

UNIVERSIDADE FEDERAL DA BAHIA  
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

Ygor Mutti Miranda Monteiro do Carmo

**DIFFLESS: UMA FERRAMENTA DE DIFF LÉXICO  
MULTILINGUAGEM**

*Trabalho apresentado ao DEPARTAMENTO DE CIÊNCIA  
DA COMPUTAÇÃO do INSTITUTO DE MATEMÁTICA  
E ESTATÍSTICA da UNIVERSIDADE FEDERAL DA  
BAHIA como requisito parcial para obtenção do grau de  
Bacharel em CIÊNCIA DA COMPUTAÇÃO.*

Orientador: Rodrigo Rocha Gomes e Souza

Salvador  
13 de dezembro de 2019

## Ficha catalográfica.

Carmo, Ygor Mutti Miranda Monteiro do

diffless: uma ferramenta de diff léxico multilinguagem/ Ygor Mutti Miranda Monteiro do Carmo – Salvador, 13 de dezembro de 2019.

48p.: il.

Orientador: Rodrigo Rocha Gomes e Souza.

Monografia (graduação) – UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA E ESTATÍSTICA, 13 de dezembro de 2019.

1. diff. 2. Heaviest Common Subsequence. 3. Compiladores. 4. Evolução de Software..

I. Souza, Rodrigo Rocha Gomes e. II. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA E ESTATÍSTICA. III Título.

NUMERO CDD

## TERMO DE APROVAÇÃO

YGOR MUTTI MIRANDA MONTEIRO DO CARMO

### DIFFLESS: UMA FERRAMENTA DE DIFF LÉXICO MULTILINGUAGEM

Este Trabalho de Graduação foi julgado adequado à obtenção do título de Bacharel em Ciência da Computação e aprovada em sua forma final pelo Departamento de Ciência da Computação da Universidade Federal da Bahia.

Salvador, 13 de dezembro de 2019

---

Prof. Dr. Rodrigo Rocha Gomes e Souza  
Universidade Federal da Bahia

---

Prof. Dr. Tiago de Oliveira Januário  
Universidade Federal da Bahia

---

Prof. Dr. Cláudio Sant'Anna  
Universidade Federal da Bahia

A todas as pessoas que dedicam seu precioso tempo à  
arte de revisar código

## **AGRADECIMENTOS**

Ao meu orientador Rodrigo Rocha, por acreditar no potencial deste projeto e por todo o apoio e paciência. Aos meus amigos que me ajudaram de inúmeras formas, em especial a Paulo Urio, Rafael Guimarães, Robson Peixoto, Gabriel Dahia e meus colegas de trabalho pelas recomendações de leitura, discussões produtivas, valiosos conselhos, incentivo e ajuda com a revisão do texto.

*A lot of times,  
people don't know what they want until you show it to them.*

— STEVE JOBS (BusinessWeek, 1998)

## RESUMO

Desde a criação da primeira ferramenta de diff, o UNIX Diff de 1976, esta área de pesquisa evoluiu bastante. Surgiram algoritmos de diff sensíveis à linguagem dos documentos, capazes de diminuir significativamente o ruído e a lacuna semântica dos diffs, o que facilita a compreensão das mudanças em um software e contribui para a prática de revisão de código moderna. Contudo, esses avanços não chegaram às ferramentas usadas na indústria, que continuam muito similares ao UNIX Diff. O desafio é encontrar a razão disso e meios de aumentar a compreensão das ferramentas sobre os documentos, sem limitar os cenários onde podem ser aplicadas.

Este trabalho apresenta o *diffless*, uma ferramenta de diff léxico criada com o propósito de trazer algoritmos sensíveis à linguagem para o cotidiano dos desenvolvedores de software. O *diffless* utiliza gramáticas no formato tmLanguage para análise léxica, assim é capaz de aproveitar um enorme acervo de gramáticas para suportar novas linguagens, com adição de poucas linhas de código. Sua implementação é facilmente integrável a outros softwares, como os usados para revisão de código através da *web*. Este trabalho também apresenta o *HCSDiff*, um algoritmo de diff com otimizações para diffs léxicos, com suporte a detecção de blocos movidos.

Os resultados obtidos em uma avaliação preliminar sugerem que os diffs obtidos com o *diffless* apresentam menos ruído que os obtidos com o Git Diff. Adicionar suporte a novas linguagens no *diffless* exige menos esforço se comparado a outras ferramentas sensíveis à linguagem.

**Palavras-chave:** diff, análise léxica, Heaviest Common Subsequence, sistemas de controle de versão, revisão de código

## ABSTRACT

Since the creation of the first diff tool, UNIX Diff [citeyear diff], this research area has evolved a long way. Diff algorithms aware of document language have emerged, and they were able to significantly reduce the noise and semantic gap of diffs, which makes it easier to understand change in software, contributing to the modern code review practice. However, these advances did not reach the tools used in the industry, that still are very similar to UNIX Diff. The challenge is to find why and ways to increase understanding of document by the tools without limiting the scenarios where they can be applied.

This work presents *diffless*, a lexical diff tool designed to bring language-sensitive algorithms into the daily lives of software developers. *diffless* uses tmLanguage grammars for lexical analysis, so it can take advantage of a huge collection of grammars to support new languages, with the addition of few lines of code. Its implementation is easily integrable with other software, such as web-based code review tools. This work also introduces *HCSDiff*, a diff algorithm optimized for lexical diffs, with support for block moves detection.

Results from a preliminary evaluation suggest that diffs obtained with *diffless* contain less noise than those obtained with Git Diff. Adding support for new languages in *diffless* requires less effort compared to other language-aware diff tools.

**Keywords:** diff, lexical analysis, Heaviest Common Subsequence, version control systems, code review



# SUMÁRIO

<b>Capítulo 1—Introdução</b>	1
1.1 Deficiências do diff tradicional . . . . .	2
1.1.1 Ruído e lacuna semântica . . . . .	2
1.2 Estado da arte . . . . .	3
1.3 Objetivo e proposta . . . . .	4
1.4 Metodologia de avaliação . . . . .	4
1.5 Organização da monografia . . . . .	5
<b>Capítulo 2—Fundamentação teórica</b>	6
2.1 Entrada . . . . .	6
2.1.1 Documento . . . . .	8
2.1.2 Diretório . . . . .	8
2.2 Análise da entrada . . . . .	9
2.3 Algoritmos de diff . . . . .	10
2.3.1 Correspondência dos átomos . . . . .	10
2.3.1.1 Longest Common Subsequence (LCS) . . . . .	11
2.3.1.2 Heaviest Common Subsequence (HCS) . . . . .	13
2.3.2 Cálculo do diff . . . . .	14
2.3.2.1 Algoritmo tradicional . . . . .	15
2.3.2.2 Algoritmo de detecção de movimentações . . . . .	15
2.4 Saída . . . . .	17
2.4.1 Ruído . . . . .	18
2.4.1.1 Causas . . . . .	19
2.4.2 Lacuna semântica . . . . .	20
2.4.2.1 Causas . . . . .	21
2.5 Ferramentas sensíveis à linguagem . . . . .	21
2.6 Trabalhos relacionados . . . . .	23
2.6.1 TopBlend . . . . .	23
2.6.2 Git Diff . . . . .	24
2.6.3 IDiff . . . . .	25
<b>Capítulo 3—Solução proposta</b>	27
3.1 Entrada . . . . .	28
3.2 Análise da entrada . . . . .	29
3.2.1 Análise léxica . . . . .	29

3.3	Algoritmo de diff . . . . .	32
3.4	Saída . . . . .	33
3.4.1	Formatos de saída . . . . .	35
3.5	Avaliação preliminar . . . . .	36
<b>Capítulo 4—Conclusão</b>		<b>42</b>
4.1	Trabalhos futuros . . . . .	43

## LISTA DE FIGURAS

2.1	Fluxograma básico de uma ferramenta de diff . . . . .	7
2.2	Documentos textuais “a.txt” e “b.txt” . . . . .	8
2.3	Cálculo do comprimento da LCS . . . . .	12
2.4	Cálculo da LCS . . . . .	13
2.5	Diff entre “a.txt” e “b.txt” obtido com o GNU diff . . . . .	16
2.6	Diff entre “a.txt” e “b.txt” obtido com o Meld . . . . .	16
2.7	Diff com movimentações obtido com o Git Diff . . . . .	17
2.8	Diff com diferenças de formatação . . . . .	19
2.9	Exemplos de diffs com e sem dessincronização . . . . .	22
3.1	Interface de linha de comando do <i>diffless</i> . . . . .	28
3.2	Diagrama de classes do modelo de entrada do <i>diffless</i> . . . . .	28
3.3	Diagrama de classes das opções do algoritmo <i>HCSDiff</i> . . . . .	29
3.4	Diagrama de classes da integração com vscode-textmate . . . . .	30
3.5	Diagrama de classes do modelo de análise léxica do <i>diffless</i> . . . . .	31
3.6	Diagrama de classes da implementação de <i>HCSDiff</i> . . . . .	34
3.7	Diagrama de classes do modelo de saída do <i>diffless</i> . . . . .	35
3.8	Exemplo de exibição de movimentação na interface gráfica . . . . .	36
3.9	Exemplo: Formatação e adição de caractere - Padrão . . . . .	37
3.10	Exemplo: Formatação e adição de caractere - Limpo . . . . .	38
3.11	Exemplo: Átomo comprido - Padrão . . . . .	39
3.12	Exemplo: Átomo comprido - Limpo . . . . .	40
3.13	Exemplo: Átomo comprido - Versão antiga da análise léxica . . . . .	41

## LISTA DE SIGLAS

<b>AST</b>	Abstract Syntax Tree	
<b>JSON</b>	JavaScript Object Notation .....	32
<b>LCS</b>	Longest Common Subsequence.....	ix
<b>LSP</b>	Language Server Protocol .....	6
<b>HCS</b>	Heaviest Common Subsequence .....	13
<b>HTML</b>	HyperText Markup Language .....	23
<b>POSIX</b>	Portable Operating System Interface .....	1
<b>SCM</b>	Software Configuration Management .....	8
<b>SES</b>	Shortest Edit Script	
<b>URI</b>	Uniform Resource Identifier .....	7
<b>URL</b>	Uniform Resource Locator.....	7
<b>VCS</b>	Version Control System	
<b>VSCo</b>	Visual Studio Code .....	29
<b>CLI</b>	Command Line Interface .....	27

## Capítulo

# 1

*Contextualiza e expõe a motivação para o desenvolvimento deste trabalho, seus objetivos, um resumo da solução proposta, a metodologia de avaliação dos resultados e a organização dos capítulos.*

## INTRODUÇÃO

Os (VCSs, do inglês *Version Control Systems*) desempenham um papel central na Gerência de Configuração de Software (SCM, do inglês *Software Configuration Management*), uma das tecnologias de Engenharia de Software mais bem-sucedidas na indústria, utilizada para controlar a evolução do software (DART, 1991; ESTUBLIER, 2000). Entre as principais funcionalidades de um VCS estão servir de repositório para o histórico de versões de um conjunto de documentos, informar o que mudou entre as versões e facilitar a criação de novas versões de forma colaborativa (KOC; TANSEL, 2011).

O componente do VCS que permite que o usuário visualize o que mudou entre versões é a ferramenta de diff, isto é, um utilitário que compara documentos e retrata as diferenças entre eles como uma sequência de operações de edição legível por humanos, chamada de diff (HUNT; MCILROY, 1976; HUNT; VO; TICHY, 1998). Os algoritmos de *merge* dos VCSs, que permitem integrar edições desenvolvidas em paralelo em uma versão em comum, se baseiam em diffs para realizar essa tarefa (KHANNA; KUNAL; PIERCE, 2007). A revisão de código moderna é uma técnica de SCM que traz benefícios significativos para a qualidade do software, e basicamente consiste da revisão por pares dos diffs das mudanças propostas, com apoio de ferramentas especializadas integradas aos **giVCS!s** (**giVCS!s**) (BACCHELLI; BIRD, 2013; MCINTOSH et al., 2014). Além disso, ferramentas de diff também podem ser utilizadas de forma independente dos VCSs, inclusive para comparar documentos de origens distintas.

O UNIX Diff (1976) foi a primeira ferramenta de diff de propósito geral eficaz para comparação de documentos de texto. Ela se popularizou antes mesmo que surgissem os primeiros VCSs, que a utilizavam para exibir as diferenças e para armazenar o histórico de versões de forma compacta (HUNT; VO; TICHY, 1998), e alcançou tamanha aceitação e relevância que sua Interface de Linha de Comando (CLI, do inglês *Command Line Interface*) foi incluída na influente norma Portable Operating System Interface (POSIX) como a especificação do comando `diff`, obrigatório em um sistema operacional compatível com POSIX (IEEE; The Open Group, 2018).

Além da sua importância histórica, o UNIX Diff estabeleceu o paradigma seguido pela maioria das ferramentas de diff, chamado aqui em diante de diff tradicional. Estas ferramentas enxergam os documentos como sequências de átomos, geralmente linhas (i.e. usam granularidade de linha), independente da linguagem em que foram escritos. Elas buscam átomos correspondentes entre os documentos através de algoritmos para o problema da Maior Subsequência Comum (LCS, do inglês *Longest Common Subsequence*) ou similares. Os átomos sem correspondentes são considerados diferenças e apresentados como adições, remoções ou substituições (HUNT; MCILROY, 1976).

## 1.1 DEFICIÊNCIAS DO DIFF TRADICIONAL

O limitado poder de processamento e espaço em memória do hardware da época da criação do UNIX Diff restringiram as possibilidades da ferramenta (HUNT; MCILROY, 1976). Desde então houve um crescimento do poder computacional e barateamento do hardware em ritmo exponencial (Experts Exchange, 2018). A evolução do hardware e avanços no estado da arte viabilizaram o desenvolvimento de ferramentas de diff mais complexas, citadas ao longo desta monografia, que apresentaram resultados melhores que os das ferramentas tradicionais sob diversos aspectos.

Entretanto, ferramentas que produzem diffs similares aos do UNIX Diff continuam populares. Dois grandes exemplos são o Git Diff, ferramenta de diff integrada ao Git (TORVALDS; HAMANO et al., 2018), que é o VCS mais popular atualmente (RhodeCode, 2016); e o GNU Diff (MACKENZIE; EGGERT; STALLMAN, 2017), implementação do comando diff compatível com POSIX, inclusa em diversos sistemas operacionais<sup>1,2</sup> baseados em UNIX, usados por mais da metade dos desenvolvedores de software (Stack Overflow, 2019). Ferramentas populares de *merge* e revisão de código, como diff3 (KHANNA; KUNAL; PIERCE, 2007), Git Merge, Gerrit<sup>3</sup> e GitHub Pull Requests<sup>4</sup> também são baseadas em diffs tradicionais.

Não depender da linguagem dos documentos e se basear em conceitos genéricos (documentos, linhas, etc) é ao mesmo tempo um ponto forte dessas ferramentas, pois maximiza sua aplicabilidade, e um ponto fraco, pois o conhecimento que elas são capazes obter sobre os documentos é raso e daí surgem muitas de suas deficiências. As possibilidades de melhoria usando abordagens independentes de linguagem aparentemente estão se esgotando e a evolução das ferramentas de SCM disponíveis no mercado durante muitos anos se limitou a pequenos refinamentos das técnicas existentes (ESTUBLIER, 2000).

### 1.1.1 Ruído e lacuna semântica

Este trabalho define os conceitos de ruído e lacuna semântica no diff, que abrangem as deficiências que motivaram o desenvolvimento de grande parte das melhorias nas ferramentas de diff desde o UNIX Diff. Ruído é uma diferença irrelevante apresentada no diff sem distinção em relação às diferenças que são relevantes para o usuário, restando ao

---

<sup>1</sup><<https://manpages.ubuntu.com/manpages/cosmic/pt/man1/diff.1.html>>

<sup>2</sup><<https://opensource.apple.com/source/gnudiff/gnudiff-21/diffutils/>>

<sup>3</sup><<https://www.gerritcodereview.com/>>

<sup>4</sup><<https://github.com/features/code-review/>>

usuário analisá-la sem necessidade. Lacuna semântica é a distância que há entre a representação das diferenças no diff e a informação que o usuário deseja extrair delas. Quanto maior a lacuna, maior o esforço cognitivo necessário para interpretar o diff. Neste trabalho é feita uma discussão aprofundada sobre suas causas e consequências, incluindo exemplos.

O ruído e a lacuna semântica das ferramentas tradicionais são fatores que, ao dificultarem a compreensão dos diffs, dificultam a revisão de código moderna (BACCHELLI; BIRD, 2013; SEEMANN, 2015). Além disso, também aumentam o risco de conflitos de *merge*, ou seja, edições feitas em paralelo que afetam os mesmos documentos de formas que a ferramenta de *merge* do VCS não é capaz de conciliar. Edições conflitantes precisam ser combinadas manualmente pelo usuário do VCS, uma tarefa complexa e sujeita a erros devido às interconexões sintáticas e semânticas entre os trechos envolvidos nos conflitos e os demais (MENS, 2002).

## 1.2 ESTADO DA ARTE

Uma abordagem bastante utilizada para resolver as deficiências do diff tradicional é o uso de algoritmos sensíveis à linguagem dos documentos que compreendem os documentos a nível sintático ou semântico (YANG, 1991; HORWITZ, 1990). Os primeiros esforços para utilizá-los em ferramentas de SCM datam do início dos anos 80. Infelizmente, na maioria dos casos os benefícios não compensaram os custos em termos de complexidade, generalidade e eficiência, justificando a baixa aceitação e descontinuação do desenvolvimento de várias dessas ferramentas ainda enquanto protótipos de pesquisa. Encontrar formas de aumentar o conhecimento das ferramentas sobre os documentos sem aumentar esses custos de forma proibitiva é um dos grandes desafios em SCM (ESTUBLIER, 2000).

Nos últimos anos é notável o surgimento de ferramentas de diff sensíveis à linguagem desenvolvidas fora do meio acadêmico, que utilizam algoritmos mais simples do que os que representam o estado da arte e têm um foco maior em usabilidade. Alguns exemplos são o SemanticMerge (Código Software, 2018), SmartDifferencer<sup>5</sup>, JSON Diff (GROSSBART; SAVIO; MALINA, 2018) e Pretty Diff (CHENEY et al., 2018). As estatísticas de tráfego dos websites dessas ferramentas<sup>6</sup> e as solicitações de ferramentas de diff sensíveis a linguagem em comunidades de desenvolvimento de software, como o StackOverflow<sup>7</sup>, indicam que existe demanda da indústria por ferramentas desse tipo.

A análise empírica das ferramentas sensíveis à linguagem existentes revela diversos inconvenientes em relação às ferramentas tradicionais, possíveis razões destas continuarem sendo mais populares. De modo geral: as ferramentas desenvolvidas pela academia não se encontram facilmente disponíveis para download; as ferramentas de código fechado possuem custos de licenciamento; as ferramentas mais avançadas suportam apenas um subconjunto de uma linguagem, ou quando suportam uma linguagem completamente não dão suporte a diversas linguagens populares; estender essas ferramentas para adicionar

---

<sup>5</sup><<https://www.semanticdesigns.com/Products/SmartDifferencer/>>

<sup>6</sup>Estatísticas obtidas através da ferramenta de análise de tráfego de websites Alexa, consultada em abril de 2019

<sup>7</sup><<https://stackoverflow.com>>

suporte a novas linguagens ou integrá-las a outras ferramentas — como as de revisão de código, VCSs e ambientes de desenvolvimento — é uma tarefa que exige esforços consideráveis, entre outros inconvenientes.

### 1.3 OBJETIVO E PROPOSTA

O objetivo deste projeto é disponibilizar uma ferramenta de diff capaz de trazer diffs sensíveis à linguagem para o dia-a-dia dos desenvolvedores de software, minimizando o prejuízo causado pelas deficiências do diff tradicional à Engenharia de Software. É uma tentativa de compilar boas ideias da academia e da indústria em uma ferramenta, sem cometer os mesmos erros, ou seja, sem limitar demasiadamente a aplicabilidade da ferramenta.

Esta monografia introduz o algoritmo de diff *HCSDiff*, que utiliza uma função de peso para guiar a busca por átomos correspondentes entre os documentos e, além das operações de edição usuais, é capaz de detectar movimentações. Através do ajuste da função de peso é possível obter diffs mais compactos, por exemplo. Em cenários onde ocorrem refatorações, a detecção de movimentações ajuda a diminuir a lacuna semântica (SILVA et al., 2014; FOWLER, 2018). Além disso, o *HCSDiff* foi implementado de modo a suportar qualquer tipo de átomo, desde que seja possível representar os documentos como sequências de átomos desse tipo.

A partir desse algoritmo foi desenvolvido o *diffless*, uma ferramenta de diff sensível à linguagem, gratuita e de código aberto. Ela oferece uma CLI que permite uma fácil integração com VCSs e ambientes de desenvolvimento. Sua arquitetura permite o reuso dos seus componentes como um framework para o desenvolvimento de novas ferramentas e algoritmos de diff, podendo também ser integrada a ferramentas de revisão de código online como uma biblioteca.

Diferente da maioria das ferramentas de diff sensíveis à linguagem, o *diffless* se baseia na análise léxica dos documentos ao invés de análise sintática ou semântica. Dessa forma é possível reaproveitar um amplo leque de gramáticas léxicas usadas por editores de código para oferecer suporte a um grande número de linguagens, o que também torna fácil e atraente implementar suporte a novas linguagens no *diffless*. Naturalmente, um diff léxico apresenta mais ruído e lacunas semânticas do que um diff a nível sintático ou semântico. É um meio-termo entre aumentar a compreensão da ferramenta sobre os documentos e dar suporte ao maior número possível de linguagens.

Como o algoritmo *HCSDiff* compreende os documentos como sequências de átomos, nos casos em que não há suporte para a linguagem desejada o *diffless* permite o uso de granularidades de linha e caractere, se comportando de maneira similar a uma ferramenta de diff tradicional, mas ainda assim conta com os benefícios trazidos pelo uso de pesos no *HCSDiff* e detecção de movimentações.

### 1.4 METODOLOGIA DE AVALIAÇÃO

Para validar as abstrações utilizadas na arquitetura do *diffless*, os principais pontos de extensão foram exercitados pelo menos duas vezes, e para garantir a qualidade do código



foram implementados testes unitários para os algoritmos e testes de regressão para diffs obtidos a partir de um conjunto de documentos de exemplo escolhidos para representar os prós e contras da solução proposta. Em uma avaliação preliminar, os pares de documentos comparados nos testes também foram comparados usando Git Diff e os resultados das duas ferramentas foram confrontados.

Os resultados obtidos sugerem que, graças ao uso da análise léxica, o *diffless* é inteligente o suficiente para detectar diversas mudanças que não afetam a semântica do documento — como a maioria das mudanças de formatação — e delimitar melhor as edições, apresentando menos ruído e lacunas semânticas que as ferramentas tradicionais. Ao mesmo tempo, observou-se que são necessária menos linhas de código para adicionar suporte a novas linguagens no *diffless* do que em outras ferramentas sensíveis a linguagem.

## 1.5 ORGANIZAÇÃO DA MONOGRAFIA

Este capítulo introduz o trabalho, mostrando a importância das ferramentas de diff e o impacto das deficiências das ferramentas de diff disponíveis, que motivaram o desenvolvimento do *diffless*. O Capítulo 2 apresenta um compilado de conceitos que serviram de fundamentação teórica para o *diffless*, além de uma breve comparação do *diffless* com trabalhos relacionados. O Capítulo 3 apresenta em detalhes a arquitetura do *diffless*, o algoritmo *HCSDiff* e uma breve discussão sobre os resultados alcançados. Por fim, o Capítulo 4 conclui a monografia elencando as vantagens e desvantagens do *diffless* em relação as demais ferramentas e os planos para o futuro do projeto.

*Apresenta os conceitos necessários para compreensão desta monografia, além de trazer informações sobre trabalhos relacionados.*

## FUNDAMENTAÇÃO TEÓRICA

Uma ferramenta de diff é um software utilitário que recebe como entrada um par ordenado de documentos<sup>1,2</sup> ou de diretórios e produz como saída uma sequência de edições, chamada de diff, que ao ser aplicada sobre o primeiro item do par o transforma num equivalente ao segundo item do par (IEEE; The Open Group, 2018).

Em geral, o funcionamento de uma ferramenta de diff obedece ao fluxograma da figura 2.1: inicialmente ela analisa os itens da entrada transformando-os em representações intermediárias envolvendo os elementos lógicos reconhecidos nos documentos, denominados átomos. Em seguida, um algoritmo de correspondência mapeia quais átomos do primeiro item da entrada correspondem a quais átomos do segundo item da entrada. Por fim, é feito o cálculo do diff, etapa onde os átomos sem correspondentes são agrupados em blocos e mapeados para edições, que serão apresentadas como saída da ferramenta. A combinação de algoritmos que realizam a correspondência dos átomos e o cálculo do diff da ferramenta é chamada de algoritmo de diff.

### 2.1 ENTRADA

Os itens do par ordenado de entrada da ferramenta de diff costumam ser referenciados individualmente como esquerda e direita, respectivamente. A maioria das ferramentas

---

<sup>1</sup>As referências bibliográficas adotam terminologias incompletas, limitantes e inconsistentes entre si. Para garantir a consistência do texto, facilitar a comparação entre algoritmos e evidenciar como podem ser aplicados no contexto deste trabalho alguns termos presentes nos trabalhos originais foram substituídos por termos que representam melhor os conceitos na opinião do autor, e novos conceitos foram introduzidos, resultando em uma terminologia própria inspirada por diversos trabalhos da área e no Language Server Protocol (LSP) (Microsoft, 2018a) — protocolo de comunicação em crescente adoção por ferramentas de edição e análise de código-fonte (Sourcegraph et al., 2018).

<sup>2</sup>Apesar de *arquivo* ser o termo mais usado nesse contexto, *documento* é um termo mais adequado pois um *arquivo* é apenas uma das possíveis formas de armazenamento de documentos suportadas por ferramentas de diff (IEEE; The Open Group, 2018).

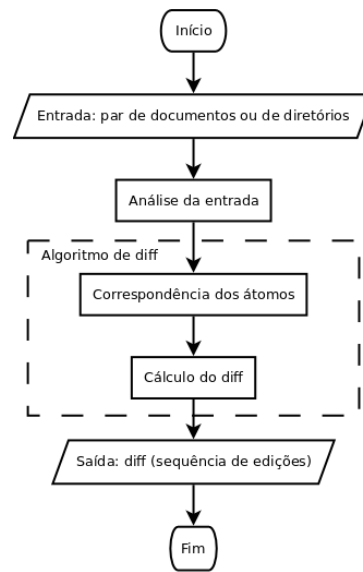


Figura 2.1: Fluxograma básico de uma ferramenta de diff

também aceitam opções de configuração na sua entrada, importantes para que possam atender os mais diversos cenários de uso e para o ajuste fino dos algoritmos, que pode trazer melhorias significativas na qualidade dos diffs (MACKENZIE; EGGERT; STALLMAN, 2017).

Por questões de praticidade, identificadores são frequentemente usados na entrada de ferramentas de diff ao invés de representações dos documentos e diretórios propriamente ditos. Naturalmente, a ferramenta precisa estar preparada para acessá-los através dos seus identificadores. Alguns tipos de identificadores são:

- caminhos de sistemas de arquivos (IEEE; The Open Group, 2018);
- referências a versões de diretórios e documentos armazenadas em um Sistema de Controle de Versão (VCS, do inglês *Version Control System*) (TORVALDS; HAMANO et al., 2018);
- Uniform Resource Locators (URLs) e demais tipos de Uniform Resource Identifiers (URIs) (BERNERS-LEE; FIELDING; MASINTER, 2004; Microsoft, 2018a).

Certas ferramentas de diff também recebem como entrada um histórico de edições produzido por um editor de documentos. Com esta informação se torna trivial determinar os átomos correspondentes (átomos não envolvidos em edições) e detectar operações de edição complexas suportadas pelo editor, como refatorações de código automáticas. Por outro lado, essa abordagem requer o uso de editores específicos que forneçam esse histórico e, a depender da granularidade das edições fornecidas, pode dificultar a detecção das operações de edição (MALPOHL; HUNT; TICHY, 2003). Este trabalho foca em estratégias que não requerem o histórico de edições, suportando dessa forma a comparação de documentos editados por qualquer programa.

### 2.1.1 Documento

Um documento é uma cadeia — i.e. uma sequência finita de símbolos de um alfabeto justapostos (MENEZES, 2000) — que representa um conjunto de dados e pode ser referenciada através de um identificador único (Microsoft, 2018a).

Um documento cujo alfabeto (da cadeia) é um conjunto de caracteres e está organizado em zero ou mais linhas é denominado documento textual. ASCII (GORN; BEMER; GREEN, 1963) e Unicode (Unicode Consortium et al., 2018) são exemplos de conjuntos de caracteres. Uma linha é uma sequência de caracteres sem separadores de linhas, seguida de um separador de linha ou do final do documento. Separadores (ou quebras) de linhas, representados aqui por  $\leftarrow$ , são cadeias arbitrárias que indicam o final de uma linha (IEEE; The Open Group, 2018).

As quebras de linha organizam o texto como uma tabela, onde cada célula contém um caractere. A numeração ordinal das linhas e colunas dessa tabela é extremamente útil para referenciar posições do documento de forma precisa e intuitiva, como ilustrado na figura 2.2, sendo utilizada por diversos editores de texto e ferramentas de diff.

≡ a.txt		≡ b.txt	
1	a	1	w
2	b	2	a
3	c	3	b
4	d	4	x
5	e	5	y
6	f	6	z
7	g	7	e

Figura 2.2: Documentos textuais com identificadores “a.txt” e “b.txt”. Os números em azul são os ordinais das linhas

As posições do documento são representadas neste trabalho usando a notação  $l:c$ , onde  $l$  é o ordinal da linha e  $c$  o da coluna, ambos contados a partir do 1. Por exemplo, no documento “a.txt” da figura 2.2, a posição 3:1 contém o caractere “c” e a posição 3:2 contém o caractere “ $\leftarrow$ ”.

Um documento não textual é denominado documento binário. Devido a grande diferença nas formas de analisar e exibir documentos textuais e binários é razoável que uma ferramenta de diff dê suporte a apenas um ou outro (IEEE; The Open Group, 2018). Além disso, os documentos binários possuem menor importância no contexto de Software Configuration Management (SCM), pois costumam ser gerados automaticamente a partir de documentos textuais contendo código-fonte (DART, 1991). Por esses motivos, este trabalho aborda somente ferramentas de diff para texto, apesar de diversos algoritmos apresentados também serem aplicáveis a documentos binários (HUNT; VO; TICHY, 1998).

### 2.1.2 Diretório

Um diretório é o nó raiz de uma árvore cujas folhas são documentos e os demais nós com filhos são denominados subdiretórios. Cada nó é rotulado com um identificador único

entre os seus irmãos (IEEE; The Open Group, 2018).

A partir de um algoritmo de diff de documentos é relativamente simples construir um algoritmo de diff de diretórios, bastando aplicar um algoritmo de correspondência de diretórios (e documentos). O algoritmo mais tradicional é recursivo: assume-se que os diretórios da entrada são correspondentes, e que os filhos são correspondentes se possuem identificadores equivalentes e os pais são correspondentes (MACKENZIE; EGGERT; STALLMAN, 2017).

Há algoritmos de correspondência de diretórios mais avançados, que utilizam heurísticas e algoritmos de otimização. Ainda assim, tais técnicas são incrementos sobre algoritmos de diff de documentos (SILVA et al., 2014). Por esse motivo, este trabalho foca nos algoritmos de diff de documentos, deixando os diffs de diretórios para trabalhos futuros.

## 2.2 ANÁLISE DA ENTRADA

Antes que a ferramenta possa comparar os itens da entrada é necessário analisá-los e convertê-los numa representação intermediária, no caso, uma estrutura de dados contendo átomos compatível com o algoritmo de diff. Essas estruturas podem ser sequências, árvores, grafos, etc, e representam os relacionamentos entre os átomos de cada item da entrada (KIM; NOTKIN, 2006).

Um átomo<sup>3</sup> é um elemento lógico que a ferramenta de diff foi capaz de reconhecer na entrada. Alguns tipos de átomos comuns nas ferramentas tradicionais são: diretórios, arquivos, linhas, palavras e caracteres (MACKENZIE; EGGERT; STALLMAN, 2017; PINARD; GINGERICH; GAGERN, 2014; TORVALDS; HAMANO et al., 2018). O conjunto dos tipos de átomos que a ferramenta é capaz de reconhecer determina a granularidade das edições presentes nos diffs da ferramenta (SILVA et al., 2014).

A localização dos átomos nos documentos é armazenada durante a análise da entrada para ser utilizada nas etapas seguintes da ferramenta. Uma localização é um intervalo de um documento associado ao identificador deste documento e representa unicamente uma ocorrência de uma subcadeia na entrada. Um intervalo, por sua vez, é um par ordenado de posições, formado pela posição de início e pela posição de fim do intervalo (Microsoft, 2018a).

Por exemplo, o UNIX Diff reconhece os documentos como sequências de linhas, usa o número ordinal das linhas para representar a localização dos átomos e usa caminhos do sistema de arquivos como identificadores dos documentos. Devido às limitações dos computadores contemporâneos ao UNIX diff, átomos mais granulares que uma linha inviabilizariam o uso da ferramenta até em documentos de tamanhos típicos. Por esse motivo, o UNIX Diff otimiza o uso de memória e CPU representando cada linha internamente através de um *hash*, mais compacto e fácil de comparar. Essas sequências de *hashes* de linhas (representação intermediária; estrutura de dados contendo átomos) são comparadas por um algoritmo de diff compatível com sequências, e o diff obtido no final

---

<sup>3</sup>Na bibliografia é mais frequente o termo *símbolo* ou o próprio nome do tipo de átomo (linha, caractere, etc) ao invés de *átomo*. Khanna, Kunal e Pierce (2007) abstraíram todos esses conceitos como *átomos*.

possui granularidade de linha, ou seja, as edições envolvem sempre linhas inteiras, motivo pelo qual os ordinais das colunas não são necessários (HUNT; MCILROY, 1976).

Neste trabalho os intervalos são representados por  $[i, f)$ , onde  $i$  é a posição de início e  $f$  a de fim. Seguindo a convenção adotada pelo LSP, os intervalos são fechados em  $i$  e abertos em  $f$ , i.e. o caractere na posição  $f$  não pertence ao intervalo. As localizações são denotadas por  $I@id$ , onde  $I$  representa o intervalo e  $id$  o identificador do documento. Alguns exemplos, baseados nos documentos da figura 2.2:

- a subcadeia “ $\leftarrow$ ” ocorre em diversos intervalos nos dois documentos:  $[1:2; 2:1)$ ,  $[2:2; 3:1)$ ,  $[3:2; 4:1)$  e assim por diante;
- o intervalo  $[1:1; 3:1)$  existe nos dois documentos, contendo a subcadeia “ $a\leftarrow b\leftarrow$ ” em “a.txt” e “ $w\leftarrow a\leftarrow$ ” em “b.txt”;
- a subcadeia “ $a\leftarrow b\leftarrow$ ” existe nos dois documentos, nas localizações  $[1:1; 3:1)@a.txt$  e  $[2:1; 4:1)@b.txt$ ;
- num diff com granularidade de linha, a localização  $[1:1, 2:1)@a.txt$  contendo a subcadeia “ $a\leftarrow$ ” referencia uma linha (átomo); a localização  $[2:1, 3:1)@a.txt$  contendo a subcadeia “ $b\leftarrow$ ” constitui outro átomo.

## 2.3 ALGORITMOS DE DIFF

O algoritmo de diff realiza duas tarefas: a correspondência entre átomos e o cálculo do diff. A maioria dos algoritmos apresentados aqui é uma composição de um algoritmo de correspondência de átomos e um algoritmo de cálculo de diff, que recebe como entrada a saída do algoritmo de correspondência (KIM; NOTKIN, 2006). Há também algoritmos de diff, como o de Myers (1986), que não são formados por composição de algoritmos e realizam ambas as tarefas simultaneamente. Para simplificar o texto, os termos algoritmo de correspondência e algoritmo de cálculo de diff são utilizados em ambos os casos.

### 2.3.1 Correspondência dos átomos

O propósito do algoritmo de correspondência é encontrar similaridades entre os átomos dos itens da entrada. A entrada deste algoritmo é a representação intermediária construída na etapa anterior e sua saída são pares ordenados de átomos correspondentes, contendo um átomo da esquerda e um átomo da direita. Dois átomos são considerados correspondentes se, e somente se, seus conteúdos e localizações também são correspondentes, de acordo com os critérios definidos pelo algoritmo. Os pares geralmente são escolhidos de modo que maximizem uma métrica de similaridade determinada no projeto do algoritmo (HUNT; MCILROY, 1976).

Os algoritmos de correspondência usados nas ferramentas de diff tradicionais tem como entrada sequências de átomos. Por padrão, o conteúdo é considerado correspondente quando é igual, caractere por caractere. Existem opções de configuração que relaxam o critério de correspondência de conteúdo, como a opção de ignorar espaços em branco no final das linhas (IEEE; The Open Group, 2018). A maioria desses algoritmos se

baseia ou no problema da Maior Subsequência Comum (LCS, do inglês *Longest Common Subsequence*) ou no problema do Menor Script de Edição (SES, do inglês *Shortest Edit Script*) e adotam como métrica de similaridade o número de átomos correspondentes, visando maximizar o número de átomos não modificados no diff (HUNT; MCILROY, 1976; MYERS, 1986). Esta métrica de similaridade produz diffs pouco intuitivos em alguns cenários, o que motivou a criação de algoritmos de correspondência baseados em outras métricas (COHEN, 2010).

Para simplificar a exposição, pares de átomos correspondentes são tratados como se fossem iguais, deixando implícito que seus conteúdos e localizações podem ser diferentes, assim como fazem outros trabalhos da área. Da mesma forma, sequências de átomos (e outras representações intermediárias) onde todos os átomos são correspondentes também são consideradas iguais.

**2.3.1.1 Longest Common Subsequence (LCS)** Sejam  $A$ ,  $B$  e  $C$  cadeias,  $C$  é uma subsequência de  $A$  se, e somente se, é possível obter  $C$  a partir da remoção de símbolos de  $A$ . Por exemplo, “course” é uma subsequência de “computer science”.  $C$  é uma subsequência em comum entre  $A$  e  $B$  se, e somente se,  $C$  é uma subsequência de  $A$  e também uma subsequência de  $B$ . O problema da LCS entre duas sequências  $A$  e  $B$  consiste de encontrar uma subsequência em comum entre  $A$  e  $B$  de comprimento máximo, representada aqui por  $LCS(A, B)$  (WAGNER; FISCHER, 1974). Suas primeiras áreas de aplicação foram corretores ortográficos e bioinformática (BERGROTH; HAKONEN; RAITA, 2000).

Adaptando o problema para o contexto de ferramentas de diff,  $A$  e  $B$  se tornam sequências de átomos<sup>4</sup> e as  $LCS(A, B)$  se tornam sequências de pares ordenados de átomos correspondentes.

Partem da definição do problema algumas propriedades úteis:

1. se o último átomo de  $A$  e o último átomo de  $B$  são correspondentes em conteúdo, então este é o último par ordenado de átomos de uma  $LCS(A, B)$ ;
2. senão,  $LCS(A, B)$  será a alternativa de maior comprimento entre  $LCS(A_{0..i-1}, B)$  e  $LCS(A, B_{0..j-1})$ , onde  $A_{0..i-1}$  e  $B_{0..j-1}$  são prefixos de  $A$  e  $B$ , respectivamente, obtidos removendo o último átomo;
3.  $LCS(A_{0..i-1}, B)$  e  $LCS(A, B_{0..j-1})$  podem ter o mesmo comprimento. Nesses casos existirão mais de uma LCS entre  $A$  e  $B$ .

A partir das propriedades 1 e 2 é possível obter uma fórmula recursiva para o comprimento da LCS, i.e.  $|LCS(A, B)|$  (BERGROTH; HAKONEN; RAITA, 2000). Seja

---

<sup>4</sup>A bibliografia utiliza o conceito de *cadeia* — sequência de símbolos que pertencem a um alfabeto — porém este trabalho considera o conjunto de caracteres utilizado no documento como o alfabeto, portanto o termo *sequência* é utilizado aqui para diminuir a ambiguidade e deixar claro que os algoritmos que se aplicam a cadeias também são aplicáveis a sequências de átomos

$R[i, j] = |LCS(A_{0..i}, B_{0..j})|$ :

$$R[i, j] = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \text{ (i)} \\ R[i - 1, j - 1] + 1 & \text{se } A_i = B_j \text{ (ii)} \\ \max\{R[i - 1, j], R[i, j - 1]\} & \text{se } A_i \neq B_j \text{ (iii)} \end{cases} \quad (2.1)$$

Wagner e Fischer (1974) definiram formalmente o problema da LCS e sua solução clássica<sup>5</sup> que usa programação dinâmica e necessita de  $O(mn)$  tempo e espaço, onde  $m$  e  $n$  são os comprimentos das sequências  $A$  e  $B$ , respectivamente. A solução consiste de dispor os átomos das sequências como linhas e colunas de uma matriz e preenchê-la linha por linha, coluna por coluna, começando por  $R[0, 0]$  e seguindo a fórmula 2.1, como ilustrado na figura 2.3. O comprimento da LCS é o valor final de  $R[m, n]$ , ou seja, o valor da última coluna da última linha (WAGNER; FISCHER, 1974).

		B										
		j	0	1	2	3	4	5	6	7	8	9
A	i	0	c	b	a	c	b	a	a	b	a	
	0	0	0	0	0	0	0	0	0	0	0	0
	1	a	0	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
	2	b	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
	3	c	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
	4	d	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
	5	b	0	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>
	6	b	0	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>4</b>	<b>4</b>

Figura 2.3: Cálculo do comprimento da  $LCS(A, B)$ , onde  $A$  é “abcdbb” e  $B$  é “cbacbaaba”, usando uma versão simplificada do algoritmo “X” de Wagner e Fischer (1974). As linhas dividem a matriz em regiões onde o valor de  $R[i, j]$  é igual, os números em negrito indicam correspondência de conteúdo entre os átomos. Em verde, o resultado do algoritmo.

A LCS é encontrada fazendo *backtracking* a partir de  $R[m, n]$  sobre a matriz  $R$  já preenchida. Para isso, algumas soluções armazenam ponteiros nessa matriz indicando de onde vem o valor da célula atual, como ilustrado na figura 2.4. Outras soluções recalculam esse dado com base no valores da célula atual e de suas vizinhas. Em cada passo do caminho, cada vez que uma correspondência de conteúdo é encontrada — i.e. quando é encontrado um ponteiro na diagonal ou quando  $R[i, j] = R[i - 1, j - 1] + 1$  — o par de átomos  $(A_i, B_j)$  é inserido no início da LCS. A condição de parada do algoritmo é encontrar uma célula com valor zero. Caso haja mais de um caminho possível na mesma célula (como ocorre em  $R[2, 3]$  na figura 2.4) o algoritmo pode percorrer todos os caminhos e retornar mais de uma LCS ou simplesmente optar por um deles (BERGROTH; HAKONEN; RAITA, 2000).

Este problema foi amplamente estudado pela academia e surgiram diversos algoritmos reduzindo a complexidade de tempo e espaço do problema (BERGROTH; HAKONEN;

<sup>5</sup>Mais precisamente, trata-se de uma solução para o problema da distância de edição entre duas cadeias, do qual o problema da LCS é um subproduto. Os algoritmos aqui apresentados são simplificados para encontrar a LCS



		B										
		j	0	1	2	3	4	5	6	7	8	9
A	i	∅	c	b	a	c	b	a	a	b	a	
	0	∅	0	0	0	0	0	0	0	0	0	0
	1	a	0	0	0	↖1	-1	-1	↖1	↖1	-1	↖1
	2	b	0	0	↖1	-1	-1	↖2	-2	-2	↖2	-2
	3	c	0	↖1	-1	-1	↖2	-2	-2	-2	-2	-2
	4	d	0	1	-1	-1	1	-2	-2	-2	-2	-2
	5	b	0	1	2	-2	-2	↖3	-3	-3	-3	-3
	6	b	0	1	2	-2	-2	1	-3	-3	↖4	-4

Figura 2.4: Cálculo da  $LCS(A, B)$  onde  $A$  é “abcdbb” e  $B$  é “cbacbaaba”, usando uma versão simplificada do algoritmo “Y” de Wagner e Fischer (1974). As setas representam ponteiros indicando a origem do valor da célula. Em verde e amarelo, o caminho percorrido pelo algoritmo. Em verde as correspondências que pertencem à LCS. As LCSs nesse exemplo são  $((A_1, B_3), (A_3, B_4), (A_5, B_5), (A_6, B_8))$  (ou seja, “acbb”) e  $((A_2, B_2), (A_3, B_4), (A_5, B_5), (A_6, B_8))$  (ou seja, “bcb”) e

RAITA, 2000). O algoritmo de correspondência de átomos empregado no UNIX Diff, por exemplo, é uma solução de LCS que ficou conhecida como algoritmo de Hunt-McIlroy, com complexidade de tempo  $O(n^2 \log n)$ , na época o estado da arte (HUNT; MCILROY, 1976).

Aho, Hirschberg e Ullman (1976) provaram que em um modelo computacional baseado em comparações de igualdade (i.e. correspondência de conteúdo) entre os átomos o limite inferior da complexidade de tempo da solução é  $O(n^2)$  — onde  $n$  é o comprimento da entrada — quando o alfabeto tem tamanho irrestrito, e  $O(\sigma n)$  quando o tamanho do alfabeto é restrito a  $\sigma$ . Mais tarde, Hirschberg (1977) demonstrou que, usando um modelo computacional baseado em comparações do tipo  $\leq$  (um problema similar), o limite inferior de complexidade de tempo é  $O(n \log(n))$ . A solução teoricamente mais rápida conhecida, o algoritmo de Masek e Paterson (1980), tem complexidade de tempo  $O(n^2 / \log(n))$  (JACOBSON; VO, 1992; BERGROTH; HAKONEN; RAITA, 2000).

**2.3.1.2 Heaviest Common Subsequence (HCS)** Dada uma função  $w(a, b)$  que atribui pesos ao par ordenado de átomos  $(a, b)$  — possivelmente levando em consideração o conteúdo e/ou localização dos átomos — o problema da Subsequência Comum de Maior Peso (HCS, do inglês *Heaviest Common Subsequence*) entre duas sequências de átomos  $A$  e  $B$ , denotado por  $HCS(A, B)$ , consiste de encontrar a subsequência comum entre  $A$  e  $B$  cujos pares de átomos correspondidos maximizem o somatório dos pesos dados pela função  $w$  (JACOBSON; VO, 1992).

Este problema é uma generalização do problema da LCS, que equivale à HCS quando a função de peso é constante igual a 1. A solução clássica em programação dinâmica pode ser facilmente adaptada para encontrar a HCS, sendo necessário substituir apenas a equação  $ii$  da fórmula recursiva do comprimento da LCS. Seja  $W(A, B)$  o somatório dos pesos de uma sequência de pares de átomos dados pela função  $w$  e

$R[i, j] = W(HCS(A_{0..i}, B_{0..j}))$ , temos que:

$$R[i, j] = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \text{ (i)} \\ R[i - 1, j - 1] + w(A_i, B_j) & \text{se } A_i = B_j \text{ (ii)} \\ \max\{R[i - 1, j], R[i, j - 1]\} & \text{se } A_i \neq B_j \text{ (iii)} \end{cases} \quad (2.2)$$

Da mesma maneira, o peso da HCS é o valor final da célula  $R[m, n]$ , e os mesmos algoritmos de *backtracking* podem ser usados para obter as HCSs a partir da matriz  $R$  preenchida. Esta solução possui a mesma complexidade de tempo e espaço,  $O(nm)$ , inclusive a otimização de Hirschberg (1975) sobre a solução clássica de HCS para reduzir a complexidade de espaço à  $O(n)$  também pode ser aplicada (LI, 2008). Jacobson e Vo (1992) apresentaram uma solução com complexidade de tempo  $O(r \log(n))$ , onde  $r$  é o número de átomos com conteúdo correspondente entre as duas seqüências.

Comparado com encontrar a LCS, que é considerado um problema clássico de ciência da computação, encontrar a HCS é um problema bem menos conhecido, inclusive no contexto de ferramentas de diff. No buscador acadêmico *Google Scholar*, uma busca por “longest common subsequence” retorna aproximadamente 37.100 resultados, enquanto uma busca por “heaviest common subsequence” retorna aproximadamente 3.700 resultados. Ao incluir o termo “diff” nas buscas, o número de resultados aproximados obtidos relacionados a LCS e HCS cai para, respectivamente, 19.100 e 3.760, porém no segundo caso há apenas um resultado relevante na primeira página, sobre a ferramenta de diff *TopBlend* (CHEN et al., 2000), enquanto todos os resultados na primeira página são relevantes no caso da LCS<sup>6</sup>.

O cálculo da HCS no lugar da LCS em ferramentas de diff permite ao usuário e ao projetista do algoritmo realizar ajustes finos na forma como é feita a correspondência de átomos sem precisar desenvolver um novo algoritmo do zero. Por exemplo, uma forma de reduzir a quantidade total de caracteres envolvidos no diff, com o objetivo de facilitar sua leitura, é utilizar uma solução de HCS e adotar o comprimento dos átomos como função de peso  $w$  (CHEN et al., 2000).

### 2.3.2 Cálculo do diff

A entrada do algoritmo de cálculo de diff é composta pelas representações intermediárias construídas na etapa de análise da entrada e pelas correspondências entre átomos detectadas na etapa anterior. Sua saída é uma seqüência de edições capazes de transformar o documento à esquerda num documento equivalente ao da direita.

Cada edição traz duas informações essenciais: quais localizações da entrada estão envolvidas na edição e qual operação de edição foi realizada, ou seja, como essas localizações foram afetadas. As operações de edição tradicionais são: adição, remoção e substituição (WAGNER; FISCHER, 1974).

É comum que átomos vizinhos sejam alvos de uma mesma operação de edição. Em algoritmos baseados em seqüências de átomos, esses átomos podem ser agrupados em blocos para simplificar o diff (IEEE; The Open Group, 2018). Um bloco é um intervalo de um documento contendo um ou mais átomos, sendo que suas posições de início e de fim

<sup>6</sup>As buscas foram realizadas em novembro de 2019.

são, respectivamente, a posição de início do primeiro átomo no bloco a posição de fim do último átomo no bloco (MACKENZIE; EGGERT; STALLMAN, 2017). Por exemplo, se os átomos nas localizações [6:1, 7:1>@a.txt e [7:1, 8:1>@a.txt foram removidos, é possível simplificar o diff dizendo que o bloco [6:1, 8:1>@a.txt foi removido.

Uma edição de remoção referencia a localização no documento à esquerda do bloco removido, enquanto uma adição referencia a localização no documento à direita do bloco que foi adicionado. Uma substituição referencia as localizações do bloco substituído no documento à esquerda e do bloco que o substituiu no documento à direita (MACKENZIE; EGGERT; STALLMAN, 2017). As figuras 2.5 e 2.6 mostram exemplos de edições exibidas na saída de duas ferramentas de diff populares.

Além das operações de edição tradicionais, outra operação relativamente comum é a movimentação de blocos, que indica que um bloco foi removido à esquerda e um bloco de conteúdo equivalente foi adicionado à direita num contexto diferente. Essa edição referencia a localização original do bloco (origem) e sua nova localização (destino) (TICHY, 1984). Também existem algoritmos que detectam renomeações e outros tipos de refatorações (HUNT; TICHY, 2002).

**2.3.2.1 Algoritmo tradicional** O algoritmo de cálculo de diff tradicional — usado no UNIX Diff — parte da premissa de que o editor de um documento busca fazer o mínimo necessário de alterações para atingir seus objetivos (CANFORA; CERULO; PENTA, 2009). Por esse motivo, seu objetivo é encontrar a menor sequência de adições, remoções e substituições que transforma o documento à esquerda num equivalente ao documento à direita, maximizando a quantidade de átomos inalterados entre os documentos. Essa sequência de edições é chamada de Menor Script de Edição SES (HUNT; MCILROY, 1976; MYERS, 1986).

O algoritmo usa as correspondências entre átomos obtidas através do algoritmo de LCS para calcular o SES. Os átomos que não fazem parte da LCS são considerados diferenças e mapeados para edições: caso o átomo pertença ao documento à esquerda trata-se de uma remoção; caso contrário, trata-se de uma adição. A operação de substituição é equivalente a uma remoção e uma adição em localizações dos documentos que possuem o mesmo contexto, ou seja, em localizações que são vizinhas de átomos correspondentes entre os documentos (HUNT; MCILROY, 1976).

O cálculo do diff é feito em tempo linear, iterando simultaneamente sobre os documentos e a LCS, verificando quais átomos pertencem ou não à LCS, mapeando-os para edições e agrupando os átomos em blocos.

**2.3.2.2 Algoritmo de detecção de movimentações** Um algoritmo de LCS também pode ser usado para detectar movimentações, executando-o iterativamente sobre os átomos não correspondidos entre os documentos. A cada iteração as novas correspondências encontradas pelo algoritmo são mapeadas para edições de movimentação e as iterações seguintes são feitas sobre os átomos que não foram correspondidos, até que não seja encontrada mais nenhuma correspondência entre os documentos (TICHY, 1984). Após esse processo o algoritmo tradicional é aplicado sobre os átomos que não possuem

```

$ diff -u a.txt b.txt
--- a.txt      2017-09-21 14:30:10.449486696 -0300
+++ b.txt      2017-09-23 19:45:17.580847725 -0300
b)-@@ -1,7 +1,7 @@
  +w } c)
  +a }
  +b } d)
  -c }
  -d }
  +x } e)
  +y }
  +z }
  -e } d)
  -f } f)
  -g }

```

Figura 2.5: Diff entre os documentos “a.txt” e “b.txt” obtido com GNU diff (MACKENZIE; EGGERT; STALLMAN, 2017), com granularidade de linha. Destaques: a) identificadores de documentos da entrada; b) localização do trecho em exibição em cada documento, com intervalos de posições representados por ordinais das linhas; c) adição; d) contexto (blocos contendo átomos correspondidos); e) substituição; f) remoção.

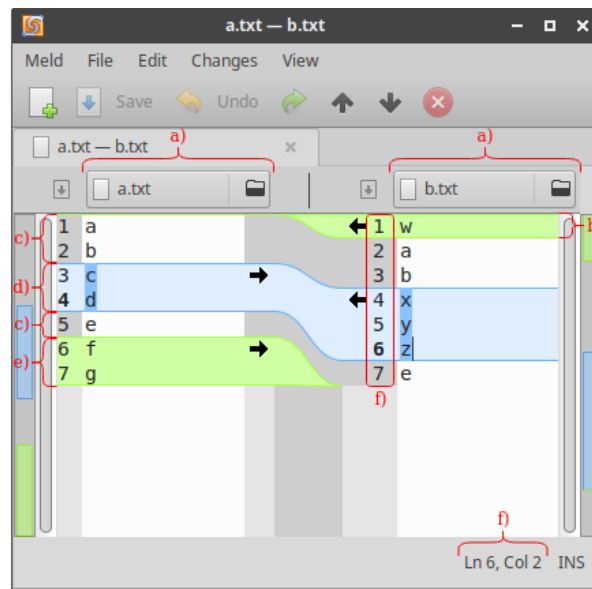


Figura 2.6: Diff entre os documentos “a.txt” e “b.txt” obtido com o Meld (WILLADSEN et al., 2018), com granularidade mista de linha e caractere. Destaques: a) identificador do documento de entrada; b) adição; c) contexto; d) substituição; e) remoção; f) uso dos ordinais das linhas e dos caracteres para expressar posições. Além disso, o Meld utiliza um código de cores para indicar linhas e caracteres editados.

```

$ git diff --color-moved --no-index before.c after.c
diff --git a/before.c b/after.c
index ba1970f..cb62985 100644
--- a/before.c
+++ b/after.c
@@ -1,15 +1,15 @@
-void func2() {
-    x += 2 // increments x by two
-}
-
-void functhreehalves() {
-    x += 1.5
-}
-
void func1() {
    x += 1 // increments x by one
}

+void func2() {
+    x += 2 // increments x by two
+}
+
void func3() {
    x += 3 // increments x by three
}
+
+void functhreehalves() {
+    x += 1.5
+}

```

Figura 2.7: Diff obtido com o Git Diff (TORVALDS; HAMANO et al., 2018), com granularidade de linha e detecção de movimentações. Os blocos de origem são exibidos em duas cores alternadamente e os blocos de destino são exibidos em azul claro.

correspondentes nem foram mapeados para movimentações (SILVA et al., 2014).

## 2.4 SAÍDA

O requisito mínimo da saída da ferramenta de diff — o diff propriamente dito (RAYMOND et al., 2004) — é indicar as localizações e operações da sequência de edições detectadas pelo algoritmo de cálculo de diff. Um diff pode ser usado para diversos fins, que podem impor requisitos adicionais sobre a representação das edições.

Por exemplo, o diff pode ser usado para transmitir edições feitas em documentos através da Internet e para armazenar múltiplas versões das entradas em disco de forma compacta, sendo necessário manter apenas as diferenças entre elas ao invés de cada versão por inteiro. Nos dois últimos cenários é necessário que a saída da ferramenta de diff seja a mais compacta possível <sup>7</sup>, portanto o formato de saída mais adequado a esse cenário não necessariamente será legível por seres humanos (HUNT; VO; TICHY, 1998).

Um outro uso possível para o diff é auxiliar seus usuários na compreensão das mudanças em código-fonte, o foco principal desse trabalho. Algumas funcionalidades importantes presentes em ferramentas de diff utilizadas com esse propósito são:

- presença de contexto das edições na saída, ou seja, dos trechos não editados ao redor das edições, agregando mais informação sobre as edições (IEEE; The Open

<sup>7</sup>Esses diffs normalmente são chamados de *patches* ou *deltas* (RAYMOND et al., 2004).

Group, 2018);

- localizações representadas de uma forma que permita navegar até elas em outras ferramentas — principalmente editores de texto —, facilitando a tomada de ação com base no diff (MACKENZIE; EGGERT; STALLMAN, 2017);
- emprego de código de cores sobre o texto, facilitando a identificação das operações de edição (TORVALDS; HAMANO et al., 2018; WILLADSEN et al., 2018);
- combinação de edições usando múltiplas granularidades num mesmo diff, ajudando a dar maior destaque ao que realmente foi alterado quando os átomos são muito extensos (por exemplo, linhas longas) (WILLADSEN et al., 2018);

As figuras 2.5, 2.6 e 2.7 mostram exemplos de diff entre os documentos da figura 2.2 produzidos por diferentes ferramentas exemplificando tais funcionalidades. Graças ao desenvolvimento dessas e de outras funcionalidades a compreensão do diff foi se tornando mais fácil, porém dois problemas que se manifestam na saída das ferramentas de diff ainda permanecem sem uma solução satisfatória: o ruído e a lacuna semântica.

### 2.4.1 Ruído

Ruído no diff é qualquer edição irrelevante apresentada sem distinção em relação às edições relevantes para o usuário naquele momento. O ruído leva o usuário a analisar trechos de documentos sem necessidade, tornando a compreensão das mudanças mais demorada, trabalhosa e sujeita a erros. Uma das dificuldades de eliminar o ruído dos diffs é que a relevância das diferenças é algo definido subjetivamente pelo usuário, de acordo com o cenário de uso do diff (YANG, 1991).

Por exemplo, durante a revisão do código de uma nova funcionalidade o usuário do diff está interessado em diferenças que poderiam afetar o comportamento do software, enquanto diferenças de formatação de código entre as versões comparadas são irrelevantes. Se a ferramenta utilizada não possuir um meio eficaz de rotular ou eliminar as diferenças de formatação no diff o usuário só vai perceber que poderia ter ignorado essas diferenças após analisá-las.

Já ao revisar uma mudança no estilo de formatação do projeto, o usuário está interessado justamente nas diferenças de formatação, que se enquadrariam na definição de ruído no cenário anterior, e espera que não haja mudanças que afetem o comportamento do programa. Nesse caso uma ferramenta que sempre elimina as diferenças de formatação do diff não serviria. Considerando o diff da figura 2.8, no primeiro cenário o ruído seria predominante enquanto no segundo cenário não haveria ruído.

Por esta razão, as ferramentas de diff oferecem opções para reduzir o ruído que podem ser configuradas pelo usuário levando em consideração o critério de relevância do momento, assegurando a aplicabilidade da ferramenta de diff nos mais diversos cenários. Um exemplo é a funcionalidade de ignorar espaços em branco no final das linhas, presente como uma opção em praticamente todas as ferramentas de diff desde o UNIX Diff (IEEE; The Open Group, 2018).

```

$ diff -u before.json after.json --color
--- before.json      2019-01-24 00:01:28.386837378 -0300
+++ after.json       2019-01-24 00:01:27.230824790 -0300
@@ -1,13 +1,16 @@
 {
-   "message": "ValidationFailed",
-   "errors": [
-     {
-       "resource": "Issue",
-       "field": "milestone",
-       "code": "invalid_number",
-       "value": 1
-     }
-   ],
-   "version": {"major": 1, "minor":2}
+   "message": "Validation Failed",
+   "errors": [
+     {
+       "resource": "Issue",
+       "field": "milestone",
+       "code": "invalid_number",
+       "value": 1.0
+     }
+   ],
+   "version": {
+     "major": 1,
+     "minor":2
+   }
 }

```

Figura 2.8: Diff com diferenças de formatação obtido com a ferramenta GNU Diff, comparando dois objetos representados na linguagem JSON (CROCKFORD, 2006). Exceto pela diferença de valor da chave `message`, todas as diferenças interferem apenas na formatação dos documentos e provavelmente seriam consideradas irrelevantes pelo usuário do diff

**2.4.1.1 Causas** As gramáticas de linguagens de computador normalmente permitem a utilização de elementos sem efeito semântico para facilitar a leitura por humanos, como comentários, espaços em branco, etc. Elas também admitem várias formas de representar o mesmo dado, como o número 1 na linguagem JSON, que pode ser representado por `1` ou `1.0` (CROCKFORD, 2006), ou a cor branca na linguagem CSS, que pode ser representada por `white`, `#FFFFFF`, `#FFF`, `#fff`, `"frobnitz"` e `'frobnitz'` que representam a mesma cadeia em JavaScript, etc (MOZILLA et al., 2019).

A granularidade de linha — a mais comum nas ferramentas de diff tradicionais — amplifica os níveis de ruído se comparada a granularidades mais finas como caractere e palavra, que são menos utilizadas. Por exemplo, uma edição como a remoção de um espaço no final de uma linha, sem efeito semântico na maioria das linguagens, é suficiente para que a ferramenta considere que a linha inteira é diferente. Trechos contendo apenas diferenças de indentação são considerados completamente diferentes, mesmo em

linguagens onde a indentação não interfere na semântica.

São consideradas diferenças semânticas em um componente de um programa apenas aquelas que afetam as saídas produzidas por ele (HORWITZ, 1990). Elementos complexos como métodos, blocos de código e até programas inteiros podem ser representados de formas muito diferentes e ao mesmo tempo serem iguais do ponto de vista semântico. Por exemplo, refatorações são mudanças que visam melhorar a estrutura interna de um software sem afetar seu comportamento externo (FOWLER, 2018), portanto a princípio não introduzem diferenças semânticas.

Os ruídos no diff, no contexto investigado, na maioria das vezes são causados por diferenças não semânticas, classe que compreende desde simples mudanças de formatação até refatorações (HORWITZ, 1990). No geral, ou essas diferenças não são relevantes para o usuário do diff ou são menos relevantes que as diferenças semânticas e ocultá-las pode reduzir o ruído no diff. Entretanto, determinar se uma diferença é semântica ou não é equivalente a determinar se dois programas são semanticamente equivalentes, que é um problema conhecidamente indecidível. Conseqüentemente, uma ferramenta de diff no máximo será capaz de obter uma aproximação segura das diferenças semânticas, isto é, com falsos positivos mas sem falsos negativos (KIM; NOTKIN, 2006).

### 2.4.2 Lacuna semântica

Lacuna semântica é a distância que existe entre o significado de alto nível das diferenças e a representação delas no diff. Quanto maior essa distância, mais informação o usuário precisa inferir, e conseqüentemente maior o esforço necessário para análise do diff. Assim como o ruído, a lacuna semântica torna a compreensão das mudanças mais demorada e sujeita a erros.

A lacuna semântica é inversamente relacionada à riqueza de significado das operações de edição e à precisão da ferramenta em delimitar a localização das diferenças. No cenário ideal a ferramenta de diff seria capaz de informar o significado das mudanças, ao invés das operações de edição de texto genéricas, e delimitá-las de forma precisa, poupando o usuário do trabalho de inferir o significado e a extensão do impacto das mudanças. Este conceito de ferramenta é chamado de diff semântico (FOWLER, 2004).

Por exemplo, a adição de um novo parâmetro a uma função existente em TypeScript (TypeScript, 2018) pode ser descrita de várias formas num diff, dentre elas:

1. “substituir a linha 5 do documento `a/business.js`,  
i.e. `function foo(bar: int)`, pela linha 5 do documento `b/business.js`,  
i.e. `function foo(bar: int, throwOnError: boolean)`”;
2. “adicionar a cadeia `, throwOnError: boolean` na posição 5:22 do documento `business.js`”;
3. “adicionar o parâmetro `throwOnError` do tipo `boolean` à função `foo` no módulo `business`, tornando-a compatível com a interface `IFoo`”.

Da primeira forma em diante as operações de edição vão se tornando mais complexas e precisas, e a dificuldade de compreensão da mudança vai diminuindo em função da



diminuição da lacuna semântica. A forma 1, com granularidade de linha, é como a maioria das ferramentas tradicionais representa as diferenças. A forma 2 usa uma granularidade mais fina e, mesmo sem exigir que a ferramenta entenda o que é um parâmetro de função, se aproxima mais do significado da diferença apenas por delimitar as diferenças de forma mais precisa. A forma 3 representa um diff semântico hipotético.

**2.4.2.1 Causas** O conhecimento raso das ferramentas tradicionais sobre os documentos é o que as impede de exprimir operações de edição além de adições, remoções, substituições. Poucas ferramentas tradicionais vão um pouco além e detectam outras edições como movimentações e mudanças de indentação, que são possíveis de detectar sem conhecimento sobre a linguagem do documento.

Além disso, ferramentas não sensíveis à linguagem são mais suscetíveis a falhas de correspondência, ou seja, situações como a representada na figura 2.9 em que a ferramenta é incapaz de determinar quais são os átomos correspondentes entre os documentos de forma intuitiva para o usuário, fenômeno chamado de dessincronização<sup>8</sup>, que aumenta a lacuna semântica significativamente.

## 2.5 FERRAMENTAS SENSÍVEIS À LINGUAGEM

O conjunto dos tipos de átomos reconhecidos pela análise da entrada determina se a ferramenta de diff é sensível à linguagem dos documentos ou não. As ferramentas sensíveis à linguagem assumem que o documento pertence a uma linguagem formal e empregam técnicas usadas em compiladores, como análise léxica, sintática, semântica, otimização de código, verificação formal, etc (AHO; SETHI; ULLMAN, 1986). Os átomos detectados por essas ferramentas são os próprios elementos gramaticais da linguagem, como lexe-mas, nós de Árvore Sintática Abstrata (AST, do inglês *Abstract Syntax Tree*) e nós de AST com anotações semânticas (YANG, 1991; Códice Software, 2018). As ferramentas mais avançadas para comparação de programas utilizam representações intermediárias similares a grafos de fluxo de controle (HORWITZ, 1990; APIWATTANAPONG; ORSO; HARROLD, 2004; LAHIRI et al., 2012).

A grande vantagem dessa abordagem é que ela elimina diversas variações na forma de representar os átomos que não tem impacto sobre a semântica dos documentos, além de extrair mais informação sobre os átomos, o que contribui para tornar as etapas de correspondência de átomos e de cálculo do diff mais inteligentes. A maior desvantagem costuma ser a dificuldade de extensão dessas ferramentas para suportar mais linguagens, uma tarefa que pode durar semanas mesmo quando a arquitetura da ferramenta não está acoplada a nenhuma linguagem específica (HUNT; TICHY, 2002).

Neste trabalho os diffs são classificados de acordo com o nível de profundidade necessário na análise da entrada. Diffs com granularidade de *bytes*, como os obtidos usando o comando `cmp` do UNIX, são chamados de diffs binários (MACKENZIE; EGGERT; STALLMAN, 2017); diffs envolvendo linhas, palavras e caracteres, como os obtidos com as ferramentas de diff tradicionais, são chamados de diffs textuais; diffs com granulari-

---

<sup>8</sup>Tradução livre de *missynchronize* (HUNT; MCILROY, 1976)

```

diff --git a/before.c b/after.c
index 6faa5a3..e3af329 100644
--- a/before.c
+++ b/after.c
@@ -1,26 +1,25 @@
#include <stdio.h>

-// Frobs foo heartily
-int frobnitz(int foo)
+int fib(int n)
{
-   int i;
-   for(i = 0; i < 10; i++)
+   if(n > 2)
+       {
+           return fib(n-1) + fib(n-2);
+       }
+   return 1;
+
+// Frobs foo heartily
+int frobnitz(int foo)
+{
+   int i;
+   for(i = 0; i < 10; i++)
+       {
+           printf("Your answer is: ");
+           printf("%d\n", foo);
+           return fib(n-1) + fib(n-2);
+       }
+   return 1;
+}

-int fact(int n)
+// Frobs foo heartily
+int frobnitz(int foo)
+{
-   if(n > 1)
+   int i;
+   for(i = 0; i < 10; i++)
+       {
-       return fact(n-1) * n;
+       printf("%d\n", foo);
+       return fact(n-1) * n;
+       }
-   return 1;
-}

int main(int argc, char **argv)
{
-   frobnitz(fact(10));
+   frobnitz(fib(10));
}

diff --git a/before.c b/after.c
index 6faa5a3..e3af329 100644
--- a/before.c
+++ b/after.c
@@ -1,26 +1,25 @@
#include <stdio.h>

+int fib(int n)
+{
+   if(n > 2)
+       {
+           return fib(n-1) + fib(n-2);
+       }
+   return 1;
+
+// Frobs foo heartily
+int frobnitz(int foo)
+{
+   int i;
+   for(i = 0; i < 10; i++)
+       {
-       printf("Your answer is: ");
-       printf("%d\n", foo);
+           printf("Your answer is: ");
+           printf("%d\n", foo);
+       }
+}

-int fact(int n)
-{-
-   if(n > 1)
-       {
-           return fact(n-1) * n;
-       }
-   return 1;
-}

int main(int argc, char **argv)
{
-   frobnitz(fact(10));
+   frobnitz(fib(10));
}

```

Figura 2.9: Exemplos de diffs obtidos com Git Diff, usando diferentes algoritmos. Do lado esquerdo o algoritmo Myers (MYERS, 1986), o padrão do Git Diff, exibindo dessincronização, e do lado direito o algoritmo Patience (COHEN, 2010), que obtém resultados muito mais intuitivos nesse exemplo. Do ponto de vista matemático, ambos os algoritmos representam corretamente as diferenças.

dade de lexemas, como os obtidos com o *diffless*, são chamados de diffs léxicos; diffs com granularidade de nó de AST, como os obtidos com o *cdiff* (YANG, 1991) e *Semantic-Merge* (Código Software, 2018; CHEN et al., 2000), são chamados de diffs sintáticos; e por fim, qualquer diff envolvendo uma análise mais profunda é chamado de diff semântico (FOWLER, 2004; HORWITZ, 1990).

Há ainda um caso especial de ferramentas de diff sensíveis a linguagem, que aplicam formatares automáticos de código aos documentos da entrada utilizando um conjunto restrito de regras de formatação. O objetivo disso é eliminar variações de formatação, além de apresentar documentos formatados na saída. Por um lado, essa técnica permite melhorar a qualidade dos diffs obtidos usando algoritmos de diff textuais e a formatação costuma facilitar a compreensão das diferenças no diff. Por outro lado, a formatação é um processo irreversível, que pode mudar radicalmente a aparência dos documentos e as localizações originais dos átomos nos documentos são perdidas, dificultando a tomada de ação a partir do diff (CHENEY et al., 2018; GROSSBART; SAVIO; MALINA, 2018).

## 2.6 TRABALHOS RELACIONADOS

Nesta seção são apresentados as principais ferramentas de diff que serviram de fonte de inspiração para este trabalho, por conta de suas inovações ou limitações.

### 2.6.1 TopBlend

TopBlend (CHEN et al., 2000) é uma ferramenta de diff sensível à linguagem para documentos HyperText Markup Language (HTML), voltada para detecção de mudanças em páginas da *web*. Ela utiliza um analisador sintático para que as edições detectadas sejam coerentes com a estrutura gramatical do documento, e suas etapas de análise da entrada, correspondência dos átomos e cálculo do diff são executadas iterativamente sobre a AST dos documentos.

Começando pelos filhos do elemento `<body>`, a cada iteração o analisador da entrada produz uma sequência de átomos (lexemas ou elementos HTML) representando os elementos da iteração atual. O algoritmo de HCS de Jacobson-Vo, mais rápido que o algoritmo apresentado na seção 2.3.1.2, é utilizado para encontrar as correspondências entre esses átomos (JACOBSON; VO, 1992).

A função de peso utilizada é o comprimento dos átomos em caracteres, priorizando correspondências entre os átomos mais extensos e assim evitando envolvê-los em edições. Isso pode resultar em um número maior de edições no diff se comparado a algoritmos baseados no problema do SES, porém as edições do diff sempre envolverão um número de caracteres menor ou igual.

Os átomos correspondentes são eliminados das iterações seguintes e cada bloco de átomos não correspondidos (mais precisamente, seus filhos) é processado em uma iteração seguinte do algoritmo, até não haver mais blocos para processar. Adições remoções e substituições são calculadas de forma similar ao UNIX Diff, a partir dos átomos sem correspondentes, e as edições são representadas visualmente por símbolos e cores em uma renderização do conteúdo dos documentos HTML.

Apesar de terem sido desenvolvidos de forma independente, o *diffless* possui várias semelhanças em relação ao TopBlend. Ambos: são sensíveis à linguagem, usam uma solução de HCS para correspondência de átomos, e usam o comprimento dos átomos em caracteres como função de peso, sendo as únicas ferramentas de diff que possuem estas características encontradas na literatura.

Diferente do *diffless*, o TopBlend é totalmente acoplado à linguagem HTML e sua implementação não se encontra disponível para download na Internet, inviabilizando seu uso até mesmo para comparar documentos HTML. Além disso, seu algoritmo não é capaz de detectar movimentações de blocos.

### 2.6.2 Git Diff

Um dos principais cenários de uso de uma ferramenta de diff é o uso conjunto com um VCS. O Git Diff é a ferramenta de diff integrada ao Git (TORVALDS; HAMANO et al., 2018), e o Git é o VCS mais popular atualmente (RhodeCode, 2016).

Por padrão, o Git Diff funciona de forma idêntica ao UNIX Diff. A maior diferença é o uso do algoritmo de Myers, que também visa encontrar um  $SES(A, B)$ , porém calcula as correspondências e edições simultaneamente, fazendo o papel de algoritmo de correspondência e de cálculo de diff ao mesmo tempo, e possui complexidade de tempo  $O(n\Delta)$ , onde  $\Delta$  é o comprimento da sequência de edições (MYERS, 1986).

Um uso típico das ferramentas de diff é comparar versões consecutivas de um documento, cenário onde há mais semelhanças do que diferenças entre as versões (ESTUBLIER, 2000), o que dá ao algoritmo de Myers uma vantagem sobre algoritmos como o de Hunt-McIlroy (usado no UNIX Diff) e de Jacobson-Vo (usado no TopBlend), cuja complexidade de tempo é proporcional a semelhanças entre os documentos (HUNT; SZYMANSKI, 1977; JACOBSON; VO, 1992).

A estratégia do algoritmo de Myers é representar o espaço de busca do SES em um grafo onde cada aresta representa uma edição ou uma correspondência e possuem, respectivamente, peso unitário ou peso nulo. Isto reduz o problema a encontrar um caminho de menor custo no grafo. Como o peso das edições é unitário é possível encontrar uma solução ótima aplicando um algoritmo de busca guloso. Por esse motivo, não é possível aplicar o algoritmo de Myers quando as correspondências e edições tem pesos variáveis, ou seja, não é possível adaptá-lo para encontrar a HCS.

Apesar das semelhanças com o UNIX Diff, o Git Diff oferece algumas funcionalidades opcionais adicionais bastante úteis, como:

- usar algoritmos além do Myers, como os algoritmos Patience (COHEN, 2010) e Histogram (PIERCE et al., 2010), que podem tornar os diffs mais intuitivos;
- diffs com granularidade de palavra e caractere;
- detecção de blocos movidos.

Por conta da popularidade do Git e das funcionalidades adicionais mencionadas o Git Diff foi escolhido como linha de base para o desenvolvimento do *diffless*, ajudando

a definir o conjunto mínimo de funcionalidades necessário para viabilizar o projeto como uma alternativa atraente aos usuários.

### 2.6.3 IDiff

IDiff (do inglês, *Iterative Diff*) é uma ferramenta de diff focada na detecção de edições relacionadas a refatorações de código, sem exigir recursos computacionais expressivos e sem utilizar algoritmos sensíveis à linguagem. O IDiff se divide em dois componentes principais: o FDiff (do inglês, File Diff), uma ferramenta de diff entre documentos, e o DDiff (do inglês, Directory Diff).

As etapas de análise de entrada e correspondência dos átomos do FDiff são executadas iterativamente, refinando a granularidade utilizada em cada iteração. Inicialmente os documentos são analisados usando granularidade de linha e as linhas correspondentes são detectadas. Na iteração seguinte, o FDiff quebra as linhas não correspondidas em palavras e busca mais correspondências usando essa granularidade. Na iteração final ele quebra as palavras não correspondidas em caracteres. Assumindo que versões consecutivas de um código-fonte diferem apenas em 5% (ESTUBLIER, 2000) e que os documentos possuem menos linhas do que palavras e menos palavras do que caracteres, iterando sobre as granularidades dessa forma o FDiff evita um grande número de comparações que seriam necessárias se usasse a granularidade de caractere ou palavra logo de início.

Assim como o UNIX Diff, o FDiff se baseia na LCS para encontrar as correspondências entre os átomos. A solução de LCS escolhida é o algoritmo de Hunt-Szymanski, uma variante do algoritmo de Hunt-McIlroy com complexidade de tempo  $O((r + n) \log n)$  no caso médio, onde  $r$  é o número de átomos correspondentes entre dois documentos (HUNT; SZYMANSKI, 1977).

Uma vez que todas as linhas, palavras e caracteres correspondentes foram detectados, o FDiff busca movimentações de blocos usando a solução de LCS iterativamente, conforme o algoritmo descrito na seção 2.3.2.2. Em seguida os átomos não correspondidos ou envolvidos em movimentações são mapeados para adições e remoções.

Várias refatorações envolvem movimentações de blocos entre documentos (FOWLER, 2018). Por esse motivo, quando o IDiff recebe diretórios como entrada entra em ação o DDiff. Esse algoritmo compara todos os documentos da esquerda com todos os documentos da direita (produto cartesiano) usando o FDiff, com o objetivo de detectar movimentações entre documentos, mesmo entre os que não são correspondentes. A partir do resultado do FDiff entre cada par de documentos, o DDiff calcula uma pontuação de similaridade entre eles e as correspondências entre documentos da esquerda e da direita são determinadas de modo a maximizar a similaridade global usando o método húngaro para problemas de otimização.

O uso de granularidades mais finas (palavra e caractere) em conjunto com a detecção de movimentações, inclusive entre documentos não correspondentes, fazem com que o IDiff detecte edições relacionadas a refatorações com maior precisão e *recall* que os algoritmos de diff tradicionais (SILVA et al., 2014).

O algoritmo de cálculo de diff com suporte a movimentações do *diffless* é inspirado no algoritmo FDiff, porém o *diffless* utiliza uma solução de HCS para correspondência

entre átomos e detecção de movimentações.

Apresenta o *diffless* como solução para o problema apresentado, expondo em detalhes a sua arquitetura e funcionamento

## SOLUÇÃO PROPOSTA

O *diffless*<sup>1</sup> é um projeto de software livre, disponibilizado sob a licença MIT<sup>2</sup>, que pode ser usado como uma ferramenta de diff, como um *framework* para construção de outras ferramentas de diff, ou ainda integrado a outros softwares como uma biblioteca.

O *diffless* é escrito primariamente na linguagem de programação TypeScript (TypeScript, 2018), usando conceitos de orientação a objetos e de programação funcional. Para utilizar sua Command Line Interface (CLI) é necessário apenas ter o interpretador Node.js<sup>3</sup> instalado no ambiente.

O nome do projeto — junção das palavras *diff* e *less* (menos, em inglês) — simboliza sua missão de reduzir o tempo e o esforço necessário para compreensão dos diffs no dia-a-dia, o que implica em reduzir o ruído e a lacuna semântica nos diffs. O objetivo desta iteração do projeto é disponibilizar uma ferramenta de diff capaz de trazer diffs sensíveis à linguagem, concisos e com detecção de blocos movidos para o cotidiano dos desenvolvedores de software.

Para atingir esses objetivos, o *diffless* introduz o algoritmo de diff *HCSDiff*, baseado em HCS e capaz de detectar movimentações de blocos além das operações de edição usuais. A função de peso utilizada na busca pela HCS ajuda o algoritmo a obter diffs mais concisos.

O modelo conceitual do *diffless* suporta diffs sensíveis a linguagem, sem acoplar o *diffless* a nenhuma linguagem específica. Diferente da maioria dos algoritmos de diff sensíveis a linguagem, o *diffless* utiliza analisadores léxicos como uma forma de aumentar a compreensão da ferramenta sobre os documentos sem tornar difícil estendê-la para suportar o maior número possível de linguagens.

---

<sup>1</sup><<https://github.com/ygormutti/diffless>>

<sup>2</sup><<https://choosealicense.com/licenses/mit/>>

<sup>3</sup><<https://nodejs.org/en/>>

### 3.1 ENTRADA

O *diffless* pode ser invocado a partir de sua CLI, apresentada na figura 3.1, que recebe como entrada um par de documentos texto identificados através de caminhos do sistema de arquivos. Atualmente diretórios e documentos binários não são suportados.

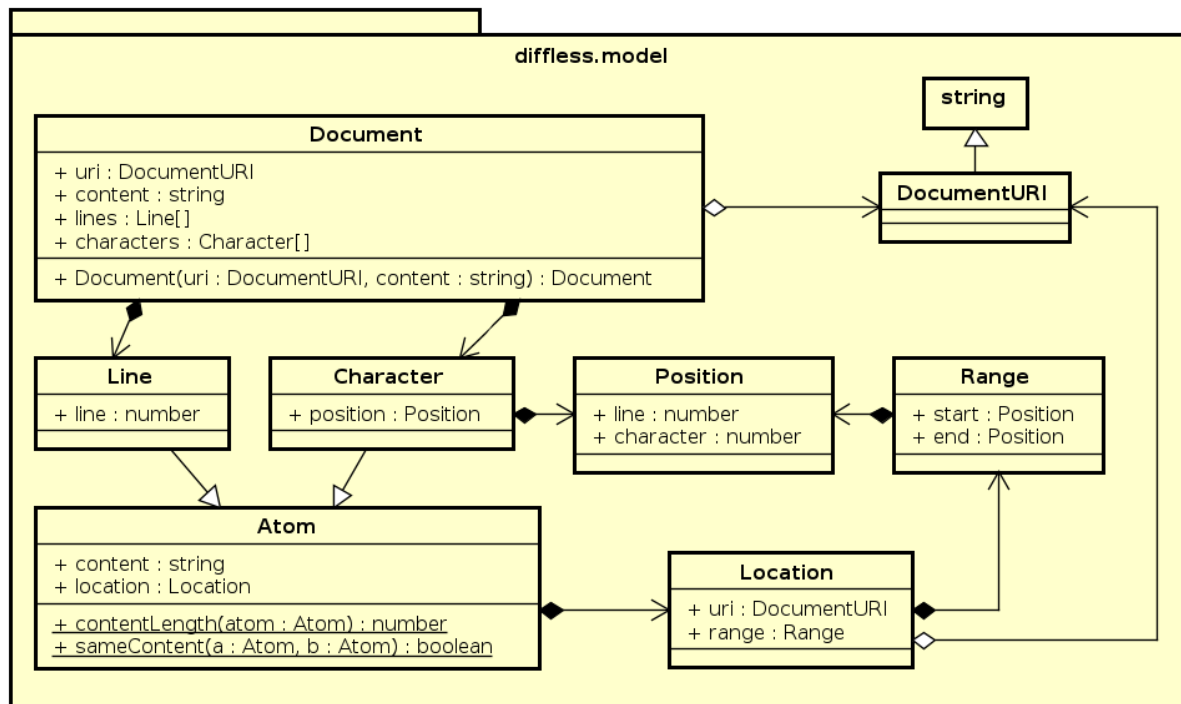
```

$ diffless --help
Usage: diffless [options] <left> <right>

Options:
  -V, --version           output the version number
  -o, --output <format> output format. One of: gui|html|json (default: "gui")
  -h, --help             output usage information
  
```

Figura 3.1: Interface de linha de comando do *diffless*

O conteúdo de cada arquivo é lido e, junto com seu identificador, usado na construção de uma instância da classe `Document` (ilustrada na figura 3.2), que representa a entrada internamente. Esta classe não é acoplada ao sistema de arquivos ou à CLI, permitindo que instâncias de `Document` sejam criadas a partir de qualquer cadeia ao usar o *diffless* como biblioteca ou *framework*.



powered by Astah

Figura 3.2: Diagrama de classes do modelo de entrada do *diffless*

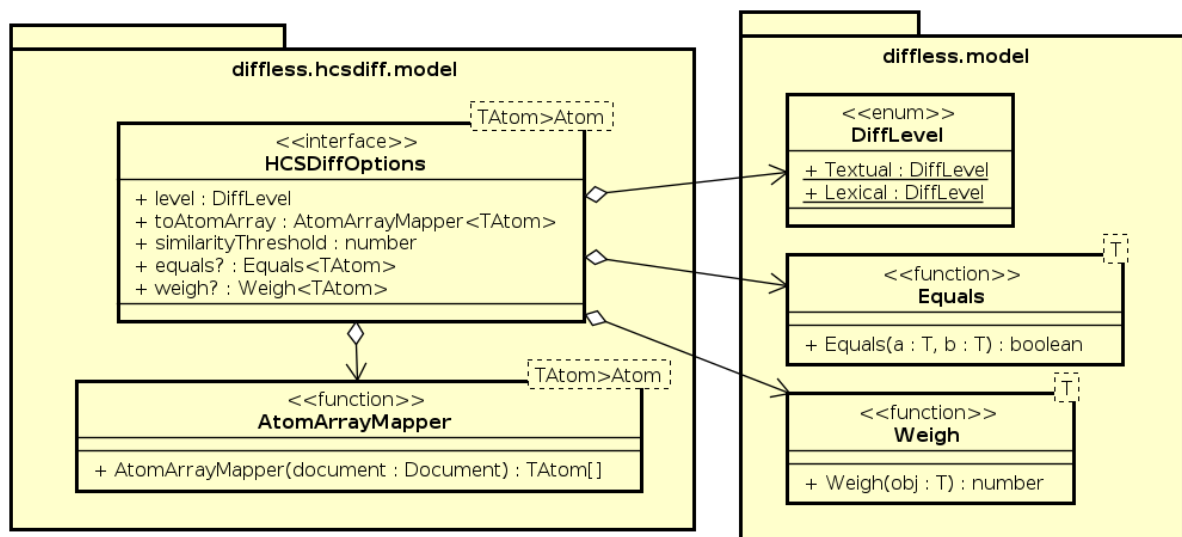


## 3.2 ANÁLISE DA ENTRADA

Durante a instanciação de `Document` é feita uma análise preliminar do documento. Os separadores de linha são normalizados e o texto é particionado em linhas e caracteres, pois esses tipos de átomos são úteis para a localização e exibição dos documentos independente da granularidade usada no diff.

As posições, intervalos e localizações são representados, respectivamente, pelas classes `Position`, `Range` e `Location`, inspiradas nas classes homônimas usadas para localização pelo LSP (Microsoft, 2018a). Os átomos, tais como linhas, caracteres e tokens, são representados por instâncias de subclasses de `Atom`.

O algoritmo `HCSDiff` é configurável e o conjunto de opções aceitas é definido pela interface `HCSDiffOptions<TAtom>`, conforme a figura 3.3. Em particular, a opção `toAtomArray` é uma função que recebe um documento e retorna uma sequência de átomos, ou seja, é responsável por realizar a análise da entrada. Dessa forma, o `HCSDiff` pode ser usado para obter diffs de qualquer granularidade, desde que a representação intermediária da entrada seja uma sequência de objetos que herdam de `Atom`.



powered by Astah

Figura 3.3: Diagrama de classes das opções do algoritmo `HCSDiff`

Para obter diffs com granularidade de linha ou caractere basta que `toAtomArray` retorne, respectivamente, o atributo `lines` ou `characters` do próprio documento. Para outras granularidades são necessários passos adicionais, como a execução de um analisador léxico.

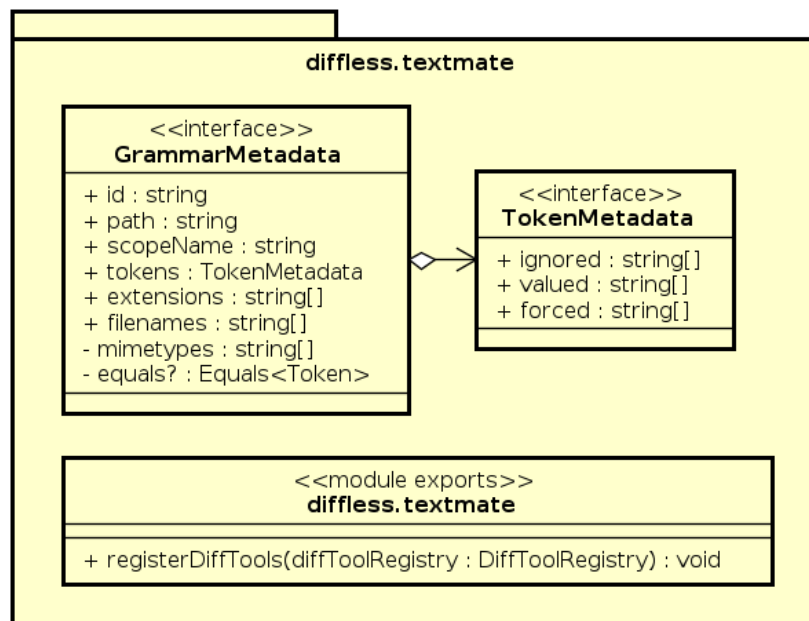
### 3.2.1 Análise léxica

A análise léxica dos documentos no `diffless` reaproveita componentes responsáveis pela funcionalidade de destaque de sintaxe do editor de código Visual Studio Code (VSCode).

Para adicionar suporte a novas linguagens ao *diffless* são necessários apenas uma gramática no formato `tmLanguage`<sup>4</sup> e alguns metadados sobre a gramática, similar a forma como novas linguagens são adicionadas ao VSCode (Microsoft, 2018b).

O formato `tmLanguage` é adotado por diversos editores de código populares além do VSCode, como TextMate<sup>5</sup> (onde o formato teve origem), Sublime Text<sup>6</sup>, Atom<sup>7</sup>, entre outros. Graças a popularidade desses editores é possível encontrar na Internet gramáticas `tmLanguage` para praticamente todas as linguagens atualmente em uso, formando um ciclo virtuoso que motiva novos editores de texto e projetistas de linguagens a suportarem o formato.

A vantagem desse modelo de extensibilidade é que, assumindo que já existe uma gramática `tmLanguage` pronta para a linguagem que se deseja adicionar ao *diffless*, não são necessários conhecimentos avançados sobre o formato `tmLanguage`, sobre análise léxica, ou mesmo sobre o *diffless*. É necessário apenas um pouco de conhecimento sobre a linguagem para definir os metadados.



powered by Astah

Figura 3.4: Diagrama de classes da integração com `vscode-textmate`

Durante a inicialização, a função `registerDiffTools` (figura 3.4) é usada para popular um registro com a configuração do *HCSDiff* mais adequada para cada linguagem suportada. Durante esse processo, os metadados de gramáticas do VSCode<sup>8</sup> representados pela interface `GrammarMetadata` são utilizados, entre outras coisas, para determinar

<sup>4</sup>[https://macromates.com/manual/en/language\\_grammars](https://macromates.com/manual/en/language_grammars)

<sup>5</sup><https://macromates.com/>

<sup>6</sup><https://www.sublimetext.com/>

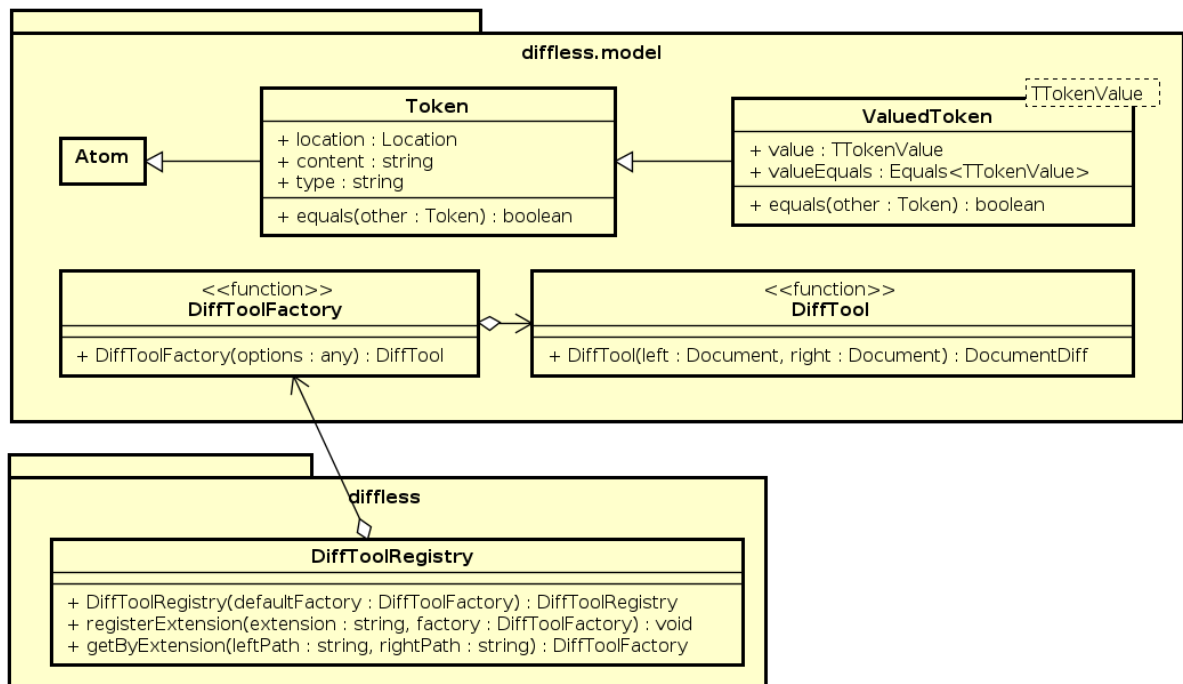
<sup>7</sup><https://atom.io/>

<sup>8</sup><https://github.com/microsoft/vscode/tree/master/extensions>

qual gramática utilizar de acordo com a extensão dos arquivos comparados. Caso nenhuma gramática adequada seja encontrada, o *diffless* utiliza granularidade de linha e caractere, permitindo seu uso mesmo quando ainda não há suporte para a linguagem desejada.

Exceto pelo atributo `tokens`, que obedece à interface `TokenMetadata` e contém os metadados específicos do *diffless*, todos os atributos de `GrammarMetadata` são preenchidos a partir de metadados de gramáticas do VSCode. Dessa forma, para adicionar uma nova linguagem ao é necessário copiar a gramática e seus metadados para dentro do projeto e preencher o `tokens`.

O *diffless* também utiliza do VSCode a biblioteca `vscode-textmate`<sup>9</sup> para interpretar as gramáticas no formato `tmLanguage` e fazer a análise léxica dos documentos. Ela informa para cada lexema identificado no texto o nome do tipo do lexema conforme definido na gramática, sua localização no texto e seu conteúdo.



powered by Astah

Figura 3.5: Diagrama de classes do modelo de análise léxica do *diffless*

O *diffless* transforma esses lexemas do `vscode-textmate` em átomos do tipo `Token` ou `ValuedToken<TTokenValue>` que representam, respectivamente, lexemas e lexemas valorados (como literais, constantes e símbolos), ou os ignora de acordo com os tipos especificados pelo `TokenMetadata`. A seguir os campos e o que significam:

- `valued`: tipos de lexemas valorados da gramática;

<sup>9</sup><https://github.com/Microsoft/vscode-textmate>

- **forced**: tipos de lexemas que não devem ser ignorados, mesmo que contenham somente espaços em branco. O `vscode-texmate` reconhece alguns espaços em branco como lexemas, apesar de não terem impacto sobre a semântica do documento em grande parte das linguagens, sendo uma das causas de ruído nos diffs tradicionais. Por padrão, o `diffless` ignora os lexemas contendo somente espaços em branco, a não ser que pertençam a esta lista;
- **ignored**: quais tipos devem ser ignorados mesmo que contenham algo além de espaços em branco. Aqui entram comentários e outros tipos de lexema que não possuem impacto sobre a semântica, mas podem ser reconhecidos por gramáticas `tmLanguage`.

Por exemplo, para adicionar suporte à JavaScript Object Notation (JSON) usando a gramática “JSON with comments” do VSCode, o `TokenMetadata` é definido assim:

```
{
  forced: [
    'string.quoted.double.json.comments',
  ],
  ignored: [],
  valued: [
    'support.type.property-name.json.comments',
    'string.quoted.double.json.comments',
    'constant.numeric.json.comments',
    'constant.language.json.comments',
  ],
}
```

Além disso, existe a opção de implementar uma função de comparação de lexemas especializada na linguagem (o `equals` de `GrammarMetadata`), para eliminar diferenças na forma de representar lexemas valorados, como os exemplos citados na seção 2.4.1.1. Lexemas não valorados são considerados iguais quando possuem o mesmo tipo, independente do seu conteúdo.

### 3.3 ALGORITMO DE DIFF

O algoritmo de diff do `diffless` — o `HCSDiff`, implementado pela classe `HCSDiffTool<TAtom>` — é uma variante do algoritmo `FDiff` do `IDiff` (SILVA et al., 2014), modificado para atender as necessidades específicas de um diff léxico multilinguagem.

Uma das principais diferenças entre o `FDiff` e o `HCSDiff` é que o primeiro utiliza uma solução de LCS para fazer a correspondência entre os átomos, enquanto o outro se baseia numa solução de LCS para isso. Os átomos de um documento podem variar bastante de tamanho, pois não existe limitação de comprimento para linhas, palavras e lexemas. No caso de um diff léxico essa variação pode ser ainda maior, pois um lexema pode ocupar várias linhas, como é o caso dos literais de cadeia multilinha.

Por conta disso, a estratégia da LCS de maximizar a quantidade de átomos correspondidos pode fazer com que o algoritmo tome escolhas ruins, que aumentam a quantidade de caracteres editados no diff ao invés de diminuir. Por exemplo, o algoritmo de LCS pode preferir evitar uma correspondência de um átomo de 100 caracteres para poder corresponder 10 átomos contendo 1 caractere cada. Usando a HCS para determinar as correspondências e o número de caracteres nos átomos como função de peso é uma forma de obter diffs menores em casos típicos — em alguns casos o número de edições pode crescer demais e superar os benefícios de ter menos caracteres editados no diff.

Outra vantagem do uso da HCS é que ela possibilita experimentar métricas diferentes de similaridade com poucas mudanças no algoritmo de diff. Por exemplo, a solução de HCS também pode ser utilizada como uma solução de LCS definindo a constante igual a 1 como função de peso, o que é feito no *diffless* durante a etapa de detecção de blocos movidos.

Assim como o FDiff, o *HCSDiff* possui um limiar de similaridade configurável, através da opção `similarityThreshold`. Ou seja, blocos correspondentes que estejam abaixo do limiar de comprimento não serão considerados correspondentes. Isto é útil para eliminar similaridades e movimentações de blocos muito curtos, que podem tornar o diff ruidoso quando a granularidade é muito fina (SILVA et al., 2014).

O algoritmo clássico de LCS adaptado para o problema da HCS, descrito na seção 2.3.1.2, é utilizado no *diffless*, devido a sua simplicidade. Tanto a função de peso como a lógica de comparação de átomos (correspondência de conteúdo) são considerados parte da entrada dessa implementação, possibilitando seu reuso nos mais diversos cenários, conforme ilustrado na figura 3.6.

Diferente do FDiff, o *HCSDiff* não itera sobre as granularidades suportadas. Ao invés disso, ele faz a união das edições de múltiplas granularidades. Esta decisão de projeto foi tomada para facilitar a comparação entre granularidades na interface gráfica do *diffless*.

### 3.4 SAÍDA

O método `run` das instâncias de `HCSDiffTool<TAtom>` é uma função do tipo `DiffTool`, que representa uma configuração do *HCSDiff*, e retorna um `DocumentDiff`. A instância de `DocumentDiff` (figura 3.7) retornada pelo `HCSDiffTool` (figura 3.6), contém os chamados itens de diff (abstração que engloba tanto as similaridades quanto as edições detectadas entre os documentos) assim como os próprios documentos.

As edições são representadas por instâncias da classe `Edit`, cujos atributos `left` e `right` são opcionais, e são preenchidos ou não de acordo com a operação de edição, conforme descrito na seção 2.3.2. As instâncias de `Similarity` representam os blocos correspondentes detectados durante a etapa de correspondência dos átomos.

A função `combine` é utilizada para combinar o resultado de diversas funções `DiffTool`. Ela concatena as listas de edições e similaridades retornadas por cada `DiffTool`. Por esse motivo, cada edição ou similaridade possui um atributo `level` que indica o nível de diff (textual ou léxico) no qual o item foi detectado, o que permite habilitar ou desabilitar a visualização de cada nível.

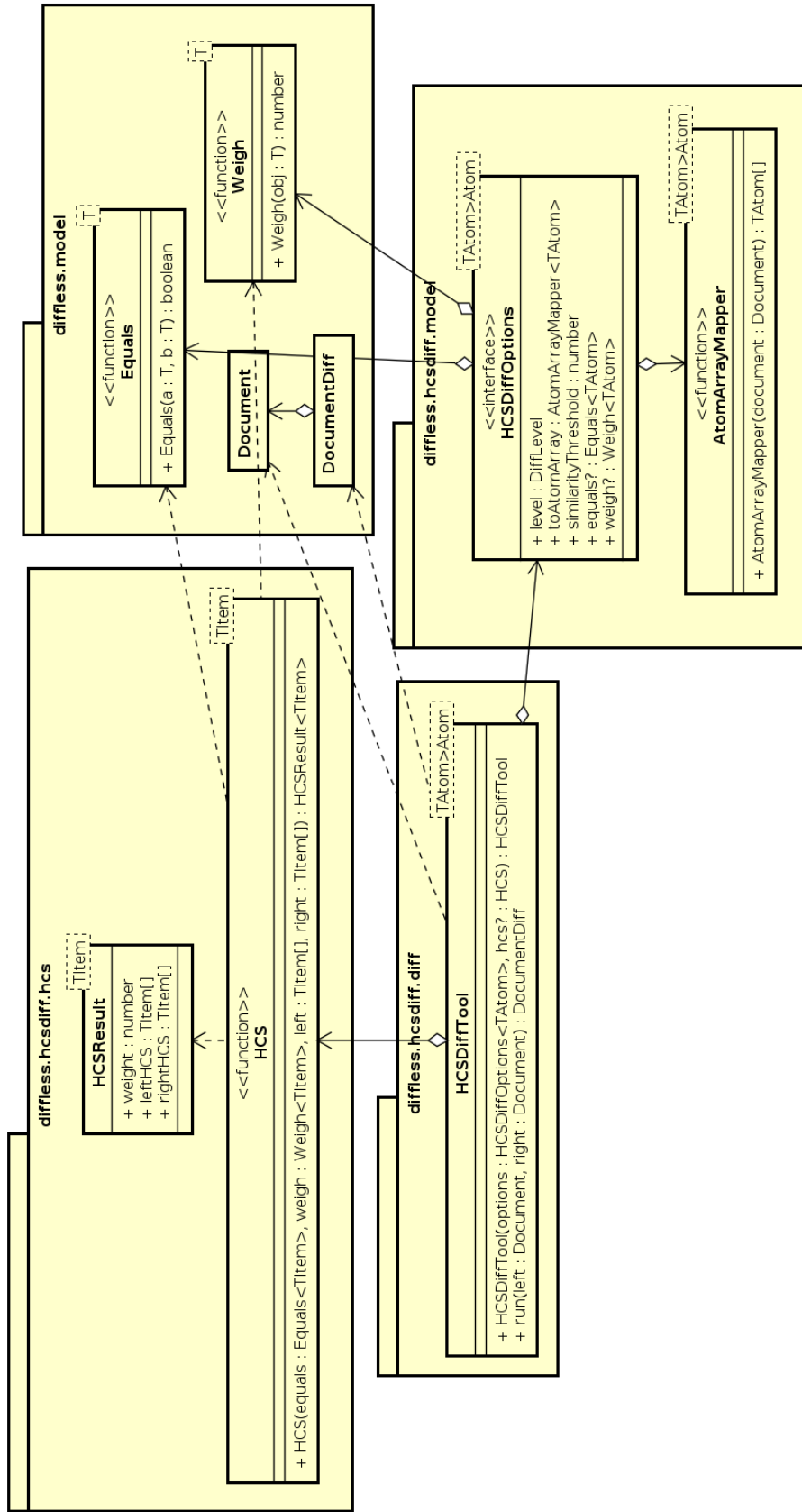


Figura 3.6: Diagrama de classes da implementação de *HCSSDiff*

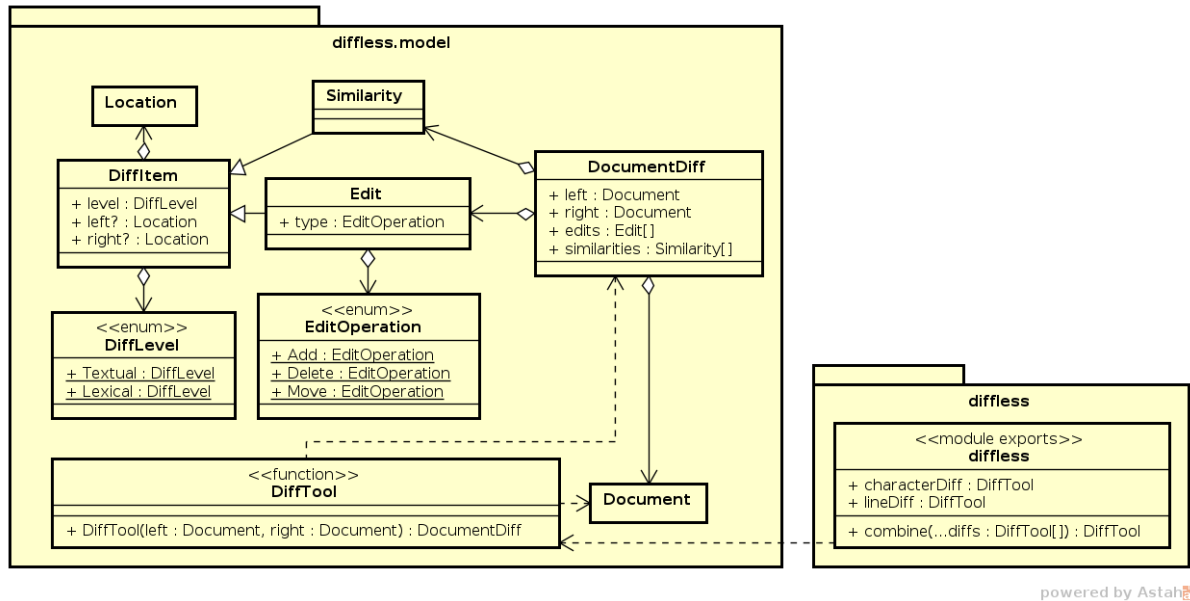


Figura 3.7: Diagrama de classes do modelo de saída do *diffless*

### 3.4.1 Formatos de saída

O *diffless* possui uma interface gráfica interativa para visualização dos diffs, implementada usando HTML, JavaScript e CSS (MOZILLA et al., 2019). As linguagens e tecnologias usadas no projeto foram todas escolhidas de modo a facilitar a integração do *diffless* a softwares *web*, como a maioria dos softwares de revisão de código.

Também é possível integrar essa interface gráfica à ferramentas *desktop* baseadas em navegadores web, como o próprio VSCode, que é construído sobre o Electron<sup>10</sup>, um navegador web voltado para construção de ferramentas *desktop*. Outra possibilidade de integração é o uso de componentes gráficos como o Qt WebView<sup>11</sup>.

A interface gráfica é acessada através da linha de comando do *diffless*, mostrada na figura 3.1, usando a opção `-o=html`, que imprime na saída padrão uma página da web estática com o diff entre os documentos, ou a opção padrão `-o=gui`, que ao invés de imprimir a página salva ela em um arquivo temporário e abre ele utilizando o navegador *web* padrão do sistema operacional.

O *diffless* utiliza um código de cores na sua interface gráfica para identificar a operação de edição. Ao posicionar o mouse sobre uma edição são exibidos detalhes sobre sua localização e, no caso de movimentações, o bloco correspondente no outro documento é destacado, conforme a figura 3.8. A interface também permite a visualização do resultado de mais de uma configuração do *HCSDiff* ao mesmo tempo, útil para comparar os resultados obtidos com cada granularidade e também para realçar as edições detectadas em mais de uma granularidade.

Por fim, há na CLI a opção `-o=json` para imprimir na saída padrão uma representa-

<sup>10</sup> <<https://electronjs.org/>>

<sup>11</sup> <<https://doc.qt.io/qt-5/qtwebview-index.html>>

```

DIFF LEVELS:  Textual  Lexical
EDIT OPERATIONS:  Add  Delete  Move

1  {
2  "message": "Validation Failed",
3  "errors": [
4    {
5      "resource": "Issue",
6      "field": "title",
7      "code": "missing_field"
8    },
9    {
10     "resource": "Issue",
11     "field": "milestone",
12     "code": "invalid"
13   }
14 ]
15 }

1  {
2  "errors": [
3    {
4      "resource": "Issue",
5      "field": "title",
6      "code": "missing_field"
7    },
8    {
9      "resource": "Issue",
10     "field": "milestone",
11     "code": "invalid"
12   }
13 ],
14 "message": "Validation Failed"
15 }

Lexical move, L: [2:5; 2:15) @ file://before.json, R: [14:3; 14:13) @ file://after.json

```

Figura 3.8: Exemplo de exibição de movimentação na interface gráfica

ção do diff em JSON (CROCKFORD, 2006), que nada mais é do que o atributo `edits` do `DocumentDiff` serializado. Isso permite a integração do *diffless* com softwares incompatíveis com as tecnologias usadas, contanto que sejam capazes de invocar a CLI a partir de uma chamada ao sistema operacional, ler o JSON da saída padrão e interpretá-lo.

### 3.5 AVALIAÇÃO PRELIMINAR

São apresentados a seguir diffs obtidos com o Git Diff e com *diffless* sobre os mesmos pares de documentos lado-a-lado. Estes documentos foram escolhidos de modo a apresentar a interface gráfica do *diffless* e ressaltar as vantagens e desvantagens percebidas em cada ferramenta. Para cada par de documentos é feita uma comparação usando as opções padrão de cada ferramenta, em seguida é feita uma comparação usando a combinação de opções mais “limpa” de cada ferramenta, isto é, a que gerou o resultado com menos caracteres editados, mas sem excluir nenhuma mudança semântica.



### Formatação e adição de caractere - Padrão

```

$ git diff --no-index before.json after.json
diff --git a/before.json b/after.json
index 2e8e71a..c91f26a 100644
--- a/before.json
+++ b/after.json
@@ -1,13 +1,16 @@
- {
-   "message": "ValidationFailed",
-   "errors": [
-     {
-       "resource": "Issue",
-       "field": "milestone",
-       "code": "invalid_number",
-       "value": 1
-     }
-   ],
-   "version": {"major": 1, "minor": 2}
+ "message": "ValidationFailed",
+ "errors": [
+   {
+     "resource": "Issue",
+     "field": "milestone",
+     "code": "invalid_number",
+     "value": 1.0
+   }
+ ],
+ "version": {"major": 1, "minor": 2}
}

```

DIFF LEVELS:  Textual  Lexical  
 EDIT OPERATIONS:  Add  Delete  Move

Figura 3.9: Neste exemplo, exceto pela primeira e última linha, todas as linhas sofreram mudanças de formatação. Como o Git Diff por padrão usa granularidade de linha todas elas foram marcadas por inteiro como edições. Enquanto isso, o *diffless* deixou bem evidente que houve mudanças de indentação. Algumas movimentações irrelevantes foram detectadas, em azul, o que indica que nesse cenário seria melhor usar um limiar de similaridade mais alto. O *diffless* também mostrou a substituição do literal 1 por 1.0 e a substituição da cadeia `ValidationFailed` por `Validation Failed`, com uma cor mais forte para o espaço que foi adicionado entre essas palavras.

### Formatação e adição de caractere - Limpo

```

$ git diff --no-index --color-words before.json after.json
diff --git a/before.json b/after.json
index 2e8e71a..c91f26a 100644
+++ a/before.json
@@ -1,13 +1,16 @@
{
  "message": "ValidationFailed", "Validation Failed",
  "errors": [
    {
      "resource": "Issue",
      "field": "milestone",
      "code": "invalid_number",
      "value": 1.0
    }
  ],
  "version": {"major": 1, "minor": 2}
},
"major": {
  "major": 1,
  "minor": 2
}
}

```

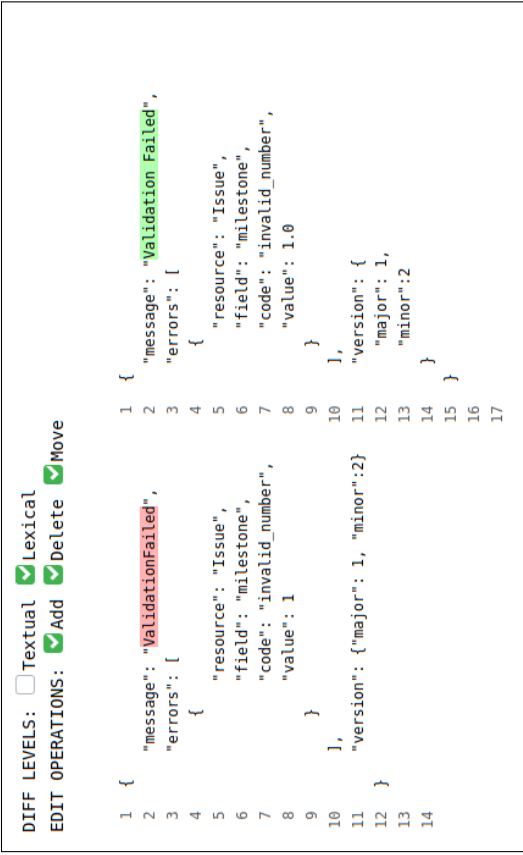


Figura 3.10: Ao habilitar a opção `--color-words` o Git Diff mudou para granularidade de palavra, o que envolve ignorar todos os espaços em branco. Se fosses documentos Python, YAML ou outra linguagem onde os espaços em branco possuem valor semântico a ferramenta teria excluído diferenças semânticas. Além disso, toda subcadeia que não envolve espaços em branco é considerada uma palavra, portanto algumas edições envolvem lexemas que não foram editados pois não há espaço em branco entre a parte que foi editada e a que não foi. Caso a opção `--ignore-all-space` fosse ativada o diff envolveria menos caracteres, porém deixaria de detectar a substituição da cadeia `ValidationFailed`. Enquanto isso, o *diffless* foi capaz de eliminar todas as mudanças de formatação, inclusive a substituição de 1 por 1.0, que em JSON não possui diferença semântica. No caso do caractere adicionado à cadeia, apesar de detectar corretamente que o conteúdo da cadeia mudou, o realce do caractere que foi modificado dentro do lexema fez falta. Marcar caracteres além do espaço inserido ajuda a sinalizar que o código que depende do intervalo sinalizado pode ser impactado pela edição, mas ao mesmo tempo aumentar o ruído da ferramenta.

### Átomo comprimido - Padrão

```

$ git diff --no-index before.json after.json
diff --git a/before.json b/after.json
index 0638ca1..07cd732 100644
--- a/before.json
+++ b/after.json
@@ -1,15 +1,14 @@
+ {
+   "message": "Validation Failed",
+   "errors": [
+     {
+       "resource": "Issue",
+       "fields": [
+         "field_with_a_really_long_name",
+         "title",
+         "author",
+         "description",
+         "field_with_a_really_long_name"
+       ],
+       "code": "missing_fields"
+     }
+   ],
+   "message": "Validation Failed"
+ }

```

DIFF LEVELS:  Textual  Lexical  
 EDIT OPERATIONS:  Add  Delete  Move

```

1 {
2   "message": "Validation Failed",
3   "errors": [
4     {
5       "resource": "Issue",
6       "fields": [
7         "title",
8         "author",
9         "description",
10        "field_with_a_really_long_name"
11      ],
12      "code": "missing_fields"
13    }
14  ],
15  "message": "Validation Failed"
16 }

```

Figura 3.11: Neste exemplo nenhuma das ferramentas se sai muito bem com suas opções padrão, mas a saída do Git Diff é a esperada para um diff com granularidade de linha. Há uma movimentação da chave "message": "Validation Failed" para a segunda linha que não é detectada como tal pelo *diffless*. Por outro lado, são detectadas várias edições pequenas, que não parecem fazer muito sentido, o que sugere que o algoritmo pode ter *bugs* ou necessita de ajustes no limiar de similaridade.

### Átomo comprimido - Limpo

```

x git diff --no-index --color-words before.json after.json
diff --git a/before.json b/after.json
index 0838ca1..07cd732 100644
--- a/before.json
+++ b/after.json
@@ -1,15 +1,14 @@
{
  "message": "Validation Failed",
  "errors": [
    {
      "resource": "Issue",
      "fields": [
        "title",
        "author",
        "description",
        "field_with_a_really_long_name"
      ],
      "code": "missing_fields"
    },
    {
      "message": "Validation Failed"
    }
  ]
}

```

DIFF LEVELS:  Textual  Lexical  
EDIT OPERATIONS:  Add  Delete  Move

```

1 {
2   "errors": [
3     {
4       "resource": "Issue",
5       "fields": [
6         "title",
7         "author",
8         "description",
9         "field_with_a_really_long_name"
10      ],
11      "code": "missing_fields"
12     },
13     {
14       "message": "Validation Failed"
15     }
16  ]

```

Figura 3.12: Neste exemplo, usar a opção `--color-words` não traz melhoria significativa para o resultado do Git Diff. O resultado obtido com o *diff*ess novamente traz uma série de problemas, como a não detecção da movimentação. Porém é possível ver o efeito do peso da HCS sobre o lexema `"field_with_a_really_long_name"` que é correspondido por ser comprimido.

### Átomo comprido - Versão antiga da análise léxica

DIFF LEVELS:  Textual  Lexical  
 EDIT OPERATIONS:  Add  Delete  Move

```

1  {
2    "errors": [
3      {
4        "resource": "Issue",
5        "fields": [
6          "title",
7          "author",
8          "description",
9          "field_with_a_really_long_name"
10       ],
11       "code": "missing_fields"
12     }
13   ],
14   "message": "Validation Failed"
15 }
16
1  {
2    "message": "Validation Failed",
3    "errors": [
4      {
5        "resource": "Issue",
6        "fields": [
7          "field_with_a_really_long_name",
8          "title",
9          "description"
10       ],
11       "code": "missing_fields"
12     }
13   ]
14 }
15

```

Figura 3.13: A integração com o `vscode-textmate` é uma funcionalidade recente do `diffless` e menos testada. Antes disso, o pacote JavaScript `json-tokenize`<sup>12</sup> era utilizado como analisador léxico de JSON. Este exemplo mostra que o analisador antigo produz um diff mais intuitivo, com menos edições. Uma das possíveis causas disso é que a gramática `tmLanguage` usada divide cada literal de cadeia em três lexemas (dois pras aspas e um para o conteúdo da cadeia) enquanto o analisador léxico antigo considera tudo como um único lexema. Nessa figura também é possível ver o efeito do peso da HCS sobre o lexema `"field_with_a_really_long_name"` que é correspondido por ser comprido.

*Oferece uma recapitulação do trabalho, destacando suas principais contribuições e propõe trabalhos futuros.*

## CONCLUSÃO

A motivação deste trabalho era disponibilizar uma ferramenta de diff sensível à linguagem, capaz de trazer esse avanço tecnológico ao cotidiano dos desenvolvedores de software, reduzindo o ruído e a lacuna semântica dos diffs, almejando facilitar a revisão de código.

Para isso foi realizada uma extensa revisão bibliográfica em busca das razões das ferramentas sensíveis a linguagem não terem se popularizado na indústria, e como evitar que isso ocorra com a ferramenta desenvolvida neste trabalho.

Essa busca mostrou que há uma tendência nas ferramentas de diff sensíveis à linguagem (principalmente da academia), de tentar aumentar o conhecimento da ferramenta sobre os documentos comparados, sem questionar muito como isso pode impactar a aplicabilidade da ferramenta no cotidiano dos desenvolvedores de software. Outra dificuldade percebida é na integração dessas ferramentas com o restante do ambiente de desenvolvimento, como ferramentas de revisão de código.

Como consequência disso, estas ferramentas são bastante complexas de estender para suportar linguagens adicionais, o que pode justificar sua baixa aceitação. Logo, uma solução seria desenvolver uma ferramenta de diff sensível a linguagem capaz de suportar muitas linguagens com pouco esforço e fácil de integrar a outros softwares.

A partir dessa premissa nasceu o *diffless*, uma ferramenta de diff léxico multilinguagem, desenvolvida usando tecnologias da *web*. Ao final desta iteração inicial, o resultado obtido é um protótipo de ferramenta de diff, que parece promissora sob diversos aspectos, mas que ainda não está pronta para o uso cotidiano e necessita de ajustes.

Sua avaliação preliminar sugere que a abordagem adotada — usar granularidade de token e ignorar diferenças de representação através de funções de comparação sensíveis a linguagem — são boas estratégias para diminuir o ruído nos diffs. Além disso, a granularidade de lexemas ajuda a melhorar a demarcação das edições, e junto a detecção de blocos movidos ajuda a diminuir a lacuna semântica dos diffs.

O algoritmo de diff *HCSDiff*, introduzido neste trabalho, mostrou resultados promissores (melhores que os do Git Diff) principalmente em relação a quantidade de ruído

originário de mudanças de formatação. Ao mesmo tempo, outros diffs avaliados revelaram que ainda existem *bugs* e ajustes finos a serem feitos para que o *HCSDiff* tenha um desempenho aceitável na maioria dos cenários, principalmente no que diz respeito a detecção de movimentações.

O uso de gramáticas tmLanguage faz com que o projeto tenha um potencial de extensão e aplicabilidade imenso, porém também impõe desafios. Essas gramáticas foram pensadas para atender as necessidades de editores de texto, que são um pouco diferentes dos analisadores léxicos comuns. Antes da integração do *diffless* com a biblioteca *vscodetextmate* o pacote *json-tokenize* foi utilizado como analisador léxico de JSON. Este analisador em alguns cenários ainda obtém resultados melhores do que o baseado nas gramáticas tmLanguage.

## 4.1 TRABALHOS FUTUROS

As desvantagens de usar o pacote *json-tokenize* são duas: ele suporta somente a linguagem JSON, e é necessário implementar muito mais código para integrar esse (e outros analisadores léxicos) ao *diffless*. Por este motivo, um dos trabalhos futuros mais importantes para o projeto é fazer o ajuste fino da integração entre o *diffless* e o *vscodetextmate*.

Outro ponto fundamental é corrigir os *bugs* na detecção de movimentações e permitir configurar o limiar de similaridade através da sua interface, visto que esse fator afeta mais os diffs do que se pensava inicialmente, motivo pelo qual essa funcionalidade não foi implementada.

Uma vez resolvidas essas questões prioritárias há uma lista de melhorias que podem ser feitas para deixar o *diffless* mais usável, como:

- Suporte à comparação de diretórios. Uma funcionalidade essencial para o uso em conjunto com VCSs, por exemplo;
- Realizar uma avaliação quantitativa do algoritmo para validar ou refutar as hipóteses apresentadas;
- Tornar o *HCSDiff* iterativo em relação as granularidades, assim como o algoritmo FDiff (SILVA et al., 2014), pois ficou claro que a combinação de diffs de diversas granularidades, apesar de útil para compará-las, não traz bons resultados para a visualização das diferenças;
- Experimentar outras funções de peso para calcular a HCS como, por exemplo, uma função proporcional à raridade do átomo no documento, para aproximar o *HCSDiff* dos bons resultados obtidos pelo algoritmos Patience e Histogram, que priorizam a correspondência dos átomos mais raros, obtendo diff mais intuitivos para o usuário (COHEN, 2010; PIERCE et al., 2010);
- Suporte a operações de edição adicionais, como cópia e substituição propriamente dita (ao invés de adições e remoções separadas);
- Implementação do algoritmo Jacobson-Vo para obter o máximo de performance possível no algoritmo *HCSDiff*;

- Melhorias na interface gráfica, para remover a dependência da linha de comando.



## REFERÊNCIAS BIBLIOGRÁFICAS

- AHO, A.; HIRSCHBERG, D.; ULLMAN, J. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM (JACM)*, ACM, v. 23, n. 1, p. 1–12, 1976.
- AHO, A. V.; SETHI, R.; ULLMAN, J. D. *Compilers, principles, techniques*. [S.l.: s.n.], 1986. 9 p.
- APIWATTANAPONG, T.; ORSO, A.; HARROLD, M. J. A differencing algorithm for object-oriented programs. In: IEEE. *Proceedings. 19th International Conference on Automated Software Engineering, 2004*. [S.l.], 2004. p. 2–13.
- BACCHELLI, A.; BIRD, C. Expectations, outcomes, and challenges of modern code review. In: IEEE PRESS. *Proceedings of the 2013 international conference on software engineering*. [S.l.], 2013. p. 712–721.
- BERGROTH, L.; HAKONEN, H.; RAITA, T. A survey of longest common subsequence algorithms. In: IEEE. *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*. [S.l.], 2000. p. 39–48.
- BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *Uniform resource identifier (URI): Generic syntax*. [S.l.], 2004.
- CANFORA, G.; CERULO, L.; PENTA, M. D. Tracking your changes: A language-independent approach. *Software, IEEE*, v. 26, p. 50 – 57, 03 2009.
- CHEN, Y.-F. et al. Topblend: An efficient implementation ofhtmldiff in java. In: *WebNet*. [S.l.: s.n.], 2000. p. 88–94.
- CHENEY, A. et al. *Pretty Diff*. 2018. Disponível em: <<https://prettydiff.com/documentation.xhtml>>.
- COHEN, B. *Patience diff advantages*. 2010. Disponível em: <<https://bramcohen.livejournal.com/73318.html>>.
- CROCKFORD, D. *The application/json media type for javascript object notation (json)*. [S.l.], 2006.
- Código Software. *SemanticMerge*. 2018. Disponível em: <<https://www.semanticmerge.com/>>.

DART, S. Concepts in configuration management systems. In: ACM. *Proceedings of the 3rd international workshop on Software configuration management*. [S.l.], 1991. p. 1–18.

ESTUBLIER, J. Software configuration management: a roadmap. In: ACM. *Proceedings of the Conference on the Future of Software Engineering*. [S.l.], 2000. p. 279–289.

Experts Exchange. *Processing power compared: Visualizing a 1 trillion-fold increase in computer performance*. 2018. Disponível em: <<https://pages.experts-exchange.com/processing-power-compared>>.

FOWLER, M. Semanticdiff. *Martin Fowler's Bliki*, 2004.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 2018.

GORN, S.; BEMER, R. W.; GREEN, J. American standard code for information interchange. *Commun. ACM*, ACM, New York, NY, USA, v. 6, n. 8, p. 422–426, ago. 1963. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/366707.367524>>.

GROSSBART, Z.; SAVIO, R.; MALINA, F. *JSON Diff*. 2018. Disponível em: <<http://jsondiff.com/>>.

HIRSCHBERG, D. S. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, ACM, v. 18, n. 6, p. 341–343, 1975.

HIRSCHBERG, D. S. *An information theoretic lower bound for the longest common subsequence problem*. [S.l.], 1977.

HORWITZ, S. Identifying the semantic and textual differences between two versions of a program. Citeseer, 1990.

HUNT, J. J.; TICHY, W. F. Extensible language-aware merging. In: IEEE. *International Conference on Software Maintenance, 2002. Proceedings*. [S.l.], 2002. p. 511–520.

HUNT, J. J.; VO, K.-P.; TICHY, W. F. Delta algorithms: An empirical analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 7, n. 2, p. 192–214, 1998.

HUNT, J. W.; MCILROY, M. D. An algorithm for differential file comparison. *Bell Laboratories Computing Science Technical Report #41*, 1976.

HUNT, J. W.; SZYMANSKI, T. G. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, ACM, v. 20, n. 5, p. 350–353, 1977.

IEEE; The Open Group. *The Open Group Base Specifications Issue 7, 2018 edition*. [S.l.], 2018. Disponível em: <<http://pubs.opengroup.org/onlinepubs/9699919799/>>.

- JACOBSON, G.; VO, K.-P. Heaviest increasing/common subsequence problems. In: SPRINGER. *Annual Symposium on Combinatorial Pattern Matching*. [S.l.], 1992. p. 52–66.
- KHANNA, S.; KUNAL, K.; PIERCE, B. C. A formal investigation of diff3. In: SPRINGER. *International Conference on Foundations of Software Technology and Theoretical Computer Science*. [S.l.], 2007. p. 485–496.
- KIM, M.; NOTKIN, D. Program element matching for multi-version program analyses. In: ACM. *Proceedings of the 2006 international workshop on Mining software repositories*. [S.l.], 2006. p. 58–64.
- KOC, A.; TANSEL, A. U. A survey of version control systems. *ICEME 2011*, 2011.
- LAHIRI, S. K. et al. Symdiff: A language-agnostic semantic diff tool for imperative programs. In: SPRINGER. *International Conference on Computer Aided Verification*. [S.l.], 2012. p. 712–717.
- LI, R. A linear space algorithm for the heaviest common subsequence problem. *Utilitas Mathematica*, Winnipeg: University of Manitoba, Department of Computer Science, 1972-, v. 75, p. 13–20, 2008.
- MACKENZIE, D.; EGGERT, P.; STALLMAN, R. *Comparing and Merging Files – for Diffutils 3.6 and patch 2.5.4*. [S.l.], 2017. Disponível em: <<https://www.gnu.org/software/diffutils/manual/diffutils.pdf>>.
- MALPOHL, G.; HUNT, J. J.; TICHY, W. F. Renaming detection. *Automated Software Engineering*, Springer, v. 10, n. 2, p. 183–202, 2003.
- MASEK, W. J.; PATERSON, M. S. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, v. 20, n. 1, p. 18–31, 1980.
- MCINTOSH, S. et al. The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects. In: ACM. *Proceedings of the 11th Working Conference on Mining Software Repositories*. [S.l.], 2014. p. 192–201.
- MENEZES, P. F. B. *Linguagens formais e autômatos*. [S.l.]: Sagra-Luzzato, 2000.
- 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.
- Microsoft. *Language Server Protocol v3.13.0*. [S.l.], 2018. Disponível em: <<https://microsoft.github.io/language-server-protocol/specification>>.
- Microsoft. *Visual Studio Code Documentation*. [S.l.], 2018. Disponível em: <<https://code.visualstudio.com/docs>>.

MOZILLA et al. *Mozilla Developer Network Web Docs*. 2019. Disponível em: <<https://developer.mozilla.org/pt-BR/>>.

MYERS, E. W. An  $O(ND)$  difference algorithm and its variations. *Algorithmica*, Springer, v. 1, n. 1-4, p. 251–266, 1986.

PIERCE, S. O. et al. *HistogramDiff*. 2010. Disponível em: <<http://download.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/diff/HistogramDiff.html>>.

PINARD, F.; GINGERICH, D.; GAGERN, M. von. *GNU Wdiff*. [S.l.], 2014. Disponível em: <<https://www.gnu.org/software/wdiff/>>.

RAYMOND, E. S. et al. *The Jargon File, version 4.4.8*. 2004. Disponível em: <<http://catb.org/jargon/>>.

RhodeCode. Version control systems popularity in 2016. 2016. Disponível em: <<https://rhodecode.com/insights/version-control-systems-2016>>.

SEEMANN, M. *10 tips for better Pull Requests*. 2015. Disponível em: <<http://blog.ploeh.dk/2015/01/15/10-tips-for-better-pull-requests/>>.

SILVA, F. F. et al. Towards a difference detection algorithm aware of refactoring-related changes. In: IEEE. *Software Engineering (SBES), 2014 Brazilian Symposium on*. [S.l.], 2014. p. 111–120.

Sourcegraph et al. *Langserver.org: A community-driven source of knowledge for Language Server Protocol implementations*. 2018. Disponível em: <<https://langserver.org/>>.

Stack Overflow. *Stack Overflow Developer Survey 2019*. 2019. Disponível em: <<https://insights.stackoverflow.com/survey/2019>>.

TICHY, W. F. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 2, n. 4, p. 309–321, 1984.

TORVALDS, L.; HAMANO, J. et al. *Git*. [S.l.], 2018. Disponível em: <<https://git-scm.com>>.

TypeScript. *TypeScript Documentation*. [S.l.], 2018. Disponível em: <<https://www.typescriptlang.org/docs/home.html>>.

Unicode Consortium et al. *The Unicode Standard: A Technical Introduction*. 2018. Disponível em: <<https://www.unicode.org/standard/principles.html>>.

WAGNER, R. A.; FISCHER, M. J. The string-to-string correction problem. *Journal of the ACM (JACM)*, ACM, v. 21, n. 1, p. 168–173, 1974.

WILLADSEN, K. et al. *Meld*. 2018. Disponível em: <<http://meldmerge.org>>.

YANG, W. Identifying syntactic differences between two programs. *Software: Practice and Experience*, Wiley Online Library, v. 21, n. 7, p. 739–755, 1991.