José Mirosmar Santos Silva

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

Belo Jardim
Novembro de 2025

José Mirosmar Santos Silva

# 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: Guilherme Cavalcanti

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 em linguagens densas. Abordagens estruturadas resolvem esse problema utilizando a árvore sintática, mas possuem alto custo de implementação. Como alternativa, a técnica CSDiff propõe o uso de separadores sintáticos para segmentar o código. Este trabalho propõe o **Haskell-SepMerge**, uma adaptação do CSDiff para Haskell, uma linguagem funcional cuja sintaxe posicional (*layout rule*) desafia a reconstrução de código. Através de um estudo empírico analisando 388 cenários de conflito reais, avaliou-se três estratégias de reconstrução. Os resultados demonstraram que a ferramenta aumenta a granularidade dos conflitos (+253%), mas reduz a quantidade de arquivos conflitantes em 10,8%. A comparação revelou que a "Colagem Plana"(Estratégia 1) obteve a maior taxa de sucesso sintático (41,9%), superando tanto a "Colagem Idiomática"(18,6%) quanto a "Colagem Híbrida Protegida"(20,9%). Isso evidencia que, em Haskell, a preservação da estrutura original é mais eficaz do que tentativas de proteção léxica ou formatação artificial.

**Palavras-chave**: Engenharia de Software. Merge de Código. Haskell. Haskell-SepMerge. Layout Rule.

# 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 in dense languages. Structured approaches address this using syntax trees but incur high costs. Alternatively, the CSDiff technique proposes using syntactic separators to segment code. This work proposes **Haskell-SepMerge**, an adaptation of CSDiff for Haskell, a functional language whose positional syntax (layout rule) challenges code reconstruction. Through an empirical study analyzing 388 real conflict scenarios, three reconstruction strategies were evaluated. Results showed that the tool increases conflict granularity (+253%) but reduces the number of conflicting files by 10.8%. The comparison revealed that "Flat Glue" (Strategy 1) achieved the highest syntactic success rate (41.9%), outperforming both "Idiomatic Glue" (18.6%) and "Protected Hybrid Glue" (20.9%). This highlights that in Haskell, preserving the original structure is more effective than lexical protection or artificial formatting attempts.

**Keywords**: Software Engineering. Code Merge. Haskell. Haskell-SepMerge. Layout Rule.

# Sumário

# 1 Introdução

A colaboração em projetos de software exige que múltiplos desenvolvedores alterem a mesma base de código simultaneamente. Para gerenciar essas alterações e os desafios inerentes ao trabalho colaborativo, Sistemas de Controle de Versão (VCS) utilizam algoritmos de *merge* (integração) para combinar as modificações (EDWARDS, 1997; 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. Embora eficiente, o *diff3* possui limitações significativas: frequentemente reporta conflitos falsos (falsos positivos) quando as modificações ocorrem na mesma linha ou em linhas consecutivas. Além do impacto na produtividade, conflitos mal resolvidos ou não detectados podem introduzir defeitos na qualidade do software final (BRINDESCU et al., 2020).

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, aumentando a granularidade da comparação.

## 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, sendo o principal a **Regra de Layout** (*layout rule*). Em Haskell, o escopo de blocos é definido pela indentação. Ferramentas de *merge* textual que alteram espaços em branco podem, inadvertidamente, alterar a semântica do programa ou impedir sua compilação.

Neste contexto, este trabalho investiga: *Como adaptar a abordagem baseada em separadores para lidar com a sensibilidade posicional da linguagem Haskell?*

## 1.2 Questões de Pesquisa

- **QP1:** O Haskell-SepMerge reduz a quantidade total de conflitos reportados?
- **QP2:** A ferramenta reduz a quantidade de arquivos que exigem intervenção manual?
- **QP3:** Qual estratégia de reconstrução produz mais código sintaticamente válido?
- **QP4:** A ferramenta introduz Falsos Negativos (divergências semânticas)?

## 1.3 Objetivos

O objetivo geral deste trabalho é desenvolver e avaliar a ferramenta **Haskell-SepMerge**, uma adaptação da técnica CSDiff para a linguagem Haskell. Os objetivos específicos são:

1. Identificar o conjunto de separadores sintáticos relevantes para Haskell;
2. Implementar e comparar diferentes estratégias de reconstrução de código (Plana, Idiomática e Híbrida) frente à Regra de Layout;
3. Desenvolver uma infraestrutura de testes automatizada para validar a eficácia e segurança da ferramenta;
4. Realizar um estudo empírico com repositórios reais para comparar o desempenho da ferramenta proposta frente ao algoritmo padrão *diff3*.

# 2 Fundamentação Teórica

## 2.1 Integração de Software e suas Abordagens

O processo de integração visa reconciliar mudanças paralelas. As abordagens podem ser classificadas em textuais e estruturadas.

A abordagem textual (*diff3*) é agnóstica à linguagem, rápida e robusta, mas sofre com a falta de contexto semântico. Em contrapartida, pioneiros como Westfechtel (1991) e Buffenbarger (1993) propuseram o *Merge Estruturado*, que opera sobre a estrutura sintática do programa. Trabalhos subsequentes, como o de Apel et al. (2011), evoluíram essa técnica utilizando Árvores Sintáticas Abstratas (AST).

Embora elimine conflitos de formatação, o merge estruturado enfrenta barreiras de adoção. Estudos comparativos indicam que essas ferramentas possuem alto custo de desenvolvimento de *parsers* e complexidade computacional elevada, além da incapacidade de processar arquivos com erros de sintaxe momentâneos (CAVALCANTI et al., 2019).

A técnica CSDiff surge como um meio-termo: utiliza conhecimento sintático leve (apenas separadores) para segmentar o texto, mantendo o motor de comparação textual.

## 2.2 Desafios Sintáticos do Haskell

A escolha da linguagem Haskell justifica-se por suas características desafiadoras.

### 2.2.1 A Regra de Layout

Diferente de linguagens baseadas em chaves (como Java), Haskell utiliza indentação significativa (MARLOW et al., 2010). O aninhamento de blocos é definido pelo alinhamento vertical. Isso cria um paradoxo: para resolver o conflito, a ferramenta precisa inserir quebras de linha, destruindo a estrutura de colunas original que precisa ser restaurada posteriormente.

### 2.2.2 Densidade de Operadores

Haskell favorece código conciso. Assinaturas de tipo e definições frequentemente compartilham a mesma linha, separadas por operadores como `::` ou `=>`. O *merge* padrão enxerga qualquer mudança nessa linha densa como um conflito total.

# 3 Metodologia

## 3.1 A Ferramenta Haskell-SepMerge

A ferramenta foi desenvolvida em *Bash* e *AWK*. Foram implementados dois modos de operação (Simples e Completo) e três estratégias de reconstrução para avaliação.

### 3.1.1 Estratégias de Reconstrução Avaliadas

Para enfrentar o problema da *Layout Rule*, foram comparadas três versões do algoritmo de pós-processamento:

1. **Estratégia 1 (Colagem Plana):** Prioriza a estabilidade. Reconstrói o arquivo tentando "colar"os tokens na mesma linha física anterior sempre que possível, preservando o espaçamento original dos ramos.

2. **Estratégia 2 (Colagem Idiomática):** Tenta embelezar o código. Ao encontrar separadores verticais (como `=>`), insere uma quebra de linha e aplica uma indentação fixa de 4 espaços.

3. **Estratégia 3 (Híbrida/Protegida):** Combina a lógica da Estratégia 1 com uma camada extra de proteção léxica. Tokens compostos (como `||` ou `==`) são substituídos por marcadores únicos (`HSS_...`) antes do merge para evitar corrupção, sendo restaurados ao final.

Figura 1 – Comparativo visual das Estratégias de Reconstrução propostas

**Cenário:** Reconstrução de uma assinatura de função complexa.

**Estratégia 1 (Colagem Plana)**

```
foo :: (Monad m) => a -> m b
foo = ...
```

*Resultado:* Mantém a linha única. Preserva a estrutura original dos ramos, mas pode ficar longo visualmente.

**Estratégia 2 (Idiomática)**

```
foo :: (Monad m)
  => a
  -> m b
foo = ...
```

*Resultado:* Insere quebras em '=>' e '->' com 4 espaços. Visualmente agradável, mas arriscado para o compilador.

**Estratégia 3 (Mecanismo de Proteção Híbrida)**

```
-- Passo 1: Substituição (Pré-Merge)
foo HSS_DOUBLE_COLON (Monad m) HSS_FAT_ARROW a HSS_ARROW m b

-- Passo 2: Merge Textual (Diff3)
...

-- Passo 3: Restauração (Pós-Merge)
foo :: (Monad m) => a -> m b
```

*Mecanismo:* Os tokens são "blindados"para evitar que o algoritmo de merge os fragmente (ex: transformar '=>' em '= >'), garantindo integridade léxica antes da reconstrução plana.

Fonte: Elaborado pelo autor.

## 3.2 Infraestrutura e Processo de Análise

Para validar os resultados, desenvolveu-se um script de automação em Python que opera da seguinte forma para cada cenário minerado:

1. **Execução:** Roda o *diff3* e as três variações do *Haskell-SepMerge* em paralelo.
2. **Métricas:** Contabiliza conflitos e arquivos resolvidos.
3. **Validação (Parse Check):** Se resolvido, invoca `ghc -fno-code` para verificar a validade sintática.
4. **Equivalência:** Compara o resultado automático com a resolução manual, ignorando espaços em branco.

Foram minerados os repositórios *ShellCheck*, *Pandoc*, *Cabal* e *Stack*, totalizando 388 cenários de conflito.

# 4 Resultados e Discussão

Esta seção apresenta os dados consolidados do experimento no modo Completo (*–full*).

## 4.1 Granularidade e Eficácia (QP1 e QP2)

A Tabela 1 apresenta o impacto da ferramenta na quantidade de conflitos.

Tabela 1 – Comparativo de Conflitos e Arquivos Pendentes

| **Métrica** | **Diff3 (Padrão)** | **Haskell-SepMerge** | **Variação** |
|---|---|---|---|
| Total de Conflitos | 2.191 | ≈ 7.700 | **+253%** |
| Arquivos com Conflitos | 387 | 345 | **-10,85%** |

Fonte: Elaborado pelo autor.

Observa-se uma explosão na granularidade (+253%), comum a todas as estratégias. No entanto, essa fragmentação permitiu resolver automaticamente 43 arquivos (redução de 10,85% no esforço manual).

## 4.2 Análise de Validade Sintática (QP3)

A validação com o GHC revelou diferenças críticas. A Tabela 2 mostra o sucesso das estratégias nas 43 resoluções automáticas.

Tabela 2 – Taxa de Sucesso Sintático por Estratégia de Reconstrução

| **Estratégia** | **Arquivos Válidos (ParseOK)** | **Taxa de Sucesso** |
|---|---|---|
| Estratégia 1 (Colagem Plana) | **18** | **41,9%** |
| Estratégia 3 (Híbrida/Protegida) | 9 | 20,9% |
| Estratégia 2 (Colagem Idiomática) | 8 | 18,6% |

Fonte: Elaborado pelo autor.

**A Estratégia 1 (mais simples) obteve o melhor desempenho (41,9%)**, superando as abordagens mais complexas. A Estratégia 2 falhou ao tentar impor indentação artificial (4 espaços) em contextos aninhados. Surpreendentemente, a Estratégia 3 (Híbrida) também teve baixo desempenho. Embora protegesse os tokens, a alteração textual causada pelos marcadores de proteção (`HSS_...`) interferiu no algoritmo de alinhamento do *diff3*, gerando integrações que, embora léxicamente seguras, quebravam o layout posicional.

## 4.3 Divergências e Falsos Negativos (QP4)

A análise de divergência reforça a superioridade da abordagem simples:

- **Estratégia 1:** 17 divergências em relação ao manual.
- **Estratégia 3 (Híbrida):** 32 divergências.

A alta divergência na Estratégia Híbrida indica que a "proteção de tokens"introduz ruído excessivo, levando a ferramenta a produzir código que difere substancialmente da solução humana, aumentando o risco de falsos negativos.

# 5 Conclusão

Este trabalho avaliou o Haskell-SepMerge. Os resultados levam a conclusões importantes sobre o *merge* em linguagens baseadas em indentação:

1. **Potencial de Resolução:** A ferramenta reduz em **10,8%** a quantidade de arquivos conflitantes.

2. **Simplicidade vs. Complexidade:** A Estratégia 1 (Colagem Plana) provou-se superior à Estratégia 2 (Idiomática) e à Estratégia 3 (Híbrida). Em Haskell, preservar o espaçamento original (mesmo que "feio") é mais seguro do que tentar sintetizar indentação ou proteger tokens agressivamente. A proteção excessiva de tokens (Estratégia 3) provou-se contraproducente, gerando mais divergências do que benefícios.

## 5.1 Trabalhos Futuros

Recomenda-se o uso de uma arquitetura de pipeline: utilizar a Estratégia 1 para resolver a lógica do conflito e, imediatamente após, invocar um formatador externo robusto (como *ormolu*) para corrigir a estética, garantindo integridade semântica e visual.

# 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.

BRINDESCU, C. et al. An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering*, Springer, v. 25, n. 1, p. 562–590, 2020.

BUFFENBARGER, J. Syntactic software merging. In: *Software Configuration Management*. [S.l.]: Springer, 1993. p. 153–172.

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.

EDWARDS, W. K. Flexible conflict detection and management in collaborative applications. In: *Proceedings of the 10th annual ACM symposium on User interface software and technology*. [S.l.: s.n.], 1997. p. 139–148.

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.

MARLOW, S. et al. *Haskell 2010 language report*. [S.l.], 2010. Disponível em: <https://www.haskell.org/onlinereport/haskell2010/>.

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.

WESTFECHTEL, B. Structure-oriented merging of revisions of software documents. In: ACM. *Proceedings of the 3rd international workshop on Software configuration management*. [S.l.], 1991. p. 68–79.