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 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 propõe a ferramenta **Haskell-SepMerge**, uma adaptação da técnica 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*, *Cabal* e *Stack*, demonstrou-se que o Haskell-SepMerge é capaz de reduzir falsos positivos em cenários de inserção adjacente e, em casos específicos, produzir integrações semanticamente superiores às manuais, embora aumente significativamente a granularidade visual dos conflitos.

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

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 proposes **Haskell-SepMerge**, an adaptation of the CSDiff technique for the Haskell language, whose positional syntax and functional operators challenge traditional algorithms. Through an empirical study with real repositories, such as *ShellCheck*, *Cabal*, and *Stack*, it was demonstrated that Haskell-SepMerge can reduce false positives in adjacent insertion scenarios and, in specific cases, produce semantically superior integrations compared to manual merges, although it significantly increases the visual granularity of conflicts.
Keywords: Software Engineering. Code Merge. Haskell. Haskell-SepMerge.

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 Questões de Pesquisa

Para responder ao problema proposto, foram definidas as seguintes Questões de Pesquisa (QPs), que guiarão o estudo empírico:

- **QP1:** A ferramenta proposta (Haskell-SepMerge) reduz a quantidade total de conflitos reportados em comparação ao *merge* padrão (Diff3)?
- **QP2:** A ferramenta reduz a quantidade de arquivos ou cenários com conflitos pendentes?
- **QP3:** A ferramenta é capaz de reduzir Falsos Positivos (resolvendo conflitos que o Diff3 não consegue)?
- **QP4:** A ferramenta introduz Falsos Negativos (integrações incorretas ou divergentes do manual)?

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 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 (como *ShellCheck*, *Stack* e *Cabal*) para coletar dados que respondam às QPs;
4. Analisar qualitativamente os cenários de conflito para identificar casos de redução de falsos positivos e potenciais falsos negativos.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais necessários para a compreensão da proposta deste trabalho. Inicialmente, discute-se o problema da integração de software e as limitações dos algoritmos tradicionais. Em seguida, abordam-se as estratégias de *merge* estruturado e a abordagem híbrida CSDiff. Por fim, são detalhadas as características sintáticas da linguagem Haskell que impõem desafios específicos ao processo de integração.

2.1 Sistemas de Controle de Versão e Integração

O desenvolvimento de software em equipe exige o uso de Sistemas de Controle de Versão (VCS), como o Git, para gerenciar a evolução do código-fonte. Uma função crítica desses sistemas é o *merge* (integração), que combina modificações realizadas paralelamente em diferentes ramos (*branches*) de desenvolvimento (MENS, 2002).

O cenário padrão de integração envolve três artefatos: a versão base (ancestral comum), a versão local (*left*) e a versão remota (*right*). O objetivo do algoritmo de *merge* é produzir uma versão integrada que contenha todas as mudanças não conflitantes.

2.1.1 O Algoritmo Diff3 e o Merge Textual

A técnica predominante na indústria é o *merge* não estruturado, ou textual. O algoritmo mais utilizado é o *diff3*, formalizado por Khanna, Kunal e Pierce (2007). Este algoritmo opera tratando o código-fonte como uma sequência de linhas de texto, sem conhecimento da sintaxe da linguagem de programação.

O *diff3* identifica blocos de linhas que foram alterados em relação à base. Se as alterações ocorrem em regiões distintas do arquivo, o algoritmo as aceita automaticamente. No entanto, se as alterações ocorrem na mesma linha ou em linhas adjacentes, o algoritmo reporta um **conflito físico**, exigindo intervenção manual.

Embora eficiente, a abordagem textual sofre de limitações significativas:

- **Falsos Positivos:** Ocorrem quando alterações em linhas consecutivas são marcadas como conflito, mesmo que sejam logicamente independentes (ex: adicionar dois métodos novos em sequência).
- **Falsos Negativos:** Ocorrem quando o algoritmo integra automaticamente alterações que, embora textualmente distantes, interferem na semântica do programa, gerando código com erros de compilação ou lógica.

2.2 Estratégias de Merge Estruturado

Para superar as limitações da abordagem textual, pesquisadores propuseram o *merge* estruturado e semiestruturado (APEL et al., 2011). Essas técnicas não operam sobre linhas, mas sobre a estrutura sintática do programa.

O processo envolve realizar o *parsing* dos arquivos para construir Árvores Sintáticas Abstratas (ASTs). A integração é então realizada comparando os nós dessas árvores. Segundo Cavalcanti et al. (2019), essa abordagem elimina conflitos de formatação e ordenação, pois a árvore abstrai detalhes como espaços em branco e a ordem de declaração de métodos independentes.

Entretanto, o *merge* estruturado apresenta barreiras de adoção:

1. **Custo de Implementação:** Exige um analisador sintático completo e robusto para cada linguagem suportada.

2. **Desempenho:** O processo de construção e comparação de árvores é computacionalmente mais custoso que a comparação de texto.

3. **Fragilidade:** Arquivos com erros de sintaxe não podem ser processados, pois impedem a construção da AST.

2.3 A Abordagem Híbrida: CSDiff

Diante do *trade-off* entre a simplicidade do texto e a precisão da estrutura, Clementino, Borba e Cavalcanti (2021) propuseram o *Custom Separators Diff* (CSDiff). Esta abordagem busca simular o comportamento de ferramentas estruturadas mantendo a base algorítmica textual.

A premissa do CSDiff é que muitos conflitos ocorrem porque o *diff3* usa a quebra de linha (`\n`) como única unidade de separação. Em linguagens modernas, muitas instruções lógicas podem ocupar a mesma linha física. O CSDiff resolve isso através de um pré-processamento que utiliza **separadores sintáticos** específicos da linguagem (como ";", "{", "}" em Java) para aumentar a granularidade do texto.

O fluxo de execução do CSDiff consiste em:

1. **Segmentação:** O código original é reescrito, inserindo quebras de linha artificiais e marcadores únicos ao redor de cada separador sintático identificado. Por exemplo, a expressão `int a=1;` torna-se uma sequência vertical de tokens.

2. **Integração:** O algoritmo *diff3* padrão é executado sobre os arquivos segmentados. Como o código está "expandido", mudanças que antes colidiam na mesma linha passam a ocorrer em linhas distintas, permitindo a resolução automática.

3. **Reconstrução:** Os marcadores são removidos e o código é reagrupado, restaurando (idealmente) o formato original.

2.4 Desafios Sintáticos da Linguagem Haskell

A escolha da linguagem Haskell para este estudo de adaptação justifica-se por suas características sintáticas que diferem radicalmente das linguagens imperativas (como Java e C++) onde o CSDiff foi originalmente validado.

Segundo a definição da linguagem (MARLOW et al., 2010), Haskell é uma linguagem funcional pura que utiliza uma sintaxe posicional forte. Dois aspectos são particularmente desafiadores para ferramentas de *merge*:

2.4.1 A Regra de Layout

Diferente de Java, que usa chaves para delimitar blocos, Haskell utiliza a *layout rule* (indentação significativa). O escopo de funções, condicionais e listas é definido pelo alinhamento vertical dos caracteres. Ferramentas de *merge* textual que não preservam rigorosamente o espaçamento podem introduzir erros semânticos silenciosos ou quebrar a compilação.

2.4.2 Densidade de Operadores

Haskell favorece construções concisas. É comum encontrar definições de tipos, *pattern matching* e lógica de negócios complexa em uma única linha, utilizando operadores customizados como separadores lógicos. Por exemplo, a assinatura de uma função `f :: Int -> Int -> Int` usa o operador `->` como separador. Se dois desenvolvedores alteram partes diferentes dessa assinatura na mesma linha, o *diff3* reportará conflito. A adaptação da técnica para Haskell, portanto, exige a identificação precisa desses operadores funcionais (e.g., `::`, `=>`, `<-`, `@`) para atuar como os novos delimitadores de contexto.

3 Metodologia

Para alcançar os objetivos e responder às questões de pesquisa, foi desenvolvida a ferramenta **Haskell-SepMerge** e uma infraestrutura de automação experimental.

3.1 A Ferramenta Haskell-SepMerge

O Haskell-SepMerge foi implementado utilizando *Bash* e *AWK*. Diferente da versão original do CSDiff 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 (`--`) e uma lógica de proteção de tokens para evitar que separadores compostos (como `=>`) fossem fragmentados incorretamente por separadores simples (como `=`).

3.2 Infraestrutura de Avaliação

Para validar o Haskell-SepMerge, 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*, *Stack*, *Cabal*);
2. Itera sobre *commits* de merge, reconstruindo as versões Base, Left e Right;
3. Filtra merges triviais (onde não houve mudança real nos arquivos);
4. Executa paralelamente o *diff3* e o *Haskell-SepMerge*;
5. Compara os resultados com o *merge* manual (gabarito) utilizando uma comparação "minificada" (ignorando espaços em branco) para focar na equivalência semântica e evitar falsos negativos cosméticos.

4 Resultados e Discussão

O experimento foi conduzido minerando os repositórios *ShellCheck*, *Cabal* e *Stack*, totalizando a análise de 758 cenários de *merge* conflitantes. Os dados coletados foram consolidados para responder às questões de pesquisa (QPs) definidas na introdução.

4.1 QP1: A ferramenta reduz a quantidade total de conflitos?

Para responder à QP1, contabilizou-se o número total de blocos de conflito reportados por cada ferramenta na amostra analisada. A Tabela 1 apresenta o comparativo entre o algoritmo tradicional (*diff3*) e a ferramenta proposta (*Haskell-SepMerge*).

Tabela 1 – Comparativo de Conflitos e Arquivos Conflitantes (Amostra de 758 cenários)

Métrica	**Diff3**	**Haskell-SepMerge**	**Variação**
Total de Conflitos (Blocos)	3.532	13.977	+295,72%
Arquivos com Conflitos	757	696	-8,06%
Cenários Resolvidos (0 conflitos)	0	61	N/A

Fonte: Elaborado pelo autor com base nos dados do experimento.

Os dados consolidados revelam um **aumento expressivo na quantidade total de conflitos** (+295,72%). Este resultado confirma e intensifica a tendência de granularidade observada em estudos similares para Java (+105%) (CLEMENTINO; BORBA; CAVALCANTI, 2021).

A inclusão do repositório *Stack* foi determinante para este índice. Por se tratar de uma ferramenta de *build*, o código contém extensas definições de registros (*Records*) para configuração. Nestes arquivos, o Haskell-SepMerge fragmentou blocos únicos de conflito em dezenas de micro-conflitos de atribuição.

4.2 QP2: A ferramenta reduz a quantidade de cenários com conflitos?

Apesar da explosão no número de blocos individuais, a segunda linha da Tabela 1 demonstra uma **redução de 8,06% na quantidade de arquivos que exigem atenção manual** (de 757 para 696).

Isso significa que, em **61 cenários**, a granularidade extra permitiu que a ferramenta resolvesse **todas** as divergências do arquivo automaticamente. Esses são casos onde o desenvolvedor não precisaria abrir o editor para realizar o *merge*, representando um ganho direto de produtividade.

4.3 Análise de Falsos Positivos e Negativos (QP3 e QP4)

4.3.1 Sucesso na Redução de Falsos Positivos (QP3)

Os 61 cenários classificados como "Resolvidos"representam a redução de Falsos Positivos. Nesses casos, o *diff3* reportou conflito, mas o Haskell-SepMerge integrou o código automaticamente. A análise qualitativa de casos como o commit `3fa5b7d` (*ShellCheck*) confirmou que essa resolução ocorre graças ao isolamento de inserções adjacentes.

4.3.2 Análise de Divergência Semântica (QP4)

A ferramenta gerou alertas de divergência em relação ao *merge* manual. Uma análise qualitativa revelou que, em alguns casos, a ferramenta superou o humano.

Um exemplo emblemático ocorreu no arquivo `CmdClean.hs` (commit `6f19128` do *Cabal*). 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

Haskell-SepMerge (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 com dados do repositório Cabal.

O Haskell-SepMerge 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. Isso evidencia que a ferramenta pode ser mais robusta que a intervenção manual em conflitos mistos de formatação e lógica.

4.4 Impacto na Legibilidade e Layout

Observou-se que a inserção de quebras de linha para isolar separadores pode alterar o *layout* do código. Em assinaturas de tipo complexas (com `=>`), a ferramenta tende a isolar o operador em uma nova linha. Embora o código gerado seja sintaticamente válido para o compilador GHC na maioria dos casos, ele viola as convenções de estilo da linguagem, sugerindo a necessidade de uso conjunto com ferramentas de formatação automática (*code formatters*) no pós-processamento.

5 Conclusão

Este trabalho apresentou o **Haskell-SepMerge**, uma adaptação da técnica de *merge* baseada em separadores sintáticos para a linguagem 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.

Retomando as questões de pesquisa:

- **QP1 (Total de Conflitos):** O Haskell-SepMerge aumenta a quantidade total (+295%) devido à granularidade extrema.
- **QP2 (Cenários Resolvidos):** A ferramenta reduz o número de arquivos com conflitos pendentes (-8%).
- **QP3 (Falsos Positivos):** A eficácia na redução de falsos positivos foi comprovada em 61 cenários da amostra.
- **QP4 (Falsos Negativos):** A ferramenta apresentou divergências em relação ao manual, mas a análise qualitativa indicou que, em casos críticos, o Haskell-SepMerge estava correto e o humano equivocado, embora limitações semânticas (escopo de variáveis) ainda existam.

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 e a incorporação de um formatador automático na etapa de pós-processamento.

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.

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.