Common Abstract Syntax Tree (AST) for Automated Software Analysis of Homework Assignments in Submitty

Elizabeth Dinella, Samuel Breese, Ana Milanova, and Barbara Cutler

Rensselaer Center for Open Source

Static Analysis

Static analysis is a critical tool to ensure software security and dependability. In industry and academia, static analysis can be used to find potential bugs or design problems. On "Submitty", an open source homework server created at RPI, static analysis is used to ensure structural correctness on homework assignments. This functionality is used in RPI's Computer Science 1 (CSCI 1100) to automatically grade small in-lecture exercises. For example, consider the following assignment:

```
for i in range(0,3):
    print i
```

(a) Assignment: rewrite the above code using a while loop

```
i = 0
while i < 3:
    i += 1
```

(b) Correct Student Rewrite

```
#while loop
for i in range(0,3):
    print i
```

(c) Incorrect Student Rewrite - creates a false positive when using tools like "grep"

In this example, all three segments of code would produce the same textual output. Static analysis is required to accurately assess student's progress and avoid false positives that occur in simpler alternatives.

Motivation

In this motivating example, a student in an introductory computer science course is tasked with the following assignment:

`Create a loop in Python or C++ that prints all odd numbers between 1 and 10`

Consider the following correct solutions to this assignment:

```
for i in range(1,10,2):
    print i
```

(d) Python Solution

```
#include <iostream>

using namespace std;

int main(){
    for(int i=1; i<10; i+=2){
        cout << i << endl;
    }
}
```

(e) C++ Solution

Static analysis tools have been requested for other porgramming languages. To implement a simple static analysis tool for while/for loop detection for C++ and Java, the entire framework and the tool must be re-written in each language. Our project provides an alternative by enabling the re-use of static analysis tools across multiple languages.

Introduction

Our project builds a Common Abstract Syntax Tree (Common AST) which captures the structural similarity of the different languages and covers the use cases for static analysis on Submitty. The Common AST is extracted from C++ and Python code and is stored in a standardized XML format. This framework allows a simple process for adding new static analysis tools for different programming languages. The common AST captures relevant structures from Python and C++:

```
ASTNode  = Stmt | Module | Expr
Mod      = Module(ASTNode* body)
Stmt     = FunctionDef(Identifier name, CompoundSmt body)
         | ClassDef(Indentifier name, Bases bases, ASTNode* body)
         | Return(Expr value)
         | Assign(Expr* targets, Expr value)
         | AugAssign(Expr target, Expr value)
         | For(CompoundStmt body)
         | While(Expr test, CompoundStmt body)
         | If(Expr test, CompoundStmt body, Stmt orelse)
         | Bases(Identifier* data)
         | Try(CompoundStmt body, Except except)
         | Except(CompoundStmt body)
         | Raise()
         | Import(Identifier* names)
         | Exec()
         | VariableDecl(Expr* right)
Expr     = BinOp(Expr left, Expr right)
         | UnaryOp(Expr operand)
         | Comparison(Expr left, Expr* comparators)
         | Call(String func, String obj)
```

(f) Common AST

Standardized XML Format

We begin with a syntax directed translation process. In this phase, we translate the original AST into a standardized XML format that represents the Common AST.

- Remove all information that is irrelevant to the Submitty use cases
- output a sequence of tokens in the following format: `<node, level>`
  - `node` is the name of the corresponding structure in the source code. It may represent any of the productions in the common grammar (figure d).
  - `level` represents the depth in the syntax tree.

This format is created using third party libraries that allow users to process the abstract syntax trees of external programs. For C++, Clang/LLVM AST Matchers are used to traverse the AST of an input file. During traversal, when a node relevant to the use cases is found, a token is created and printed to an output file. A similar process occurs for Python using the Python AST library.

Figures e and f show the standardized XML output from the two assignment solutions.

```
<module,1>
<forLoop,2>
<compoundStmt,3>
```

(g) Python Standardized XML corresponding to figure d

```
<module,1>
<importing,2>
<functionDef,2>
<name: main,3>
<compoundStmt, 3>
<forLoop, 4>
<compoundStmt, 5>
```

(h) C++ Standardized XML corresponding to figure e

Recursive Descent Parser

The standardized XML output contains the common AST structure. In this phase, we parse the XML into a Common AST in memory. vspace0.5in To recursively create a Common AST in a depth first top-down order we:

- Take the standardized XML output from the first step as input
- At each level of the recursion, the token in the AST is dispatched to an appropriate processing function based on the node type
  - There is a processing function for each production that follows the grammar. The member variables are initialized recursively. In this way, a node's children are stored in a has-a pointer relationship.

```
FunctionDef* parseFunctionDef(int level){
    FunctionDef* fd = new FunctionDef();
    fd->level = level;
    fd->name = parseIdentifier();
    fd->compoundStmt = parseCompoundStmt();
    return fd;
}
```

(i) FunctionDef Processing Function

Use Cases

Use cases suggested by Submitty instructors:

- Detecting or counting for/while loops and if statements; nested loops and crude complexity analysis
- Detecting language specific keywords such as "goto", "malloc", "auto", etc
- Detecting member function calls from an outside class such as "vector.erase()"
- Counting number of calls to a given function
- Detecting exceptions and ensuring a corresponding handler
- Detecting "code clones" between different languages
- Detecting is-a (inheritance) and has-a (composition) relationships; detecting design patterns

Ongoing Work

- Implementation of use cases
- Large scale testing on RPI Data Structures (CS 1200) and RPI Computer Science 1 (CS 1100) assignments
- Possible extension of framework to include Java

Related Publications

- *Using Static Analysis for Automated Assignment Grading in Introductory Programming Classes*, Breese, Milanova, Cutler, SIGCSE 2017 Static Analysis Poster
- *User Experience and Feedback on the RPI Homework Submission Server*, Wong, Sihsobhon, Lindquist, Peveler, Cutler, Breese, Tran, Jung, and Shaw, SIGCSE 2016 Poster
- *A Flexible Late Day Policy Reduces Stress and Improves Learning* Tyler, Peveler, and Cutler, SIGCSE 2017 Poster
- *Submitty: An Open Source, Highly Configurable Platform for Grading of Programming Assignments*, Peveler, Tyler, Breese, Cutler, and Milanova, SIGCSE 2017 Demo Presentation

Acknowledgments

- Red Hat Software
- Rensselaer Center for Open Source (RCOS)
- `http://submitty.org/`
- `https://github.com/Submitty/Submitty`