Summary - Related Work

Jort van Gorkum

December 8, 2021

Chapter 1

An Efficient Algorithm for Type-Safe Structural Diffing

1.1 Abstract

Effectively computing the difference between two versions of a source file has become an indispensable part of software development. The de facto standard tool used by most version control systems is the UNIX diff utility, that compares two files on a line-by-line basis without any regard for the structure of the data stored in these files. This paper presents an alternative datatype generic algorithm for computing the difference between two values of any algebraic datatype. This algorithm maximizes sharing between the source and target trees, while still running in linear time. Finally, this paper demonstrates that by instantiating this algorithm to the Lua abstract syntax tree and mining the commit history of repositories found on GitHub, the resulting patches can often be merged automatically, even when existing technology has failed.

1.2 Introduction

- A consequence of the by line granularity of the UNIX diff is it inability to identify more fine-grained changes in the objects it compares.
- Ideally, however, the objects under comparison should dictate the granularity of change to be considered. This is precisely the goal of structural differencing tools.
- In this paper we present an efficient datatype-generic algorithm to compute the difference between two elements of any mutually recursive family
- The *diff* function computes these differences between two values of type a,
- and *apply* attempts to transform one value according to the information stored in the *Patch* provided to it.
- We expect certain properties of our diff and apply functions.
  - The first being *correctness*: the patch that *diff* x y computes can be used to faithfully reproduces y from x.

$$\forall\, x\, y \,.\, apply (diff\ x\ y)\ x \;\equiv\; Just\, y$$

- The second being *preciseness*:

$$\forall\, x\; y\, .\; apply (diff\; x\; x)\; y \;\equiv\; Just\; y$$

- The last being *computationally efficient*: Both the *diff* and *apply* functions needs to be space and time efficient.

• There have been several attempts at generalizing UNIX diff results to handle arbitrary datatypes, but following the same recipe: enumerate all combinations of insertions, deletions and copies that transform the source into the destination and choose the 'best' one. We argue that this design has two weaknesses when generalized to work over arbitrary types:

  - The non-deterministic nature of the design makes the algorithms inefficient.
  - There exists no canonical 'best' patch and the choice is arbitrary.

• This paper explores a novel direction for differencing algorithms: rather than restricting ourselves to *insertions, deletions,* and *copy operations*, we allow the *arbitrary reordering, duplication,* and *contraction of subtrees.*

1.3 Tree Diffing: A Concrete Example