Abstract—
Index Terms—

I. Introduction

Code analysis tools are an essential part of modern coding. Using them, software engineers find bugs, identify security flaws, increase future maintainability, and comply with business rules and regulations. Modern development workflows use code analysis tools to provide coding assistance, thereby increasing productivity. Software build processes integrate them into their pipelines for quality assurance and other services. Improvements in the technologies that these tools use allow them to solve more problems.

A. Problem Statement

B. Research Objectives

Many code analysis tools transform source code into alternative forms that are easier to process before performing their analysis. One of these forms is an abstract syntax tree. The abstract syntax tree of code preserves the meaning and structure of the code, while removing details that are usually unimportant, such as spaces. The abstract syntax tree produced for code is specific to the programming language, described by a grammar, that the code was written in. The focus of this research is to develop an automated method to combine grammars into an island grammar. This island grammar should describe similar constructs in the original grammars the same so that tools designed to work on this grammar will function for any programming language in the initial grammars. We anticipate that this will allow tool designers to easily support multiple programming languages in a much more maintainable way. This is especially relevant as many codebases are becoming multilingual. There is currently extremely little research on the development of multilingual parsers, which has made static analysis of these multilingual codebases difficult ## Context

C. Organization

II. Background and Related Work

A. Context Free Grammars

A context free grammar can be described as $G = (V, \Sigma, P, S)$ haoxiang_languages_1988. 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. These productions are written as $a \to b$ with $a \in V$ and $b \in (V \cup \Sigma)^*$. When b is empty, the production is denoted by $a \to \varepsilon$. $S \in V$ is the starting symbol. A valid string is represented by a grammar if it can created by repeatedly applying productions moonen_generating_2001. $L(G)$ is used to denote the set of all valid sentences, or language, of grammar G.

B. Abstract Syntax Trees

An abstract syntax tree is an ordered tree describing the symbols and productions of a grammar that are applied for a given sentence in the grammar. An ordered tree is a tree for which the children of each node are numbered. Given a grammar $G = (V, \Sigma, P, S)$, the leaf nodes of a tree must be elements from $\Sigma \cup \{\varepsilon\}$ (ε represents the empty string). The concatenation from left to right of the leaf nodes should equal the sentence for which the parse tree is for. Each internal node must be a member of V. For each internal node v with children $c_1, c_2, ...$, the production $v \to c_1 c_2 ...$ must be a member of P. The root node of the tree must be S. reinhard_wilhelm_compiler_1995

C. Island Grammars

Island grammars are specialized grammars designed to match certain constructs of interest, called islands moonen_generating_2001. They are also designed to blindly match surrounding content that is not of interest as water. They offer several advantages over regular grammars for many applications including faster development time, lower complexity, and better error tolerance. Island grammars have been used for many applications including documentation extraction and processing deursen_building_1999, impact analysis moonen_lightweight_2002, and extracting code embedded in natural language documents bettenburg_what_2008 bacchelli_extracting_2011. Of particular interest to our research is their use for creating multilingual parsers synytskyy_robust_2003, which inspired this research, and the development of tolerant grammars klusener_deriving_2003 goloveshkin_tolerant_2018 kurs_bounded_2015.

D. Tolerant Grammars

Tolerant grammars were initially designed as a way of making island grammars more robust. To do this, klusener_deriving_2003 klusener_deriving_2003 first identified possible problems with island grammars. False positives are content identified by the island grammar as islands even though they are not constructs of interest. False negatives are constructs of interest that the island grammar fails to recognize as islands. To minimize the number of false positives and false negatives, klusener_deriving_2003 came up with the idea to share productions between the island grammar and the full grammar it's related to. The grammars created in this process minimize false negatives and false positives. The shared portions between the island grammar and the full

grammar must start at their start nodes and extend to a certain depth.

E. Maximum Common Subgraph (MCS)

The maximum common subgraph problem is the problem where you try to find maximal isomorphic subgraphs in two labeled graphs. It's commonly used in bioinformatics for finding similarities between chemical structures. This problem is usually broken down into different variations depending on the connectedness of the subgraph, the maximality measure used, and whether approximate or exact solutions are desired. These variations can usually be converted between one another with little work (with exception between approximate and exact variations). For our research, we depend on the connected maximum common induced subgraph variation (connected MCIS). An induced subgraph S of a graph J has an edge between two vertices if and only if J has the same edge between the same two vertices. In connected MCIS, the subgraphs found must be connected and must be induced. The maximality measure used is the number of vertices in the isomorphic subgraphs.

The maximum common subgraph problem is an NP-Complete problem. Because of the difficulty of the problem, several algorithms have been designed for solving the different variations. The algorithms for exact solutions tend to fall into two different kinds of algorithms. The first kind are clique-based algorithms that depend on reducing the MCS problem to the maximal-clique (MC) problem. This is done by calculating the modular product between two graphs. The second kind of algorithms used are backtracking algorithms. These algorithms function by modeling possible solutions with a search tree and then pruning away non-promising solutions. TODO: Citations and list possible algorithms.

For more than two graphs, this problem reduces to the maximal frequent subgraph problem. While these problems might seem quite similar, the algorithms for solving them are quite different. !INCLUDE "contributions.md" # Experimental Design

F. Goals, Hypotheses and Variables
G. Experimental Design
H. Experimental Subjects
I. Experimental Objects
J. Instrumentation
K. Data Collection Procedures
L. Analysis Procedures
M. Evaluation of Validity

III. Execution

A. Sample
B. Preparation
C. Data Collection Performed
D. Validity Procedures

IV. Analysis

A. Descriptive Statistics
B. Data Set Reduction
C. Hypothesis Testing

V. Interpretation

A. Evaluation of Results
B. Limitations of Study
C. Inferences
D. Lessons Learned

VI. Conclusions and Future Work

A. Summary of Findings
B. Relation to Existing Evidence
C. Impact
D. Limitations
E. Future Work

Appendix