Conflerge: A Syntax-aware approach to automated merge conflict resolution

John Harrison, Ishan Saksena
CSE 490 E1
Assignment #3
01/11/17

Motivation

In many version control systems, such as Git, the process of merging modified files places a significant burden on developers, requiring them to actively manage, mitigate, and maintain awareness of merge conflicts.

Version control systems primarily treat source code as little more than text, relying on line-by-line diff comparison in order to abort merges that would introduce conflicts. The result of this approach is that developers are quite often required to perform manual conflict resolution, which can be time consuming and error-prone.

The traditional text-based paradigm of reporting potential conflicts is imprecise, often alerting on false-positives that are the result of superficial changes such as formatting, whitespace, or indentation. It may be argued that when a developer seeks to avoid merge conflicts, what they are really concerned with is determining whether they have introduced meaningful semantic or syntactic conflicts into their codebase.

Building from this concern, we seek to address the problem of developing a conflict resolution process for version control systems that would be syntax-aware and able to automatically and reliably resolve a variety of merge conflicts.

By leveraging the structural and known syntactic properties of source code, devising an alternative approach to conflict reporting would have the potential to significantly enhance the merging process in a number of ways:

- It would considerably reduce the burden of managing spurious ‘false-positive’ conflicts that result from trivial formatting details that do not affect the syntactical substance of code
- Reported conflicts would have some level of guarantee that they would impact the interpretation or execution of source code; significant changes in code would be more easily identified
- Less manual merging means a lower potential for introducing new errors

Approach

Our generalized approach to solving the problem of designing an accurate, automated conflict resolution system hinges on conceptually reframing code changes not as simple line-by-line changes to text, but as edit operations on the *syntactic structure* of a block of source code.

By representing them as Abstract Syntax Trees(ASTs), we hope to be able to compare parallel code block changes and then merge them by performing merge operations between the ASTs that are generated.

For every point in a file that would typically alert as a merge conflict in Git, our solution would build ASTs from the conflicting blocks of competing revisions. Once these

comparable ASTs are formed, the system would perform structural comparisons between the two in order to determine whether the changes each AST proposes ought to be considered in conflict with one another. Since ASTs do not represent any whitespace, code punctuation or delimiters, the essential merge comparisons done between these structures would potentially eliminate the prevalence of spurious merge conflict reports outright, as the tree looks at merely the structure and content of the code.

It is important to note, however, that our solution faces distinct limitations that are dependent upon our ability to design the AST merging process such that it makes sensible decisions. Simply comparing trees for equality isn't enough to resolve merge conflicts. We need to be able to analyze changes like nested code and variable replacements, which would require in depth analysis of this particular problem.

There have been attempts to approach code merging and conflicts from this perspective, and its intuitive nature seems to appeal to many developers who are dissatisfied with the inelegant use of line-based conflict analysis typified by VCS' like Git. SemanticMerge is a similar code merging solution that is already on the market and is built to perform code merging from a structural perspective. SemanticMerge relies on a language-specific implementation that is aware of syntactic structures in individual languages, but does not seem to be easily generalized to any language. A key ambition for our project would be to devise a method to perform a structural approach to code merging that is easily generalized to a wide variety of languages, and that would have the potential to be integrated into Git itself and run as part of the git command-line tools.

Scope

With software projects built in a myriad of different programming languages, one of the biggest problems we face is making our tool language agnostic. We would need to modularize our code into modules that analyze different trees and into modules that construct the abstract syntax tree for different languages.

We also wanted to leverage currently existing code to construct the abstract syntax tree as doing so ourselves would be a huge undertaking. The focus of this project is on automating the process of resolving merge conflicts and improving git workflow. There already exist proven methods for parsing ASTs, and we will work to make effective use of these.

Challenges and Risks

The greatest potential risk that would prevent us from delivering a viable project would be in our ability to devise a sensible AST merging and comparison scheme that is both reliable and efficient.

It is our responsibility to inform developers of what is going on behind the scenes and make our conflict resolution easy to understand. This involves adjusting for errors, providing detailed summaries of performed merges, and allowing the user to override any conflict resolutions they may disagree with.

ASTs are computationally expensive to produce. We must find an approach to the tree merge that is efficient and not significantly less performant than the simple line-based conflict checking solutions already in place, it would be difficult to argue for its use if it dramatically underperforms line-by-line diffing.