Summary Review: Integrating Noninterfering Versions of Programs

xiaoyu Zhang

October 31, 2003

Providing an alternative to text-based merging is the major contribution of this paper. This paper applies program dependency graph and program slicing theory to solve three-way merging problem. Although the integration problem examined in the paper is a fairly simplified one, the experimental work focuses on the semantics of the integration which is a big improvement on the previous work that focuses on the integration of text. This paper first discusses criteria for integratability and interference, and some of the problems in textual comparison and merging. Then, it describes the algorithm for integrating noninterfering versions of programs in details. After that, three specific applications of program integration in program development environment are discussed. At last, this paper describes the related work and future work.

Importance: 4.5
Readability: 4.5

The main contribution of this paper is proposing an algorithm, Integrate, which makes use of dependence graphs, to keep track of the changes made to the original version. To divide the program file into individual graph components, this paper also proposed a technique called program slicing to find those statements of a program that determine the values of affected variables. As mentioned in the introduction, this paper borrows (or extends) the idea of dependence graphs used previously for representing programs in vectorizing and parallelizing compilers.

Semantic-base integration provides a more accurate result than text-based merging. Text-based merging on program versions can not guarantee the correctness of the resulting program. Even if there are not syntax errors in the resulting program, there may be some semantic conflicts. If we ignore these conflicts, it is dangerous. So, a more advanced algorithm is needed to detect the semantic conflicts automatically in the program version integration. This paper introduces a semantic-based 3-way integration algorithm to solve the semantic conflict problem. But, the price for this approach is that it is a very heavyweight approach. That is, this approach is very complicated and CPU-consuming.

One of the concerns for this paper is whether or not there is any practical application of the algorithm. This paper makes many simplifying assumptions for this algorithm. For example, expressions in the program contain only scalar variables and constants; statements are either assignment statements or while loops; and also there are two assumptions about the editor used to create variants A and B from copies of Base. We can imagine that the extension of this algorithm for C language will be much more complicated, since functions, macros, pointers, and more types of statement and date types will be introduced. For object-oriented languages, such as C++, is data dependence enough to represent the inheritance relationship? Will other kinds of dependence be introduced to represent the dependences among classes and objects? Is cross-file dependence check supported in this algorithm? It is common that a program file uses external variables that are defined in other file. If all these questions will be considered, the program dependence graph will be very complex. Perhaps the compiler of the programming language can be used to help
generate the dependence graph.

This article spends a lot of time to explain the algorithm, but didn’t spend much time to explain how this algorithm compares to text-based merging algorithms. It would be really great to have some examples of how this algorithm could merge revisions successfully where text-based merging could cause conflicts or mistakes. That would be very important point in the argument to adopt this algorithm, even though this is a very slow and complicated algorithm.

How this algorithm deals with ill-formatted program is another concern. Sometimes, the program being merged has syntax errors, e.g., undefined variable are used, or a """" or "begin" word is missing. Under these circumstances, how can the integration algorithm build the dependence graph? Text-based merging doesn’t have such worry.

Dependency graphs can be used to maintain precondition and postcondition requirements which is helpful in minimizing the amount of problems introduced into the code due to merging. Graph theory can be employed to prove the the correctness of the algorithm. Application of this noninterfering version merge includes optimistic concurrency control, separating changes among revisions, and propagating changes into each version. Overall, this paper is well written and has a solid mathematic and algorithmic analysis.