SEU NOME AQUI

ADAPTAÇÃO E AVALIAÇÃO DE MERGE TEXTUAL BASEADO EM SEPARADORES SINTÁTICOS PARA A LINGUAGEM HASKELL

Belo Jardim
Novembro de 2025

SEU NOME AQUI

ADAPTAÇÃO E AVALIAÇÃO DE MERGE TEXTUAL BASEADO EM SEPARADORES SINTÁTICOS PARA A LINGUAGEM HASKELL

Trabalho apresentado ao Instituto Federal de Pernambuco como requisito parcial para obtenção do título de Bacharel em Engenharia de Software.

Instituto Federal de Pernambuco – IFPE
Campus Belo Jardim
Curso de Bacharelado em Engenharia de Software

Orientador: NOME DO ORIENTADOR

Belo Jardim
Novembro de 2025

Resumo

O desenvolvimento de software moderno depende de sistemas de controle de versão para integrar modificações paralelas. As ferramentas tradicionais de *merge* não estruturado, como o *diff3*, operam linha a linha e frequentemente reportam falsos conflitos quando alterações ocorrem em linhas adjacentes. Abordagens estruturadas resolvem esse problema utilizando a sintaxe da linguagem, mas possuem alto custo de implementação. Recentemente, a técnica CSDiff (*Custom Separators Diff*) foi proposta como uma alternativa híbrida, utilizando separadores sintáticos para segmentar o código antes da comparação. Este trabalho adapta e avalia o CSDiff para a linguagem Haskell, cujas características de sintaxe posicional e operadores funcionais desafiam os algoritmos tradicionais. Através de um estudo empírico com repositórios reais, como *ShellCheck* e *Cabal*, demonstrou-se que a abordagem é capaz de reduzir falsos positivos em cenários de inserção adjacente e, em casos específicos, produzir integrações semanticamente superiores às manuais, embora aumente a granularidade visual dos conflitos remanescentes.

Palavras-chave: Engenharia de Software. Merge de Código. Haskell. CSDiff.

Abstract

Modern software development relies on version control systems to integrate parallel modifications. Traditional unstructured merge tools, such as *diff3*, operate line-by-line and often report false conflicts when changes occur in adjacent lines. Structured approaches address this using language syntax but incur high implementation costs. Recently, the CSDiff (Custom Separators Diff) technique was proposed as a hybrid alternative, using syntactic separators to segment code before comparison. This work adapts and evaluates CSDiff for the Haskell language, whose positional syntax and functional operators challenge traditional algorithms. Through an empirical study with real repositories, such as *ShellCheck* and *Cabal*, it was demonstrated that the approach can reduce false positives in adjacent insertion scenarios and, in specific cases, produce semantically superior integrations compared to manual merges, although it increases the visual granularity of conflicts.

Keywords: Software Engineering. Code Merge. Haskell. CSDiff.

Sumário

1 Introdução

A colaboração em projetos de software exige que múltiplos desenvolvedores alterem a mesma base de código simultaneamente. Para gerenciar essas alterações, Sistemas de Controle de Versão (VCS) utilizam algoritmos de *merge* (integração) para combinar as modificações (MENS, 2002).

A técnica predominante na indústria é o *merge* textual não estruturado, exemplificado pelo algoritmo *diff3* (KHANNA; KUNAL; PIERCE, 2007). Esta abordagem trata o código-fonte como uma sequência simples de linhas de texto, comparando-as sequencialmente. Embora eficiente e agnóstico à linguagem, o *diff3* possui limitações significativas: frequentemente reporta conflitos falsos (falsos positivos) quando as modificações ocorrem na mesma linha ou em linhas consecutivas, mesmo que não haja interferência lógica entre elas.

Para mitigar esses problemas, Clementino, Borba e Cavalcanti (2021) propuseram o *Custom Separators Diff* (CSDiff). Esta técnica propõe um pré-processamento do texto baseado em separadores sintáticos da linguagem (como ";" ou "{" em Java), aumentando a granularidade da comparação sem o custo de construir analisadores sintáticos completos (*parsers*).

1.1 Problema de Pesquisa

Enquanto o CSDiff foi validado para Java, sua eficácia em linguagens funcionais como **Haskell** permanece inexplorada. Haskell apresenta desafios únicos para o *merge* textual, incluindo:

- A regra de *layout* (indentação significativa), que define blocos de código sem marcadores explícitos;
- O uso denso de operadores customizados em linhas únicas (e.g., assinaturas de tipo e *pattern matching*);
- Definições concisas onde múltiplas instruções lógicas ocupam a mesma linha física.

Neste contexto, este trabalho investiga: *A abordagem de merge baseada em separadores sintáticos é eficaz para reduzir conflitos de integração na linguagem Haskell?*

1.2 Objetivos

O objetivo geral é adaptar a técnica CSDiff para Haskell e avaliar sua eficácia comparativa ao *diff3*. Os objetivos específicos são:

1. Identificar e implementar o conjunto de separadores sintáticos relevantes para Haskell (e.g., `::`, `->`, `=`);
2. Desenvolver um protótipo robusto da ferramenta capaz de lidar com peculiaridades de ambiente (como finais de linha CRLF/LF);
3. Realizar um estudo empírico minerando repositórios reais de Haskell;
4. Analisar qualitativamente os cenários de conflito para identificar casos de redução de falsos positivos e potenciais falsos negativos.

2 Fundamentação Teórica

Esta seção apresenta os conceitos fundamentais sobre integração de software e detalha a abordagem CSDiff.

2.1 Merge Textual e Estruturado

O algoritmo *diff3* é o padrão da indústria para integração de três vias (Base, Left, Right). Ele identifica blocos de texto que diferem entre as versões. No entanto, sua granularidade baseada em linhas torna-o suscetível a conflitos de adjacência e ordenação (MENS, 2002).

Ferramentas de *merge* estruturado (APEL et al., 2011) convertem o código em Árvores Sintáticas Abstratas (AST) e realizam a integração nos nós da árvore. Embora eliminem falsos positivos relacionados à formatação, elas são computacionalmente custosas e difíceis de manter para múltiplas linguagens (CAVALCANTI et al., 2019).

2.2 A Abordagem CSDiff

Proposta por Clementino, Borba e Cavalcanti (2021), o CSDiff atua como um intermediário híbrido. A ferramenta não constrói uma árvore sintática, mas utiliza o conhecimento de *tokens* específicos da linguagem para quebrar o texto em unidades menores que uma linha. O processo consiste em três etapas:

1. **Pré-processamento:** Insere quebras de linha e marcadores artificiais ao redor de separadores escolhidos (ex: transforma `a=b` em linhas separadas para `a`, `=` e `b`);
2. **Execução:** Roda o algoritmo *diff3* tradicional sobre os arquivos transformados;
3. **Pós-processamento:** Remove os marcadores e reagrupa o código.

3 Metodologia

Para alcançar os objetivos, foi desenvolvida uma implementação adaptada do CSDiff e uma ferramenta de automação experimental.

3.1 Adaptação para Haskell

A ferramenta `csdiff-hs-merge` foi implementada utilizando *Bash* e *AWK*. Diferente da versão original para Java, a adaptação para Haskell exigiu o tratamento de duas classes de separadores:

- **Separadores Funcionais:** `::` (tipagem), `->` (lambda/case), `=>` (restrição), `<-` (list comprehension) e `@` (as-pattern).
- **Separadores Estruturais:** `=` (atribuição), `|` (guards), `,` (listas) e parênteses.

Foi implementada uma máquina de estados no pós-processamento para garantir a reconstrução correta do código, lidando com problemas de final de linha (CRLF vs LF) comuns em ambientes Windows/WSL. Adicionalmente, implementou-se um filtro para ignorar separadores dentro de comentários de linha (`--`).

3.2 Infraestrutura de Avaliação

Para validar a ferramenta, desenvolveu-se um script em Python (`experiment_runner.py`) inspirado na infraestrutura *MiningFramework* (CLEMENTINO; BORBA; CAVALCANTI, 2021). O script executa:

1. Clona repositórios Haskell (e.g., *ShellCheck*, *Cabal*);
2. Itera sobre *commits* de merge, reconstruindo as versões Base, Left e Right;
3. Executa paralelamente o *diff3* e o *csdiff*;
4. Compara os resultados com o *merge* manual (gabarito) para identificar Falsos Positivos e Negativos.

4 Resultados e Discussão

O experimento foi conduzido minerando os repositórios *ShellCheck* e *Cabal*, totalizando a análise de mais de 3.200 cenários de *merge*.

4.1 Redução de Falsos Positivos

A ferramenta demonstrou eficácia na resolução de conflitos de adjacência. Um caso notável foi identificado no commit `3fa5b7d` do *ShellCheck*. O ramo *Left* modificou uma expressão existente, enquanto o ramo *Right* inseriu uma nova cláusula `case` na linha imediatamente anterior. O *diff3* falhou em separar as mudanças. O CSDiff, utilizando o separador `->`, isolou a inserção como um bloco independente, resolvendo o merge automaticamente.

4.2 Segurança e Divergência Semântica

Durante a análise no repositório *Cabal*, foram identificados casos estatisticamente classificados como Falsos Negativos (onde a ferramenta integrou sem conflitos, mas o resultado diferiu do manual). Uma análise qualitativa revelou que, em alguns casos, a ferramenta superou o humano.

Um exemplo emblemático ocorreu no arquivo `CmdClean.hs` (commit `6f19128`). O ramo *Left* renomeou um comando para `"v2-clean"` e atualizou sua documentação. O ramo *Right* alterou apenas a indentação.

Figura 1 – Comparação entre a integração automática e a manual no arquivo CmdClean.hs

CSDiff (Automático)

```
{ commandName       = "v2-clean"
, commandSynopsis   = "Clean..."
, commandUsage      = \pname ->
    "Usage: " ++ pname ++ " v2-clean [
    FLAGS]\n"
```

Status: Consistente

Manual (Humano)

```
{ commandName       = "v2-clean"
, commandSynopsis   = "Clean..."
, commandUsage      = \pname ->
    "Usage: " ++ pname ++ " new-clean [
    FLAGS]\n"
```

Status: Inconsistente (Bug)

Fonte: Elaborado pelo autor.

A ferramenta realizou a integração perfeita (Figura 1). Em contrapartida, a resolução manual resultou em uma inconsistência: o nome do comando foi atualizado, mas a documentação foi revertida para a versão antiga, provavelmente um erro de atenção humana ao lidar com o conflito de formatação. Isso evidencia a robustez da abordagem proposta.

4.3 Impacto na Legibilidade

Observou-se que a alta granularidade (quebra nos separadores como `=`) pode fragmentar visualmente os conflitos que não são resolvidos automaticamente, dificultando a leitura. A implementação de um modo "Simples" (apenas operadores funcionais) mostrou-se eficaz para mitigar este problema, oferecendo um equilíbrio entre precisão e ergonomia.

5 Conclusão

Este trabalho apresentou a adaptação do algoritmo CSDiff para Haskell. Os resultados confirmam que a segmentação baseada em separadores é capaz de resolver conflitos que ferramentas tradicionais não conseguem, especialmente em casos de inserção adjacente e conflitos mistos de formatação e lógica. Conclui-se que a ferramenta é viável e segura, recomendando-se seu uso com uma configuração flexível de separadores. Como trabalhos futuros, sugere-se a integração da ferramenta como um *driver* nativo do Git.

Referências

APEL, S. et al. Semistructured merge: rethinking merge in revision control systems. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. [S.l.: s.n.], 2011. p. 190–200.

CAVALCANTI, G. et al. The impact of structure on software merging: Semistructured versus structured merge. *IEEE Transactions on Software Engineering*, IEEE, v. 47, n. 10, p. 2221–2244, 2019.

CLEMENTINO, J.; BORBA, P.; CAVALCANTI, G. Textual merge based on language-specific syntactic separators. In: *Brazilian Symposium on Software Engineering (SBES '21)*. Joinville, Brazil: ACM, 2021. p. 1–10.

KHANNA, S.; KUNAL, K.; PIERCE, B. C. A formal investigation of diff3. In: SPRINGER. *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. [S.l.], 2007. p. 485–496.

MENS, T. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, IEEE, v. 28, n. 5, p. 449–462, 2002.