Sarah Zhou, sarahsz
Anita Leung, leungak

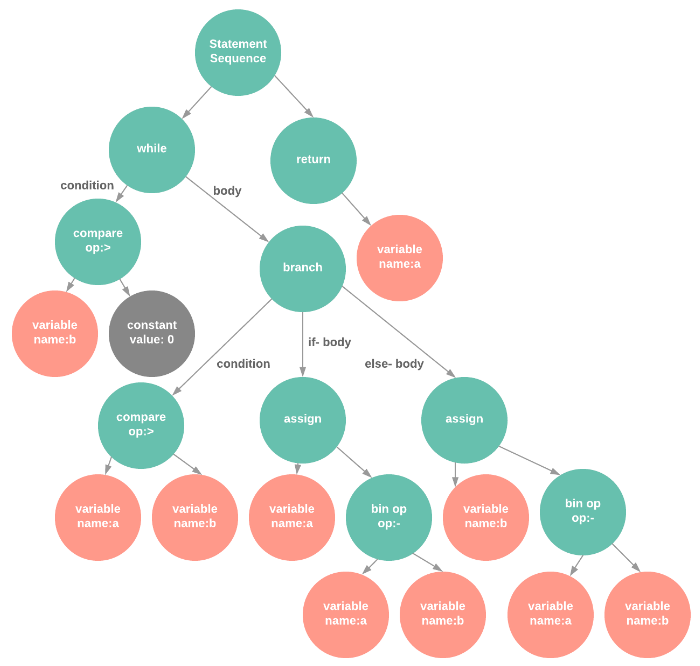
Autofix Merge Conflict

The research question we would like to propose is: *Can we automate version control in a way that detects simple merge conflicts?* One major problem with version control is that it is line-based, which means that if one line of the code you are pushing differs from what another team member is pushing, it will cause a merge conflict. In most cases conflict occurs where there is not one. As an example, when one programmer adds a line break, or adds another tab, or surrounds code by an if statement, due to indentation, the entire block of code that is within that statement would now become a merge conflict. If our system can detect these simple merge conflicts and automatically resolve them, it will save programmers and team members time and effort in the sense that they will no longer have to manually fix a merge conflict that could easily be resolved. A more useful but much more optimistic tool is detecting and solving changes such as manipulating a renamed variable. For example, we have a variable such as foo and baz used in the method bar, but Programmer A wants to push code that adds 1 to baz, and Programmer B wants to change baz to bazzer. Let's say baz is used several times in the code and we want to change every instance of that to bazzer.

For our solution, essentially, we would take in the textual edits and create an abstract sytax tree (AST) of the source code. Each node of the tree would denote a construct in the source code. Through this, we can filter out the code that has been change, and convert that into a AST for both versions. If the versions both produce the same tree, there is no conflict and we can pick either versions of the code. The following example shows how an abstract syntax tree would look and the code for the tree:

An abstract syntax tree is shown on the right for the following code for the Euclidean algorithm:

```
**while** $b \neq 0$
    **if** $a > b$
        $a := a - b$
    **else**
        $b := b - a$
**return** a
```



A related solution is Darcs, a version control system that uses a somewhat different approach to merging code during a conflict. If there are two patches (changes) that conflict, then there is an interesting solution with deciding how to deal with the conflict. It applies an algorithm to determine if it can cancel out the patches. After cancelling the patches (undo the conflict) so that neither has any effect, and inspecting the contents of the conflicting patches, it adds a third patch which indicates that there was a conflict. However, a drawback with this system was is has performance problems when used on large repositories with long histories since its merging algorithm is not efficient.

Our approach is not perfect either. A possible limitation to our approach may be that this solution might not account for the case where both party's code is identical except for the order of lines of code. In this case the solution may need to be extended to further evaluate that part of the code to determine if the difference in ordering affects the output or not. For example, if our solution is to compare whether the structure and node construct of the two ASTs are identical, then we would produce a conflict in the case the trees have different ordering of branches. If our solution only compares the nodes and ignores the structure, then we may give a false positive where no conflict is brought up when both parties code produces different outcome. Another possible limitation is choosing between documentation or comments where one may be better than the other.

We believe the single most serious challenge we would see in developing the product on schedule is implementing the comparison of the ASTs so that we cover most cases that will resolve merge conflicts automatically. We would minimize the risk by brainstorming the cases that should draw a conflict vs should automatically correct itself, and then using the list to drafting an algorithm of how we could compare the ASTs. We could also research resources that may elaborate more on the low level details of this solution.