is used, and Steel's [23] multiple comparison procedure (a non-parametric technique analogous to Dunnett's) if using the permutation F-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:

- $H_{4,0}$: There is no difference between the mean effects of similarity threshold effects and control effect.
- $H_{4,A}$: There is a difference between at least one similarity threshold level and control.

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

- $H_{4,0}$: There is no difference between the median effects of similarity threshold effects and control effect.
- $H_{4,A}$: There is a difference between at least one similarity threshold level and control.

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

Additionally, we are interested if there is a strict order of the effect on ΔHAL or ΔMCC for the levels of the similarity threshold factor. To evaluate this we have selected to utilize the Jonhckheer's trend test. This is a non-parametric test to determine if there is an *a priori* ordering within independent samples [24]. The hypotheses to be tested are as follows:

- $H_{5,0}$: There is no difference in median effects of the similarity threshold levels.
- $H_{5,A}$: There is a increasing trend in the effects of the similarity threshold levels, with at least one being a strict inequality.

Finally, we will also conduct a sample size analysis to determine the number of repetitions necessary to achieve the power level necessary for the experiments.

5.2.6 Validity Evaluation

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.

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 units there are no social concerns affecting the internal validity. Third, the experimental unites are grammars generated by the combination of existing grammars a process which does not affect the source grammars and which leaves no lasting effect to influence 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 designed 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 preclude 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, 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.

External Validity

The following characteristics of our experiments and data help to ensure the external 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 [26] and TXL [27]) used in practice a threat to external validity still remains. Finally, the nature of the experiment precludes an effect due to the date or time of the application of the treatment.

5.3 Experiment 3: Evaluating Islandization

This section describes the study design for the experiment used to evaluate the multilingual island grammar generation approach from @sec:approach. In developing this study we followed the guidance of Runeson [@http://zotero.org/users/1776655/items/IIF2B7Z7] and Yin [@http://zotero.org/users/1776655/items/KQJDZFIR]. The following subsections detail the research questions, case selection criteria, data collection procedures, and analysis procedures used.

5.3.1 Goals, Hypotheses, Variables

In the spirit of the GQM, we have further refined this goal into a series of directly answerable questions and their underlying rationale. The questions are as follows:

RQ1: How can we automate the combination of languages into a single island grammar?
Rationale:

RQ2: What is the comparative readability of merged grammars?
Rationale:

RQ3: What is the comparative usability of merged grammars?
Rationale:

RQ4: What is the comparative maintainability of merged grammars?
Rationale:

To facilitate answering these questions we have also defined a series of metrics, as follows:

M1: McCabe Cyclomatic Complexity (MCC) a measure of the complexity of a grammar $\mathcal{G}$.
Rationale: A higher complexity represents difficulty in comprehension of the grammar and difficulty in maintaining the grammar. An island grammar is expected to have a lower complexity value than their input grammars.

M2: Halstead Effort (HAL) a measure of effort necessary to maintain grammar $\mathcal{G}$.
Rationale: Similar to MCC, but provides a means by which the MCC is effectively relativised. Island grammars are expected to have a lower complexity and thus lower HAL than their input grammars.

M3: Size of the Set of Input Grammars (Size(G)) the cardinality of the grammars used as input for the island grammar generation.

Rationale: The size of the input grammar set is a key factor in the overall complexity of the generated island grammar and thus should be accounted for.

M4: Size of the Set of Interest Components per grammar (Size(I)) the cardinality of the interest component set used for each grammar acting as input to the island grammar generation algorithms.
Rationale: The size of the interest component set for each grammar used as input for an island grammar is a key factor in the overall complexity of the generated island grammar and thus should be accounted for.

M5: $version(\mathcal{G})$ indicative of the version of the language the grammar $\mathcal{G}$ recognizes. This is a qualitative measure with text data.
Rationale:

The following describes the data to be collected, the data collection process, and how this data is to be stored. First, we describe the data that must be collected. For each grammar under study, we extract the Grammar name, language version, Grammar type, readability score, usability score, and maintainability score for each grammar measured from the Grammar Measure Database, as collected by the tool presented in Section **??**. The data is then accumulated into a table, similar to the example shown in Table 2, with the following specifications:

- Each row of the represents data associated with a specific grammar set.
- The first column is the grammar set identifier.
- The second column represents the method for island grammar generation.
- The third column represents size of the grammar set.
- The fourth column represents the size of the interest set of each grammar in the set.
- The fifth column represents the calculated complexity of the generated island grammar.
- The sixth column represents the average complexity of the grammar set.
- The seventh column represents the calculated effort of the generated island grammar.
- The eighth column represents the average effort of the grammar set.

The data that is collected then breaks down into the following *independent* and *dependent* variable sets. The independent variables representing the properties of concern for the grammar sets are: Technique, Size(G), and Size(I). The levels associated with technique are *Naive* and *MIGG* depending on the assignment of the set to either the Naive or MIGG approaches for island grammar generation. The levels associated with Size(G) are 2, 5, or 10 which represent a sample of the possible number of grammars to be combined. Finally, the levels associated with Size(I) are 2, 5, and 10 which represent a sample of the possible number of interest components that could be selected from a particular grammar. The dependent variables of concern are ΔMCC and ΔHAL. ΔMCC is the difference between the MCC of the generated grammar and the Average MCC of the source grammars. ΔHAL is the difference between the HAL of the generated grammar and the Average HAL of the source grammars.

Based on the research questions posed above and the variables utilized to evaluate the grammars, we have constructed the following sets of hypotheses. The first set is based on evaluating the change in complexity of the grammars when reduced to a combined Island Grammar. *We*

Table 2. Example data table for this study.

Grammar Set	Method	Size(G)	Size(1)	MCC_{IG}	MCC_{avg}	HAL_{IG}	HAL_{avg}
1	Naive	2	2	0.95	0.95	0.95	0.75
1	MIGG	2	5	0.75	0.95	0.95	0.925

expect that when a selection of languages are combined into an Island Grammar with a set of interests imposed, the generated language should be less complex than the average complexity of languages in the input set. The statistical hypotheses used to test are:

$H_{1,0}$: An increase in the number of grammars shows a decrease in average ΔMCC

$H_{2,0}$: An increase in the number of interests shows a decrease in average ΔMCC

$H_{3,0}$: There is a difference in average ΔMCC between the techniques used to generate island grammars, when controlling for $Size(G)$ and $Size(I)$

The second set is based on evaluating the change in effort to maintain the grammars when reduced to a combined Island Grammar. *We expect that when a selection of languages are combined into an Island Grammar with a set of interests imposed, the generated language should less effort to maintain than the average effort of languages in the input set.* The statistical hypotheses used to test this are:

$H_{4,0}$: An increase in the number of grammars shows a decrease in the average ΔHAL

$H_{5,0}$: An increase in the number of interests shows a decrease in the average ΔHAL

$H_{6,0}$: There is a difference in average ΔHAL between the techniques used to generate island grammars, when controlling for $Size(G)$ and $Size(I)$

5.3.2 Design

To gain the most from this experiment, we have elected to use a factorial design. Thus, since we have 2 factors with 3 levels and 1 factor with 2 levels, each replication will need 36 sets of grammars. Furthermore, because the Size(G) variable is based on the number of grammars that are included in the set, we thus need 12 sets at each level of Size(G). Due to this, we need 204 grammars per replication. To gain the most from this experiment we will conduct 3 replications of the experiment, meaning that we require 612 individual grammars.

5.3.3 Experimental Units

This section describes the methods of subject sampling and group allocation. For the purposes of this experiment the experimental subjects are grammars representing software languages. Furthermore, as we are concerned with evaluating MIGG from the perspective of creating static and dynamic analysis tools, we have imposed the following restrictions on the types of grammars we are interested in:

- A Grammar may be considered a viable case, if it contains a minimum of 15 different productions.

  Rationale: the grammar should be more than a simple toy grammar.

- A Grammar may be considered a viable case, if it defines a known and oft used programming language.

  Rationale:

With these restrictions in mind we decided to utilize the Grammar Zoo collection as initial pool from which to select our grammars. Grammar Zoo (at the time of this writing) contains 1029 entries consisting of grammars for a variety of programming languages. With the exception of a few languages most in the collection represent languages currently used in practice by Software Engineers (e.g., Java, Python, Haskell, Dart, etc.).

As noted in Section **??** we will need 612 individual grammars to conduct this experiment. These grammars are selected as follows: Initially, 604 grammars are randomly selected from the initial 1029 grammars in Grammar Zoo using a simple random sampling approach. This set is then randomly partitioned into 3 subsets of 104 grammars representing each of the individual replications of the experiment. Within each replication, the grammars are randomly assigned to treatment groups based on the Size(G) characteristic. For each of the treatment groups, a Size(I) level is assigned. Based on this value, a new Interest Configuration for each of the languages is generated using the Interest Configuration generation algorithm presented in Section **??**. Finally, each treatment group is randomly assigned a generation approach for use in the evaluation process.

5.3.4 Instrumentation

As part of the experimental design measures for metrics M1–M5 must be collected for each generated island grammar. To do this, the tool described in Section **??** provides this capability via implementations of the GrammarMetric depicted in Figure **??**. During program operation, these measures are stored in a database and are retrieve and added to a data table for processing in the analysis phase of the experiment.

5.3.5 Data Collection Procedures

In this study, evaluating the effects of islandization and merging process to develop multilingual island grammars we use the process depicted in Figure 5. This process follows the path identified in Figure 5 by the numbers encircled in orange, as follows:

1. Initially each grammar is loaded by the Grammar Loader and its data is extracted into the meta model depicted in Figure **??**.
2. A lattice of all combinations of grammars is generated to evaluate the groupings of languages into island grammars.
3. The quality metrics from the quality model are measured on the grammar, and
4. stored in the database.
5. The grammar is then transformed using the *islandization*.
6. This requires that the interest configuration for that grammar be loaded.
7. The generated island grammar is then measured using the same metrics from step 2.
8. For the current combination approach, the set of grammars are then merged to generate a multilingual island grammar.

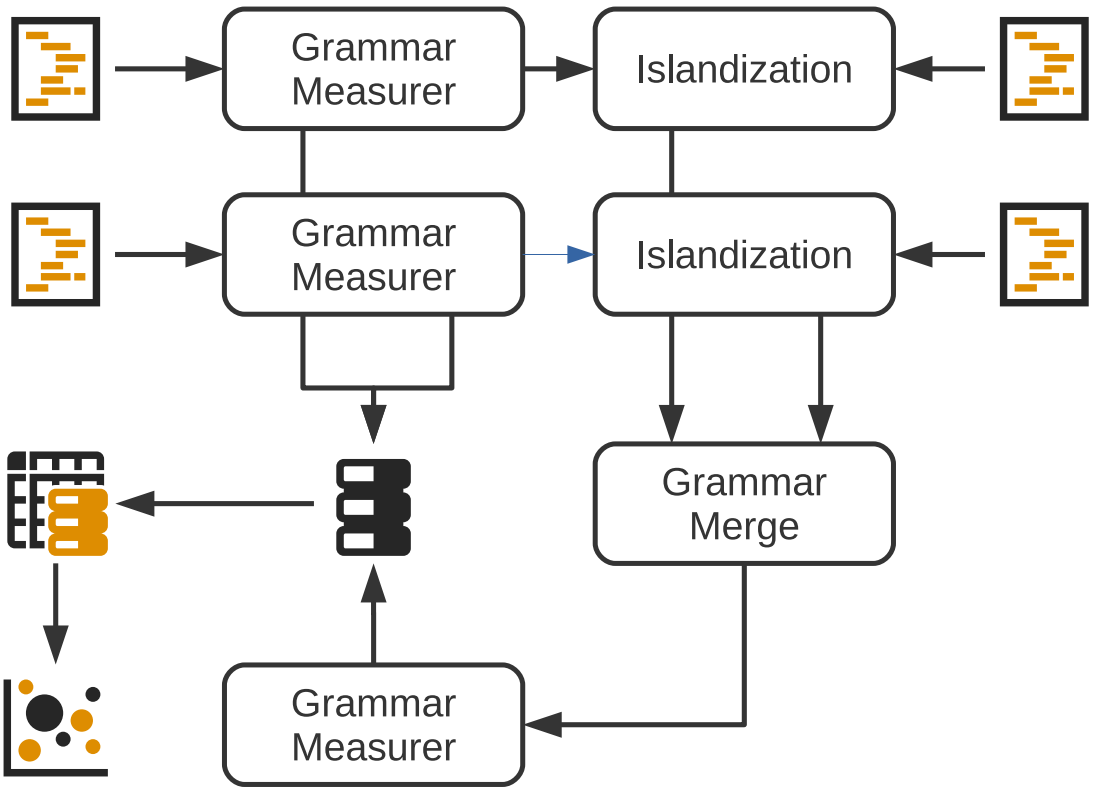


Figure 5. Data collection process for evaluating multilingual grammar islandization process.

9. This grammar is then measured, using the same approach as in step 2, and the data is stored to the database.
10. For each grouping defined in the current partitioning of grammars, the IQR of the individual island grammars is measured, and stored to the database.
11. For the partitioning the average IQR is then calculated and stored to the database. This process restarts at step 1, until there are no partitions left.
12. Once all partitions have been processed, the data for all partitions is extracted into a data table ready for analysis.

5.3.6 Analysis Procedures

The following analysis procedures are used in answering the questions posed in Section **??**. In the first part of this study we have naturally paired samples of data. Given this, we are able to utilize very rudimentary analyses to answer the questions.

For questions RQ3 – RQ5

The results will be in the form of confidence intervals defined at the $\alpha = 97.5\%$ confidence level.

Exploratory Data Analysis

To understand the data collected during experimental operation we used a number of basic statistical analyses. Specifically, to understand the dispersion and typical values of the dependent variables we evaluated the mean and standard deviations of ΔMCC and ΔHAL. Furthermore,

to understand the nature of the relationships between the dependent and independent variables we evaluated the correlations between all variables in the experiment. Furthermore, we can graphically display all of this information in a lattice structure.

Primary Analyses

The primary analyses used in the experiment are designed to evaluate the hypotheses identified in **??**. The experimental design is a factorial design and thus would be typically evaluated using an ANOVA model. But, ANOVA has several assumptions which if not met leads to a higher chance of Type-II error. To evaluate these assumptions we will be utilizing the following tests:

- To evaluate the normality of the data we will use the Shapiro-Wilk test [@http://zotero.org/users/1776655/items/X for normality.
- To evaluate the homogeneity of variance we will use Levene's test for homogeneity of variance [@http://zotero.org/users/1776655/items/QUGJ536S].
- Given the experimental approach we know that the samples are drawn independently and sampled randomly.

In the event that the assumption of the homogeneity of variance is violated but the data is normally distributed, we will apply a weighted variance approach to correct the model. On the other hand, if both assumptions are violated, we will simply return to utilizing the nonparametric Wilcoxson Rank-Sum [X] variant of the test. For both experiments evaluating HAL and MCC we will use the following model.

$$y_{ijkl} = \mu + \tau_{ijk} + \epsilon_{ijkl}$$

Where, y_{ijkl} is the observed value of HAL or MCC for the given treatment combination of ijk and error l. μ is the base line mean value of the MCC or HAL. τ_{ijk} is the treatment value for the i^{th} level of $Method$, the j^{th} level of $Size(G)$ and k^{th} level of $Size(I)$. Finally, ϵ_{ijkl} is the error term representing random error to be controlled for. In the case of both of these experiments we are concerned with determining whether there is any difference due to different treatments, which leads to the following statistical tests:

$H_{1,0}$: There is no difference in mean ΔMCC

$H_{2,0}$: There is no difference in mean ΔHAL

$H_{1,A}$: There is at least one difference in ΔMCC

$H_{2,A}$: There is at least one difference in ΔHAL

In both cases ANOVA and the Rank-Sum based approach simple detect if there is a difference in the means, but does not inform us as to which variable lead to this difference. Thus, in order to detect the difference we will be applying the Tukey-HSV [@http://zotero.org/users/1776655/items/64U6XBSU] analysis to find difference in all means. Additionally, to better understand the effects of individual variables on ΔMCC and ΔHAL we will use the following ...

5.3.7 Validity Evaluation

Conclusion Validity

Internal Validity

Construct Validity

External Validity

5.4 Experiment 4: Evaluating Tolerization

5.4.1 Goals, Hypotheses, Variables

5.4.2 Design

5.4.3 Experimental Units

5.4.4 Data Collection Procedures

5.4.5 Analysis Procedures

5.4.6 Validity Evaluation

5.5 Experiment 5: Evaluating Bounding the Seas

5.5.1 Goals, Hypotheses, Variables

5.5.2 Design

5.5.3 Experimental Units

5.5.4 Data Collection Procedures

5.5.5 Analysis Procedures

5.5.6 Validity Evaluation

5.6 Experiment 6: Evaluating Bridgization

5.6.1 Goals, Hypotheses, Variables

5.6.2 Design

5.6.3 Experimental Units

5.6.4 Data Collection Procedures

5.6.5 Analysis Procedures

5.6.6 Validity Evaluation

6 Results

6.1 Experiment 1: Evaluating Normalization

6.1.1 Descriptive Statistics

6.1.2 Hypothesis Testing

6.2 Experiment 2: Evaluating Merging

6.2.1 Descriptive Statistics

This subsection describes the data collected via a series of descriptive statistics. Table 3 shows the mean values of ΔHAL and ΔMCC for each value of the size category in the respective experiments. Along with this table the dispersion of the both ΔHAL and ΔMCC across size and similarity threshold values is displayed in Figure 6c and 6d, respectively.

As we can see from the Residuals vs. Fitted plot, depicted in Figure 6a, 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.

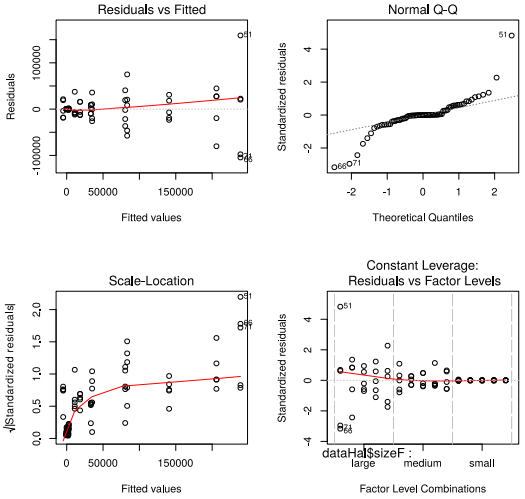
The Normal Q-Q plot, depicted in Figure 6a, indicates that the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test against the Normal distribution. In this test the null-hypothesis (H_0) is that the empirical distribution is Normal. The results show that the A statistic has a value of 6.8257 and an associated p-value of < 2.2e-16 which indicates that we can reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Figure 7a, indicates that their is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0.

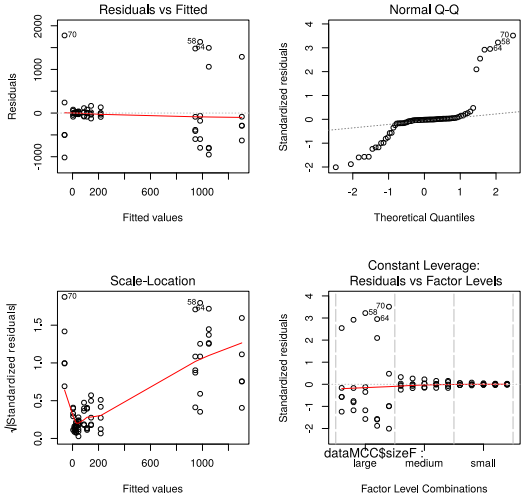
As we can see from the Residuals vs. Fitted plot, depicted in Figure 6b, the homogeneity of variance assumption is not met. To validate this observation we used Levene's Test. In this test, the null-hypotheses (H_0) asserts that the variance is equally distributed. As expected, from our observations 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.

Table 3. Descriptive Statistics.

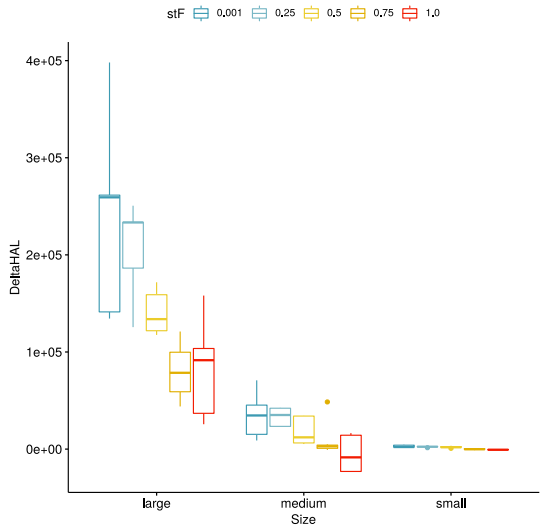
Size Category	**Mean** ΔHAL	**Mean** ΔMCC
Small	1290.2	32.32
Medium	18573	112.7
Large	149854	842.4



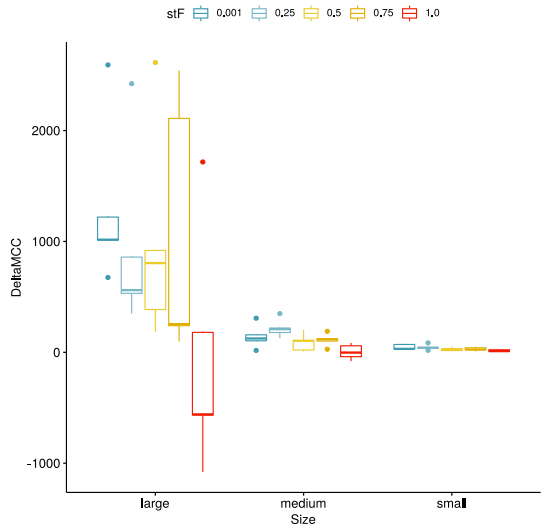
(a) Δ*Hal* experiment Q-Q plots.



(b) Δ*MCC* Experiment Q-Q plots.



(c) Δ*HAL* Boxplots.



(d) Δ*MCC* boxplots.

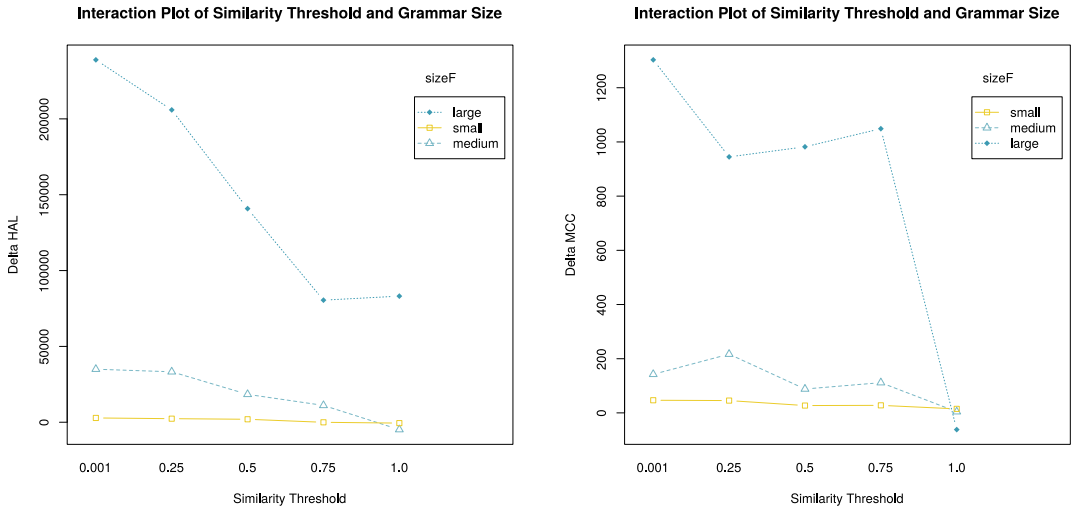
Figure 6. Summary results from the data sets.

The Normal Q-Q plot, depicted in Figure 6b, indicates that the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test against the Normal distribution. In this test the null-hypothesis (H_0) is that the empirical distribution is Normal. The results show the A statistic has a value of 9.9558 and an associated p-value of < 2.2e-16 which indicates that we can reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Figure 7b, indicates that their is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0.

6.2.2 Hypothesis Testing

This subsection details the results of the Δ*HAL* experiment and Δ*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 experiment.



(a) ΔHAL experiment interaction plot. (b) ΔMCC experiment interaction plot.

Figure 7. Interaction plots for both (a) ΔHAL and (b) ΔMCC experiments.

ΔHAL Experiment

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 ΔHAL. 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 factor levels. The results was an F-Value of XXXX and a p-value of XXXX indicating that we could reject $H_{1,0}$. We also note that the interaction indicated in Figure 7a was shown to be significant with an F-Value of XXXX and a p-value of XXXXX.

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 ΔHAL 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 ΔHAL) as the level value is lowered in the similarity threshold. To evaluate this we used Jonchkheere's trend test for ordered differences among classes, which yields a JT statistic value of XXX and a p-value of XXXX, when executed using 10000 permutations. This indicates that we can reject $H_{3,0}$ that there is no *a priori* ordering among levels of similarity threshold on median ΔHAL.

ΔMCC Experiment

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 Complexity for the combined grammars, as measured by ΔMCC. 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 factor levels. The results was an F-Value of XXXX and a p-value of XXXX indicating that we could reject $H_{1,0}$. We also note that the interaction indicated in Figure 7b was shown to be significant with an F-Value of XXXX and a p-value of XXXXX.

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 ΔMCC 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 ΔMCC) as the level value is lowered in the similarity threshold. To evaluate this we used Jonchkheere's trend 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. Indicating that we can reject $H_{3,0}$ that there is no *a priori* ordering among levels of similarity threshold on median ΔMCC.

6.3 Experiment 3: Evaluating Islandization

6.3.1 Descriptive Statistics

6.3.2 Hypothesis Testing

6.4 Experiment 4: Evaluating Tolerization

6.4.1 Descriptive Statistics

6.4.2 Hypothesis Testing

6.5 Experiment 5: Evaluating Bounding the Seas

6.5.1 Descriptive Statistics

6.5.2 Hypothesis Testing

6.6 Experiment 6: Evaluating Bridgization

6.6.1 Descriptive Statistics

6.6.2 Hypothesis Testing

7 Analysis and Interpretation

The results indicate that the 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 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. Because 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 relationship is strong even though it applies only to Antlr grammars.

8 Threats to Validity

8.1 Experiment 1: Evaluating Normalization

8.2 Experiment 2: Evaluating Merging

We focused on threats to the conclusion, construct, internal, and external validity as detailed by Wohlin et al. [25]. We have identified no threats to the conclusion and construct validity of this study. But, 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 formats 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 and Antlr4 and do not include the ability to utilize TXL, SDF or other grammar formats, there is a threat to external validity.

8.3 Experiment 3: Evaluating Islandiation

8.4 Experiment 4: Evaluating Tolerization

8.5 Experiment 5: Evaluating Bounding the Seas

8.6 Experiment 6: Evaluating Bridgization

9 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 automated 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.

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 [28] 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 Section **??**. 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.

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 *International Conference on Software Maintenance*. IEEE, 2003, pp. 179–188.

[11] A. Goloveshkin and S. Mikhalkovich, "Tolerant parsing with a special kind of ńAnyż 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. Kur, 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*. Saarbrücken, 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] J. F. Power and B. A. Malloy, "A metrics suite for grammar-based software," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 16, no. 6, pp. 405–426, Nov. 2004. [Online]. Available: http://doi.wiley.com/10.1002/smr.293

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

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

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

[19] H. Levene, "Robust tests for equality of variances," *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] J. J. Higgins, *An introduction to modern nonparametric statistics*. Pacific Grove, CA: Brooks/Cole, 2004.

[22] 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

[23] 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

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

[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. 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

[27] 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

[28] 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