UNIVERSITY OF VICTORIA COMPUTER SCIENCE

CSC586A THEORETICAL MODELS FINAL EXAM

Braden Simpson
braden@uvic.ca
V00685500

August 6, 2013

1 QUESTION ONE

1.1 PART A

In this course, the theoretical model I chose to study most was for the project that Jordan and I did, and that model is the Abstract Syntax Tree (AST), more specifically, algorithms to find edit distances between them. The AST is a tree represents the syntactic structure of source code, where each node is a construct (while, return, if etc.), leaves are variables, and branches are blocks of code. See Appendix A for an example.
By using these trees and the algorithms for edit distance, outlined by the fluri et al. [1], we were able to implement an algorithm for finding the changes between two ASTs. I learned from the literature how to perform algorithms on these ASTs to find matching leaves, using levehnstein, n-grams, and other measures.
I also studied procedural generation and markov chains, but they weren't as in depth as the the trees. I found the procedural generation algorithms such as L-Systems and Noise generation to be particularly interesting, especially because of their real-world use in video games and simulations. Because the framework for L-Systems is so easy to learn, anybody can envision how they might be used in nature, which is sometimes hard to do when talking about theoretical models.

1.2 PART B

I used multiple different sources for my work, most of all the paper which helped us learn the algorithms required to perform tree edit distance calculations from fluri et

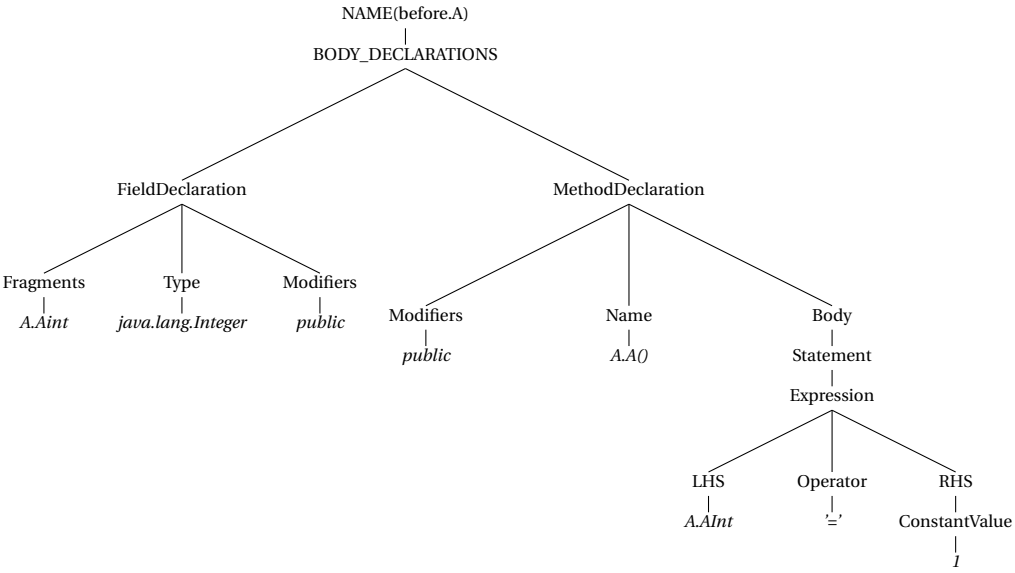
al. [1]. As well I used different research on MSR conferences to learn how to analyze and interpret the data correctly, even as I was there this year, seeing the different ways people inetepret datasets has given me the insight required to critically assess what the data means, what inferences we can get, and more importantly, what we cannot get.

1.3 PART C

For this section I will do an example run of our algorithm for finding the change types in two ASTs for a file (before and after). This will show how the algorithms are used to

```
1  package before;
2
3  public class A {
4      public Integer AInt = null;
5      public A() {
6          AInt = 1;
7      }
8  }
9
```

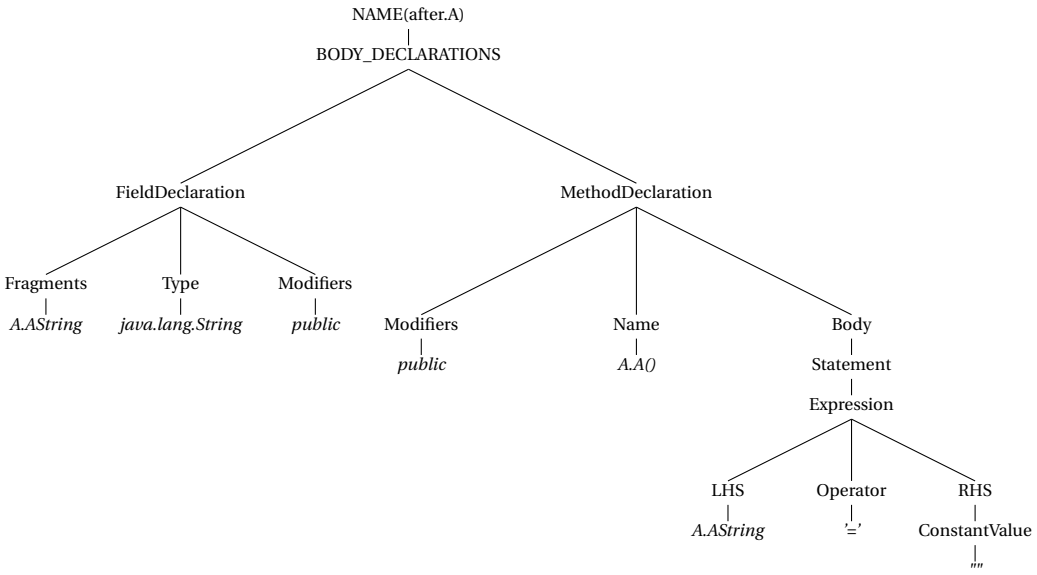
The code for class A before a change.



The AST that corresponds to Figure 1.3

```
1  package after;
2
3  public class A {
4      public String AString = null;
5      public A() {
6          AString = "";
7      }
8  }
9
```

The code for class A after a change.



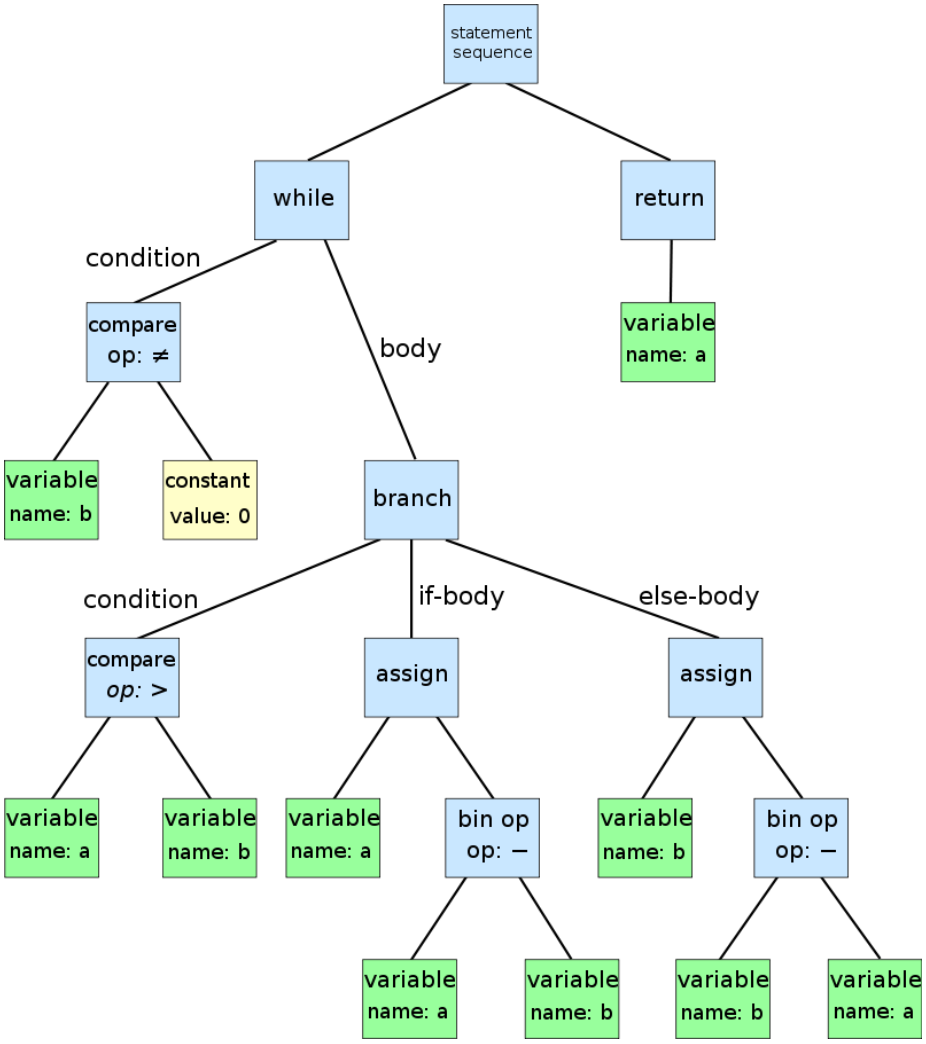
The AST that corresponds to Figure 1.3

Once we have the two ASTs, we use methods in [1] to match the nodes, using Levehnstein and n-gram similarity, to match the nodes. Then we apply our rules, to find an edit script, namely what we need to do to get $Tree_1$ to $Tree_2$.

[0]Note that for Section 1.3 I omitted a few nodes in the ASTs for simplicity, there are many that are generated by the JVM.

2 Question Two

A Abstract Syntax Trees



Data: An abstract syntax tree with matching psuedocode for Euclidean Algorithm. Taken from the wikipedia entry [**?**]

```
while b ≠ 0 do
    if a > b then
        a = a - b
    else
        b = b - a
    end
end
return a
```

REFERENCES

[1] B. fluri, M. Wuersch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Trans. Softw. Eng.*, vol. 33, no. 11, pp. 725–743, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1109/TSE.2007.70731