UNIVERSIDADE FEDERAL DE ALFENAS

JOÃO PAULO MARTYR RAGAZZO

JSIMPLES:
UMA FERRAMENTA WEB DIDÁTICA PARA O ENSINO DE COMPILADORES

ALFENAS/MG

2025

JOÃO PAULO MARTYR RAGAZZO

JSIMPLES:
UMA FERRAMENTA WEB DIDÁTICA PARA O ENSINO DE COMPILADORES

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Orientador: Prof. Dr. Luiz Eduardo da Silva

ALFENAS/MG

2025

Sistema de Bibliotecas da Universidade Federal de Alfenas
Biblioteca Unidade Educacional Santa Clara

Martyr Ragazzo, João Paulo.
JSimples : uma ferramenta web did´atica para o ensino de conceitos de teoria de linguagens e compiladores / João Paulo Martyr Ragazzo. - Alfenas, MG, 2025.
38 f. : il. -

Orientador(a): Luiz Eduardo da Silva.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal de Alfenas, Alfenas, MG, 2025.
Bibliografia.

1. Linguagem formais e automatos. 2. Ferramentas web didáticas. 3. Teoria de Compiladores. 4. Ambiente educacional. I. Silva, Luiz Eduardo da, orient. II. Título.

Ficha gerada automaticamente com dados fornecidos pelo autor.

JOÃO PAULO MARTYR RAGAZZO

JSIMPLES: UMA FERRAMENTA WEB DIDÁTICA PARA O ENSINO DE COMPILADORES

O(A) Presidente da banca examinadora abaixo assina a aprovação do(a) Trabalho de Conclusão de Curso apresentado(a) como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Aprovada em: 03 de Dezembro de 2025.

Prof. Dr. Luiz Eduardo da Silva
Presidente da Banca Examinadora
Instituição: Universidade Federal de Alfenas

Prof. Dr. Paulo Alexandre Bressan
Instituição: Universidade Federal de Alfenas

Profa. Dr. Rodrigo Martins Pagliares
Instituição: Universidade Federal de Alfenas

RESUMO

A construção de compiladores é fundamental para compreender o design de linguagens de programação, sendo um dos cursos mais importantes da computação. Entretanto, ferramentas educacionais existentes são limitadas a interfaces desktop e não oferecem visualização de etapas críticas como a análise sintática. Este trabalho apresenta o JSimples, uma plataforma web interativa para ensino e aprensizagem de compiladores, com visualização de árvores sintáticas, execução passo a passo da linguagem Simples e *feedback* em tempo real. A avaliação por inspeção especializada, baseada nas heurísticas de Nielsen e no *framework* LORI, resultou em média de 4,0/5 para usabilidade e 4,3/5 para qualidade pedagógica.

Palavras-chave: compiladores; ambiente educacional; visualização sintática; execução passo a passo; usabilidade pedagógica.

ABSTRACT

The construction of compilers is fundamental to understanding the design of programming languages and is one of the most important courses in computer science. However, existing educational tools are limited to desktop interfaces and do not offer visualization of critical stages such as syntactic analysis. This work presents JSimples, an interactive web platform for teaching and learning compilers, featuring visualization of syntax trees, step-by-step execution of the Simples language, and real-time feedback. The evaluation through expert inspection, based on Nielsen's heuristics and the LORI framework, resulted in an average of 4.0/5 for usability and 4.3/5 for pedagogical quality.

Keywords: compilers; educational environment; syntax visualization; step-by-step execution; pedagogical usability.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A digitalização alcança desde simples atividades cotidianas até processos industriais, dando origem à internet das coisas. Por trás de toda essa transformação digital, milhões de linhas de instruções bem definidas (algoritmos) permitem que máquinas executem ações, resolvam problemas e tomem decisões sem a interferência humana (Tan; Wang, 2010).

A implementação de algoritmos é feita utilizando uma linguagem de programação: uma forma de escrita intermediária entre a linguagem natural e a linguagem de máquina. Entretanto, humanos escrevem e entendem apenas linguagens de programação. As máquinas, por outro lado, só conseguem interpretar instruções em formato binário (linguagem de máquina). Surge então a necessidade de traduzir a linguagem de programação em linguagem de máquina.

A tradução responsável pela transformação digital depende fundamentalmente de compiladores: programas especializados que traduzem linguagens de programação em código executável, permitindo que as máquinas executem as instruções escritas por programadores humanos (Alfred; Monica; Jeffrey, 2007).

A construção de um compilador fornece a base fundamental para compreender o *design* de uma linguagem de programação, tornando o curso de compiladores um dos mais importantes dentro da ciência da computação (Stamenković; Jovanović, 2023) (Baïna; Benatallah, 2021). Apesar da relevância da disciplina de teoria de linguagens e compiladores, a maioria dos estudantes aparenta não ser motivada ou interessada em estudar este tópico (Chesnevar; González; Maguitman, 2004). Isso se deve não apenas à alta complexidade desse assunto, mas também à percepção intrínseca da matemática nesse ramo (Stamenković; Jovanović; Chakraborty, 2020).

Estudos apontam que estudantes preferem métodos de aprendizado ativos em oposição a métodos de aprendizado passivo baseados apenas em literaturas (Stamenković; Jovanović; Chakraborty, 2020) (Carstens *et al.*, 2021). A teoria de linguagens e compiladores envolve tópicos em um nível de abstração que torna difícil tanto aprender quanto ensinar (Chesnevar; González; Maguitman, 2004). Como Mernik e Zumer (Mernik; Zumer, 2003) apontam, os professores são constantemente desafiados a ensinar cada vez melhor. Dessa forma, recursos didáticos interativos que facilitem a complexidade de aprender e ensinar interessam a alunos e professores (Arnaiz-González *et al.*, 2018).

Este trabalho apresenta o desenvolvimento e avaliação de uma plataforma web para auxiliar no processo de ensino-aprendizagem da disciplina de compiladores, alinhando-se com

as preferências dos alunos por métodos de aprendizado ativo, como destacado por Stamenković, Jovanović e Chakraborty (Stamenković; Jovanović; Chakraborty, 2020).

1.1 OBJETIVOS GERAIS

Este trabalho tem como objetivo principal efetivar a construção e posterior estudo da usabilidade e qualidade pedagógica de uma plataforma *web* para auxiliar no processo de ensino-aprendizagem da disciplina de teoria de linguagens e compiladores.

1.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo principal, este trabalho estabelece quatro objetivos específicos. Primeiramente, realizar uma revisão bibliográfica em busca de soluções semelhantes existentes na literatura. Em seguida, estudar ferramentas que possam auxiliar na implementação. O terceiro objetivo consiste em desenvolver e implementar a plataforma *web* para ensino de teoria de linguagem e compiladores. Por fim, avaliar a qualidade por meio de inspeção heurística de usabilidade de Nielsen (Nielsen; Molich, 1990) e LORI (Learning Object Review Instrument) (Vargo J.C. Nesbit; Archambault, 2003).

2 REVISÃO BIBLIOGRÁFICA

A disciplina de teoria de linguagens e compiladores apresenta desafios tanto para professores quanto alunos. Diversos pesquisadores e professores buscam estratégias e ferramentas para facilitar a compreensão desses conceitos. Nesta seção, apresentaremos estudos e soluções que têm se proposto a auxiliar o ensino desta disciplina, com foco em ferramentas que utilizam recursos interativos para apoiar a aprendizagem de linguagens formais e compiladores.

Lisa, uma ferramenta *desktop* desenvolvida por Mernik e Zumer (Mernik; Zumer, 2003), foi um *software* pioneiro destinado a auxiliar no processo de ensino-aprendizagem de teoria de linguagens e compiladores, apresentando métodos para visualização das principais etapas do processo de compilação, como a análise léxica, sintática e semântica.

Por ser um aplicativo *desktop*, enfrenta problemas de compatibilidade para utilização em diferentes sistemas operacionais e requer instalação local. Como Stamenković, Jovanović e Chakraborty (Stamenković; Jovanović; Chakraborty, 2020) destacam, *softwares* educacionais devem ser desenvolvidos em ambiente *web*, solucionando problemas de compatibilidade, instalação e acesso multiplataforma. Este trabalho adota essa abordagem, alinhando-se com essas observações, implementando também a funcionalidade de simulação passo a passo, semelhante a Lisa.

O Java Formal Languages and Automata Package (JFLAP) também foi uma dos sistemas pioneiros, representando um marco no uso de *software* educacional para o ensino de linguagens formais e autômatos. Este *software* oferece um ambiente visual e interativo que facilita a compreensão de conceitos abstratos. Uma das inovações centrais do JFLAP é permitir que os alunos construam, simulem e transformem estruturas formais com *feedback* imediato, algo que os métodos tradicionais não permitem. Como destacado pelos autores, “os alunos podem verificar suas respostas e interagir com esses autômatos” (Rodger; Finley, 2006).

Um estudo realizado com estudantes de ciência da computação mostrou que o uso do JFLAP contribuiu significativamente para o entendimento de conceitos teóricos, promovendo maior engajamento e melhor desempenho em avaliações (Silva *et al.*, 2023). A aplicação *web* desenvolvida neste trabalho segue a linha de pensamento do JFLAP ao oferecer interatividade, porém diferencia-se ao focar especificamente na linguagem Simples, execução passo a passo da MVS (Máquina Virtual Simples) e visualização da árvore sintática e da árvore de derivação, funcionalidades não presentes no JFLAP.

Hiago Borges de Oliveira desenvolveu o Ambiente de Desenvolvimento Integrado Simples (ADIS) (Oliveira, 2014), que permite estudar o processo de compilação e execução da linguagem Simples, incluindo a execução passo a passo do código. Diferentemente desse trabalho, o JSimples se destaca por portar essas funcionalidades para o ambiente *web*, além de oferecer a visualização da árvore sintática e da árvore de derivação da linguagem Simples.

Além disso, Arnaiz-González et al. (Arnaiz-González *et al.*, 2018) desenvolveram o Seshat, uma plataforma web que complementa o JFLAP ao focar na análise léxica e na conversão de expressões regulares em autômatos finitos. Ao contrário do Seshat, o JSimples foca especialmente na demonstração da análise sintática e execução passo a passo da linguagem Simples.

Uma análise mais recente realizada por Stamenković, Jovanović e Chakraborty (Stamenković; Jovanović; Chakraborty, 2020), que avaliou 16 ferramentas de simulação voltadas ao ensino de construção de compiladores, identificou que, embora ferramentas como o JFLAP e Lisa se destaquem pela interatividade, ambas carecem de funcionalidades de simulação para a etapa de análise sintática (Stamenković; Jovanović; Chakraborty, 2020). As ferramentas existentes, por sua vez, não satisfazem às necessidades de cursos introdutórios. Nesse contexto, este trabalho desenvolveu uma plataforma web focada na compreensão de conceitos sintáticos e execução passo a passo da linguagem Simples, com o objetivo de complementar as lacunas encontradas nas ferramentas já existentes.

Stamenković e Jovanović (Stamenković; Jovanović, 2023), com base nas lacunas funcionais identificadas no estudo de Stamenković, Jovanović e Chakraborty (Stamenković; Jovanović; Chakraborty, 2020), desenvolveram o ComVis: uma plataforma *web* destinada a simular os principais tópicos da disciplina de teoria de linguagens e compiladores. ComVis permite aos alunos explorar diferentes etapas e conceitos teóricos do processo de compilação, como autômatos finitos determinísticos, análise léxica, sintática e semântica. Embora o ComVis apresente maior abrangência de tópicos, o JSimples diferencia-se ao oferecer uma experiência centrada na linguagem Simples, permitindo que estudantes brasileiros explorem compilação com uma linguagem didática em português.

A análise das ferramentas educacionais revela três lacunas principais que motivam o desenvolvimento do JSimples: (1) Lacuna de acessibilidade - ferramentas robustas como ADIS e Lisa são limitadas a ambientes *desktop*, exigindo instalação e restringindo acesso; (2) Lacuna de visualização sintática - nenhuma ferramenta *web* atual oferece visualização interativa de

árvores sintáticas combinada com execução passo a passo; (3) Lacuna contextual - ausência de ferramentas *web* em português que integrem teoria e prática de compilação. A Tabela 1 sintetiza as principais características das ferramentas educacionais discutidas em comparação com o JSimples.

Tabela 1 – Comparação entre ferramentas educacionais

Ferramenta	**Plataforma**	**Execução passo a passo**	**Linguagem específica**	**Visualização da árvore**
JFLAP	*Desktop*	✓	×	×
Lisa	*Desktop*	✓	×	×
ADIS	*Desktop*	✓	✓	×
Seshat	*Web*	×	×	×
ComVis	*Web*	×	×	✓
JSimples	***Web***	✓	✓	✓

Fonte: Autor do texto (2025).

De acordo com a Tabela 1, pode se observar que o JSimples é a única ferramenta que combina plataforma web, execução passo a passo, foco em linguagem específica didática e visualização de árvore sintática, preenchendo lacunas identificadas nas soluções existentes.

Por fim, a revisão bibliográfica evidencia uma evolução de ferramentas *desktop* especializadas (JFLAP, Lisa, ADIS) para soluções *web* multiplataforma (ComVis, Seshat), porém com persistência de lacunas funcionais específicas. O JSimples posiciona-se como síntese dessa evolução, combinando acessibilidade *web*, profundidade pedagógica centrada na linguagem Simples, árvore sintática e de derivação e contextualização para o ensino brasileiro.

3 AMBIENTE DE DESENVOLVIMENTO INTEGRADO SIMPLES (ADIS)

3.1 LINGUAGEM SIMPLES

A linguagem Simples (Oliveira, 2014) é uma linguagem de programação didática desenvolvida por Hiago Borges de Oliveira, criada para facilitar o ensino de conceitos fundamentais de programação, compiladores e arquitetura de computadores. O processo de execução da linguagem segue um fluxo bem definido, onde o código-fonte escrito em linguagem Simples é traduzido pelo Compilador Simples para instruções MVS (Máquina Virtual Simples), que são interpretadas e executadas pela máquina virtual, gerando o resultado final.

Esta linguagem foi projetada com comandos em português, tornando-a mais intuitiva para estudantes brasileiros, e inclui funcionalidades essenciais como declaração de variáveis (tipos inteiro e lógico), comandos de entrada e saída (leia e escreva), estruturas condicionais (se-então-senão), laços de repetição (enquanto-faça), funções e procedimentos (com passagem de parâmetros de valores ou referência a variáveis), e suporte a expressões aritméticas, relacionais e lógicas. Esta abordagem simplificada permite que os alunos compreendam de forma prática a relação entre uma linguagem de alto nível, o processo de compilação e a execução em nível de máquina.

3.2 DEFINIÇÃO DA MÁQUINA VIRTUAL SIMPLES (MVS)

A MVS é um projeto de máquina virtual semelhante a Máquina para Execução Pascal (Kowaltowski, 1983), sendo uma máquina de pilha, característica que proporciona um ambiente ideal para programas com recursividade e estruturas aninhadas.

Essa arquitetura de pilha reflete-se na organização da memória, que é dividida em duas regiões: o Vetor P armazena as instruções do programa, e a Pilha M armazena os valores manipuláveis durante a execução. Essas regiões trabalham de forma integrada, onde as instruções dispostas no Vetor P operam sobre os valores armazenados na Pilha M.

Para controlar a execução e o acesso a essas regiões de memória, a MVS possui três registradores essenciais: o registrador I aponta para a próxima instrução a ser executada no Vetor P; o registrador S indica o topo da Pilha M; e o registrador D aponta para a base do escopo, sendo útil em gerenciamento de escopos em chamadas de funções e procedimentos.

No que se refere ao conjunto de instruções, a MVS é composta por 31 palavras-chave mnemônicas, projetadas para tornar a operação realizada por cada instrução explícita, facilitando a compreensão, depuração de programas compilados para essa arquitetura e o entendimento de cada instrução MVS. Todas as 31 instruções são explicadas na 2.

Tabela 2 – Instruções da Máquina Virtual Simples (MVS)

Instrução	**Descrição**
Controle de Execução	
`INPP`	Inicia a execução programa principal.
`FIMP`	Finaliza a execução do programa.
`NADA`	Não realiza nenhuma operação. Útil para instruções de desvio.
Gerenciamento de Memória	
`AMEM` *k*	Aloca *k* espaços na memória.
`DMEM` *n*	Desaloca *n* posições da memória.
Carregamento de Valores	
`CRCT` *k*	Carrega o valor constante *k* no topo da Pilha M.
`CRVG` *n*	Carrega o valor da variável global armazenada no endereço *n* no topo da Pilha M.
`CRVL` *n*	Carrega o valor da variável local da posição *n* da Pilha M.
`CRVI` *n*	Carrega o valor armazenado no endereço contido na posição *n* da Pilha M.
Armazenamento de Valores	
`ARZG` *n*	Armazena o valor do topo da Pilha M no endereço global *n*.
`ARZL` *n*	Armazena o valor do topo da Pilha M na posição *n* da Pilha M.
`ARMI` *n*	Armazena o valor do topo da Pilha M no endereço contido na posição *n* da Pilha M.
Carregamento de Endereços	
`CREL` *n*	Carrega o endereço local *n* no topo da Pilha M.
`CREG` *n*	Carrega o endereço global *n* no topo da Pilha M.
Desvios	
`DSVS` *p*	Altera incondicionalmente o ponteiro I para o valor *p*.
`DSVF` *p*	Caso o valor no topo da Pilha M seja falso, altera o ponteiro I para o valor *p*.

continua na próxima página

Tabela 2 – Instruções da Máquina Virtual Simples (MVS)

Instrução	Descrição
Entrada e Saída	
`LEIA`	Lê um valor no console e o armazena no topo da Pilha M.
`ESCR`	Escreve no console o valor contido no topo da Pilha M.
Operações de Comparação	
`CMME`	Compara se o valor no topo da Pilha M -1 é menor que o valor no topo da Pilha M e empilha o resultado.
`CMMA`	Compara se o valor no topo da Pilha M -1 é maior que o valor no topo da Pilha M e empilha o resultado.
`CMIG`	Compara se o valor no topo da Pilha M -1 é igual ao valor no topo da Pilha M e empilha o resultado.
Operações Lógicas	
`DISJ`	Realiza operação de disjunção entre os dois elementos no topo da Pilha M e empilha o resultado.
`CONJ`	Realiza operação de conjunção entre os dois elementos no topo da Pilha M e empilha o resultado.
`NEGA`	Realiza a operação de negação no valor do topo da Pilha M e empilha o resultado.
Operações Aritméticas	
`SOMA`	Soma os dois elementos no topo da Pilha M e empilha o resultado.
`SUBT`	Subtrai o topo da Pilha M -1 pelo valor no topo da Pilha M e empilha o resultado.
`MULT`	Multiplica os dois elementos no topo da Pilha M e empilha o resultado.
`DIVI`	Divide o topo da Pilha M -1 pelo valor no topo da Pilha M e empilha o resultado.
Subprogramas	
`SVCP`	Empilha o contador de programa no topo da Pilha M.
`ENSP`	Indica a entrada de um subprograma.
`RTSP` n	Retorno do subprograma para o endereço n no Vetor P.

Fonte: Autor do texto (2025).

3.3 PORTABILIDADE DO ADIS AO AMBIENTE *WEB*

A ferramenta didática desenvolvida em Java por Hiago Borges para o ensino de lógica de programação, compiladores e arquitetura de computadores, utilizando linguagem Simples, é limitada apenas a ambientes *desktop*, não havendo portabilidade para outros dispositivos (Oliveira, 2014). A necessidade da instalação da Java Virtual Machine também se torna um problema, especialmente em dispositivos com recursos limitados ou políticas restritivas de instalação de software. Essas limitações motivam o desenvolvimento de uma solução *web* mais acessível e multiplataforma.

Nesse contexto, o JSimples foi portado para o ambiente *web*, o que elimina a necessidade de instalação local e possibilita a integração com plataformas de ensino por meio do padrão Sharable Content Object Reference Model (SCORM) [1]. Isso torna o acesso a ferramenta mais acessível e multiplataforma, alinhando-se com as observações feitas por Stamenković, Jovanović e Chakraborty (Stamenković; Jovanović; Chakraborty, 2020).

[1] Padrão internacional que permite integrar conteúdos educacionais em plataformas de ensino como Moodle e Canvas, facilitando a distribuição e o acesso centralizado de aplicativos educacionais.

4 METODOLOGIA DE DESENVOLVIMENTO

4.1 TECNOLOGIAS E ARQUITETURA

O desenvolvimento do JSimples baseou-se em uma abordagem *client-side*, ou seja, sem a necessidade de um *back-end* tradicional. Utilizamos tecnologias *web* atuais que garantem a execução direta no navegador sem dependências externas, garantindo uma ferramenta multi-plataforma. A escolha do JavaScript como linguagem principal fundamentou-se em sua ampla adoção no desenvolvimento *web* e compatibilidade com navegadores modernos. HTML5 e CSS3 complementam a base tecnológica, fornecendo a base semântica e visual da aplicação.

4.2 FRAMEWORKS E BIBLIOTECAS

Para otimizar o desenvolvimento do código, adotou-se o React (Meta Platforms, Inc., 2025) como *framework* principal. Esta escolha fundamentou-se na capacidade de componentização modular da interface, permitindo encapsulamento da lógica de negócio. O React também oferece um ciclo de vida bem definido para o gerenciamento de estado, facilitando o controle das interações necessárias em um ambiente de execução passo a passo interativo.

Para componentes de interface, utilizamos a biblioteca Ant Design (Ant UED, 2015), que fornece componentes React prontos. Esta escolha garante padronização visual através de um *design system* consistente e acelera o desenvolvimento ao disponibilizar componentes básicos de interface já implementados. A responsividade nativa dos componentes também assegura uma experiência adequada em diferentes tamanhos de tela.

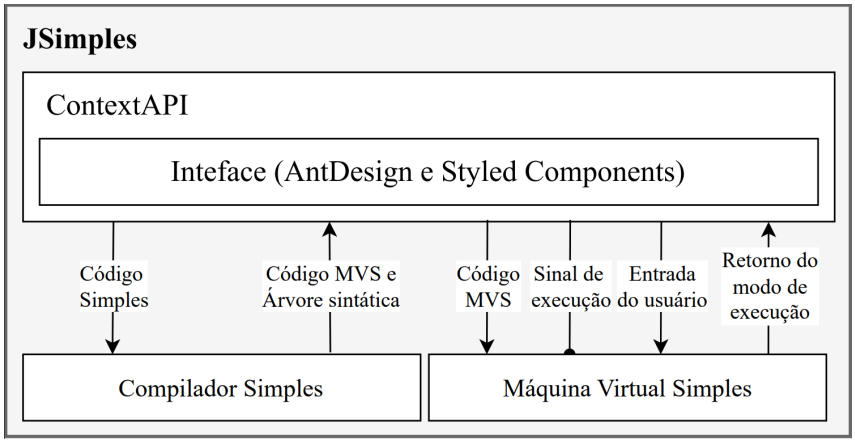
A biblioteca *Styled Components* (Maddern; Stoiber, 2016) permite a estilização do JSimples, que proporciona uma abordagem CSS-in-JS com escopo isolado. Esta solução elimina conflitos de nomenclatura CSS comuns de aplicações, além de melhorar a organização de estilos, aspectos fundamentais para a manutenibilidade do projeto.

4.2.1 Gerenciamento de Estados

Para o gerenciamento centralizado de estado da aplicação, foi adotada a Context API nativa do React. Esta solução permite o compartilhamento de dados entre os módulos principais: Compilador Simples, Máquina Virtual Simples e Interface.

A Context API gerencia diversos estados: código-fonte, código MVS, árvore sintática e de derivação, sinais de execução, entradas do usuário e retornos da execução. Esta arquitetura garante sincronização consistente, propagando alterações automaticamente entre componentes e refletindo interações em tempo real. A Figura 1 ilustra a organização da arquitetura do JSimples, evidenciando o fluxo de dados entre os módulos e o papel central da Context API na integração dos componentes.

Figura 1 – Arquitetura do JSimples



Fonte: Autor do texto (2025).

4.2.2 Implementação do editor de código

Para o editor de código, foi integrada a biblioteca CodeMirror (Haverbeke, 2018), utilizando um *wrapper* para o React. A fim de adaptar o editor às especificidades da linguagem Simples, desenvolveu-se um extensão adaptada a sintaxe da linguagem Simples, que implementa realce de sintaxe específico para a sintaxe da linguagem, como demonstrado na Figura 2. Esta personalização melhora a experiência do usuário ao proporcionar *feedback* visual imediato sobre palavras-chave utilizadas no programa, facilitando a identificação da estrutura e a compreensão geral do programa.

Figura 2 – Implementação do realce de sintaxe no editor CodeMirror

```
1  programa teste
2      inteiro a b
3      logico c d
4  inicio
5      a <- 1
6      a <- a * 3
7      escreva a
8  fimprograma
```

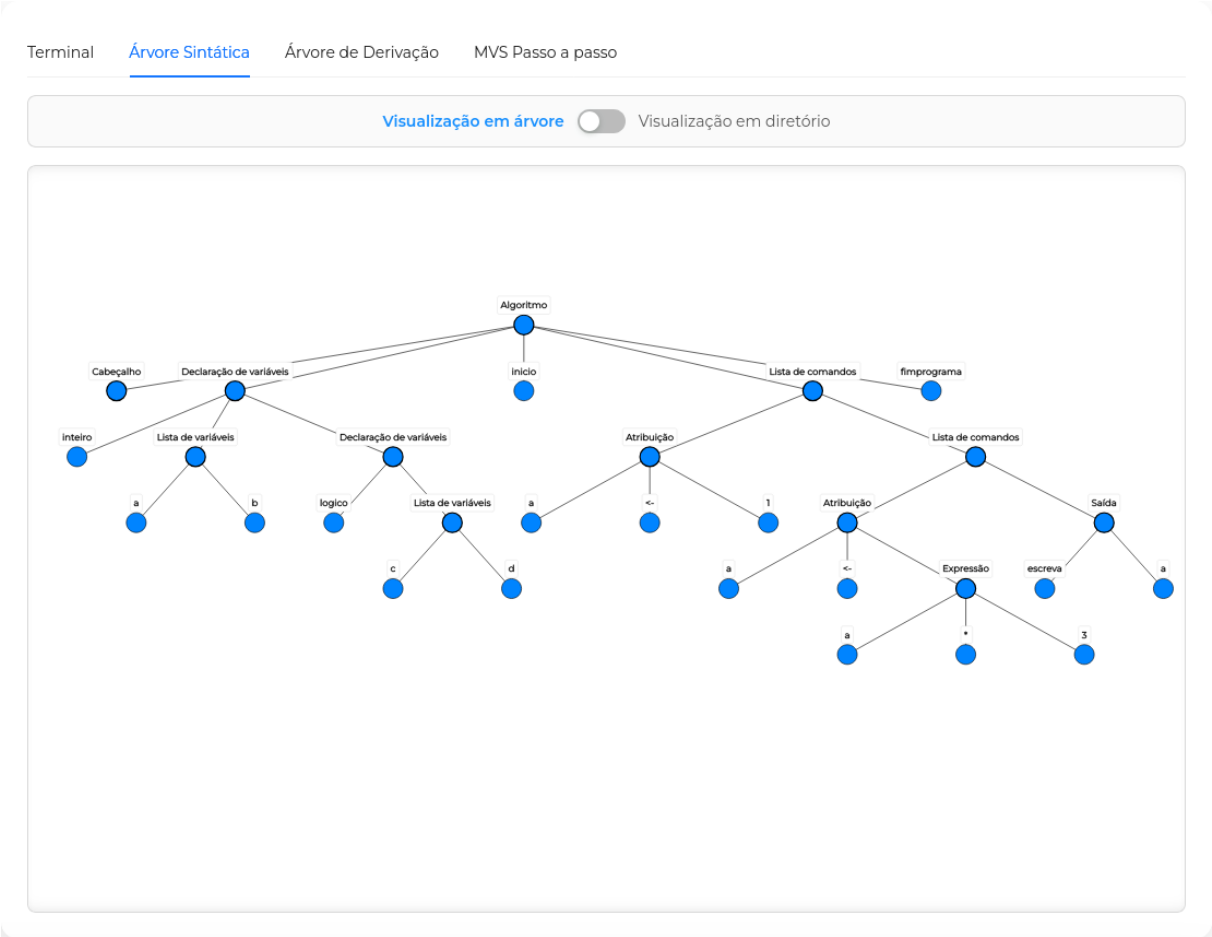
Fonte: Autor do texto (2025).

4.2.3 Implementação da representação da árvore sintática e de derivação

Para a implementação da árvore sintática e da árvore de derivação, foi utilizada a biblioteca React D3 Tree (Kremer, 2017), que permite a construção de árvores hierárquicas interativas através da integração entre React e D3.js. Esta biblioteca oferece renderização eficiente de estruturas arbóreas com suporte nativo a *zoom* e navegação interativa, características essenciais para visualizar árvores sintáticas e árvores de derivação com múltiplos níveis.

A representação visual da árvore sintática e da árvore de derivação gerada pelo compilador possibilita ao estudante compreender como o código-fonte em linguagem Simples é estruturado hierarquicamente durante o processo de análise sintática. Cada nó da árvore corresponde a uma construção sintática da linguagem, como expressões, comandos de controle ou declarações de variáveis, conforme ilustrado na Figura 3. A interatividade proporcionada pela biblioteca permite expandir e colapsar subárvores, facilitando a análise de trechos específicos do programa sem perder o contexto da estrutura global.

Figura 3 – Árvore sintática da gramática Simples gerada pelo JSimples



Fonte: Autor do texto (2025).

4.2.4 Implementação da Tabela de Símbolos

A tabela de símbolos é uma estrutura que visualiza como as variáveis são identificadas pelo compilador em tempo de compilação, exibindo informações como tipo, identificador, endereço, escopo, rótulo, categoria e parâmetros (para funções e procedimentos) conforme ilustrado pela Figura 4. Desenvolvemos a interface utilizando componentes da biblioteca Ant Design, com adaptações para aumentar a responsividade em dispositivos móveis.

Figura 4 – Tabela de símbolos gerada pela compilação de um código Simples

	Tipo	Identificador	Endereço	Escopo	Rótulo	Categoria	Parâmetros
	Inteiro	a	0	GLOBAL	-	Variável	-
	Inteiro	b	1	GLOBAL	-	Variável	-
	Lógico	c	2	GLOBAL	-	Variável	-
	Lógico	d	3	GLOBAL	-	Variável	-

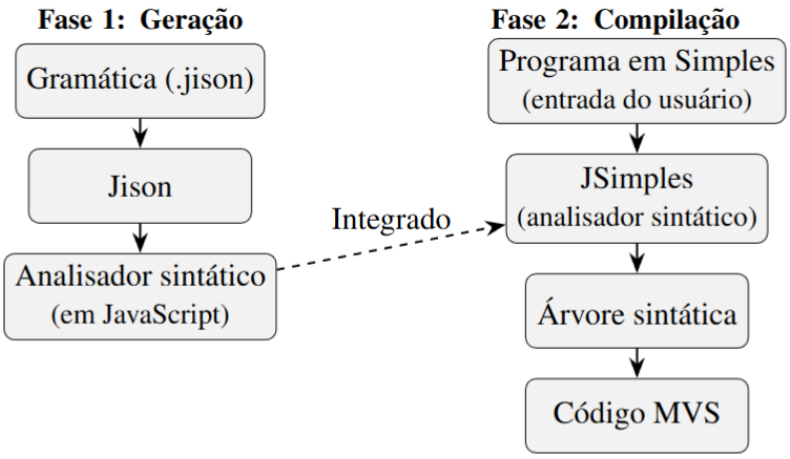
Fonte: Autor do texto (2025).

4.3 IMPLEMENTAÇÃO DO COMPILADOR

Desenvolvemos o compilador do JSimples utilizando Jison (Zach Carter, 2009), um gerador de analisadores sintáticos inspirado no Bison (Free Software Foundation, 2025) e Yacc (Johnson, 1975). A escolha desta ferramenta baseou-se em sua integração nativa com aplicações *web*, possibilitando a execução direta em ambiente de navegador sem dependências externas.

Para sua utilização, criamos uma especificação de gramática em formato `.jison`, a partir da qual o Jison gera automaticamente um analisador sintático em JavaScript. Este analisador é então integrado ao JSimples, onde processa a entrada do usuário e produz o código MVS através da construção da árvore sintática, como demonstrado na Figura 5.

Figura 5 – Processo de geração do analisador sintático do JSimples



Fonte: Autor do texto (2025).

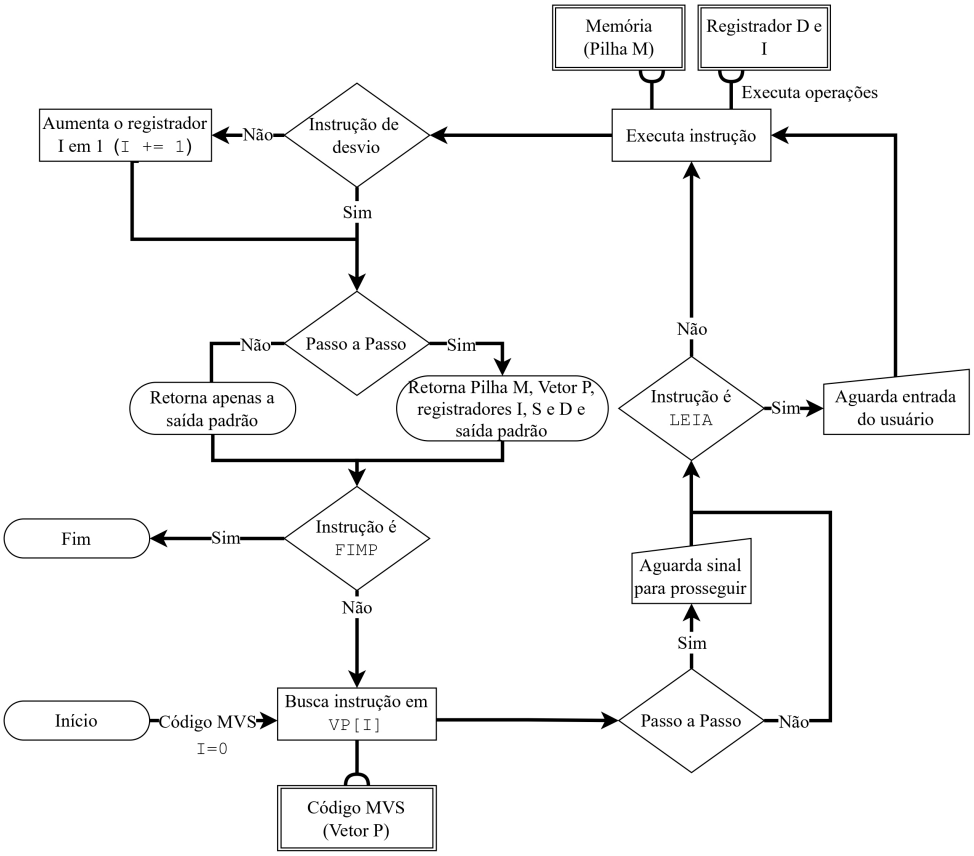
Vale ressaltar que o Jison apresenta uma limitação em relação ao Bison/Yacc tradicional: a ausência de suporte para *mid-rule actions* [1]. Esta restrição exigiu a reestruturação da gramática original da linguagem Simples, utilizando regras auxiliares menores para garantir que ações semânticas fossem executadas apenas ao final de cada regra auxiliar. Apesar do aumento no número de regras da gramática, esta abordagem manteve o resultado final esperado.

4.4 IMPLEMENTAÇÃO DA MÁQUINA VIRTUAL SIMPLES (MVS)

Implementamos a MVS em JavaScript seguindo a especificação original (Oliveira, 2014), mantendo fidelidade à arquitetura de pilha e ao conjunto de 31 instruções.

O ciclo de execução segue o modelo busca-decodificação-execução tradicional: busca da instrução no Vetor P, identificação do tipo de operação e execução sobre a Pilha M, repetindo-se até encontrar a instrução `FIMP`. A Figura 6 ilustra o funcionamento da MVS.

Figura 6 – Funcionamento da Máquina Virtual Simples (MVS)



Fonte: Autor do texto (2025).

[1] *Mid-rule actions* são blocos de código executados durante o reconhecimento de uma regra de produção, permitindo processamento semântico incremental antes que toda a regra seja derivada.

O fluxograma apresentado ilustra o ciclo de execução da Máquina Virtual Simples (MVS), que opera sobre três estruturas principais: o Vetor P (código MVS), a Pilha M (memória), e os registradores I, S e D. O processo inicia com I=0 e segue um ciclo de busca-decodificação-execução, onde cada iteração busca a instrução em VP[I] e a executa de acordo com sua natureza.

Durante a execução, o sistema verifica se o modo passo a passo está ativo. Quando habilitado, este modo retorna o estado completo da MVS (Pilha M, Vetor P, registradores I, S e D) após cada instrução, permitindo visualização detalhada da execução. Caso contrário, retorna apenas a saída padrão do programa.

A execução de cada instrução envolve decisões baseadas em seu tipo. Instruções de desvio alteram o fluxo modificando o registrador I, enquanto instruções de entrada (LEIA) aguardam dados do usuário. Instruções regulares são executadas diretamente, manipulando a Pilha M e os registradores conforme sua especificação. Após cada execução, exceto em casos de desvio, o registrador I é incrementado em 1 ($I+=1$).

O ciclo continua até encontrar a instrução FIMP, que sinaliza o término do programa. Este modelo de execução baseado em máquina de pilha permite a implementação de operações aritméticas, lógicas e de controle de fluxo, constituindo a base para a interpretação da linguagem Simples no ambiente JSimples.

4.4.1 Execução Assíncrona e Controle de Fluxo

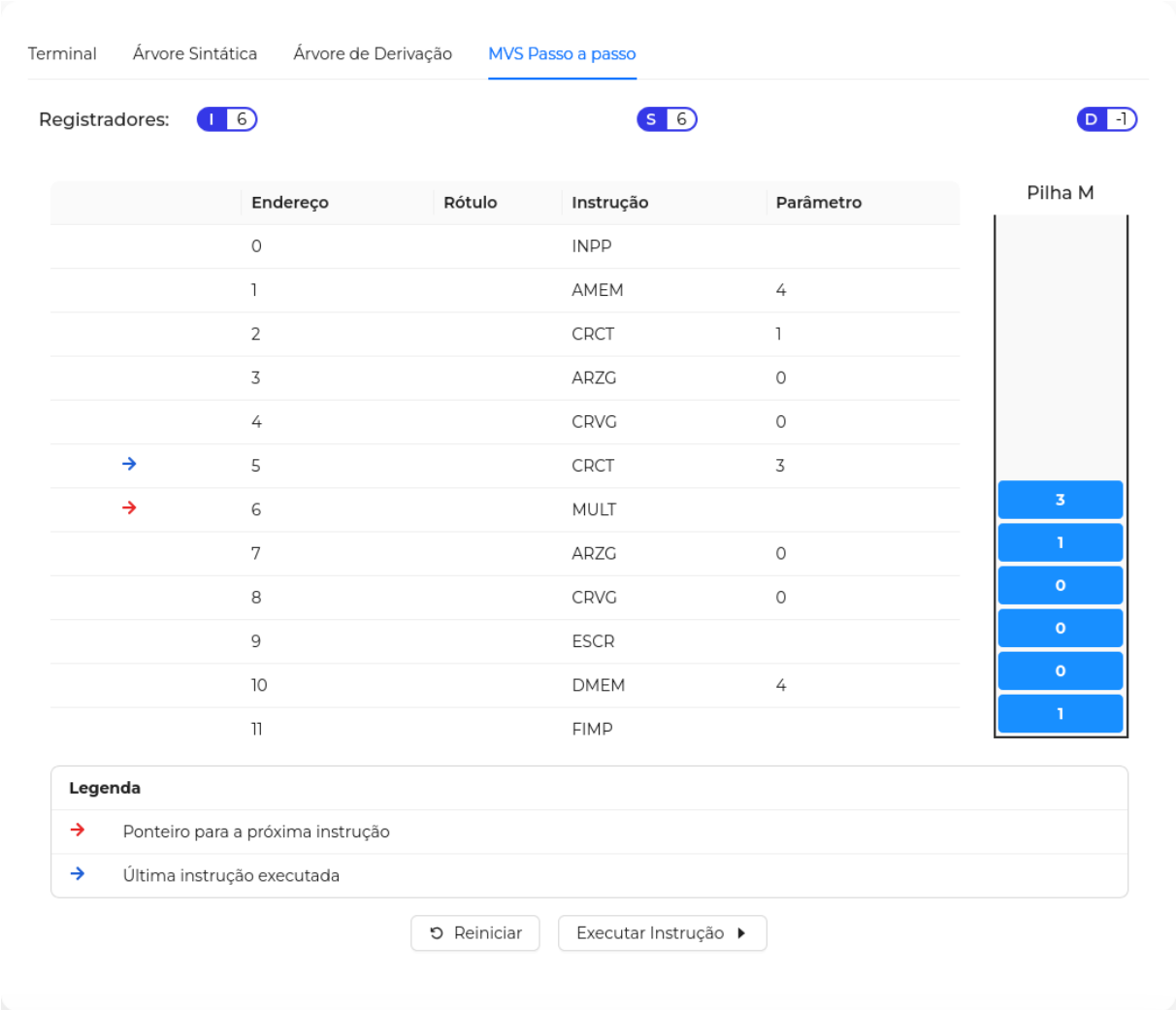
A execução é gerenciada de forma assíncrona e não-bloqueante, permitindo que a interface permaneça responsiva mesmo durante a execução de iterações extensas. O controle de fluxo permite pausar, retomar ou interromper a execução conforme necessário.

Para operações de entrada, foi implementado um sistema assíncrono que suspende temporariamente a execução, aguarda a entrada do usuário via interface gráfica, e retoma o processamento após a captura do valor. Esta abordagem garante que a aplicação não bloqueie enquanto aguarda a interação do usuário.

4.4.2 Modo de Execução Passo a Passo

A MVS suporta dois modos de execução: contínua e passo a passo. O modo passo a passo permite que o usuário avance a execução uma instrução por vez. Após cada instrução, a execução é pausada até que o usuário decida prosseguir. A cada avanço, o estado completo da máquina (registradores I, S e D, conteúdo da pilha, tabela de instrução MVS e o indicador da próxima instrução) é exibido na interface, permitindo o usuário acompanhar o estado da máquina em tempo real. O modo passo a passo é mostrado na Figura 7.

Figura 7 – Inteface do modo passo a passo do JSimples



Fonte: Autor do texto (2025).

4.5 TESTES E VALIDAÇÃO

A validação da aplicação seguiu uma metodologia contemplando testes de compatibilidade em múltiplos navegadores em suas versões mais recentes. A responsividade da aplicação também foi validada em resoluções variadas, desde telas de *320px* de largura até monitores *desktop* com resolução de *1920px*. Esta abordagem assegurou que a interface permanecesse funcional em diferentes contextos de uso, garantindo o acesso multiplataforma do aplicativo.

5 AVALIAÇÃO DA PLATAFORMA

A avaliação da plataforma JSimples combinou *frameworks* complementares de avaliação. Esta estratégia permite analisar tanto aspectos técnicos de usabilidade quanto dimensões pedagógicas específicas de *software* educacional.

5.1 METODOLOGIA DE AVALIAÇÃO

A avaliação de usabilidade é fundamentada nas 10 heurísticas de Nielsen (Nielsen; Molich, 1990), focando em princípios gerais de *design* de interface e interação humano-computador. Por outro lado, a avaliação pedagógica é baseada no instrumento Learning Object Review Instrument (LORI) (Vargo J.C. Nesbit; Archambault, 2003), específico para objetos de aprendizagem.

Esta abordagem é justificada pela natureza híbrida do JSimples, que se trata simultaneamente de uma aplicação *web* que deve ser usável (domínio de interface humano-computador) e de uma ferramenta educacional que deve promover aprendizagem efetiva (domínio pedagógico). A análise isolada de apenas uma dimensão seria insuficiente para avaliar adequadamente uma ferramenta didática interativa.

5.1.1 Protocolo de Avaliação

A avaliação foi conduzida pelos autores deste trabalho, adotando perspectiva de especialista com conhecimento em desenvolvimento de *software* e teoria de linguagens e compiladores. Para cada *framework*, seguimos os seguintes procedimentos:

Heurísticas de Nielsen: Navegação sistemática pela interface exercitando todas as funcionalidades principais (compilação, execução contínua, execução passo a passo, visualização de árvore sintática, visualização da árvore de derivação e visualização da tabela de símbolos).

LORI: Análise individual de cada dimensão pedagógica, atribuindo pontuação fundamentada em evidências observadas na ferramenta, seguindo escala Likert de 5 pontos conforme recomendado pelos autores.

5.2 AVALIAÇÃO DE USABILIDADE (HEURÍSTICAS DE NIELSEN)

As heurísticas de usabilidade de Nielsen constituem dez princípios gerais para *design* de interface que permitem identificar problemas de usabilidade através de inspeção sistemática. As heurísticas avaliadas são:

1. **Visibilidade do status do sistema:** O sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de *feedback* em tempo real.

2. **Correspondência entre sistema e mundo real:** O sistema deve falar a linguagem dos usuários, com palavras, frases e conceitos familiares, ao invés de termos técnicos.

3. **Controle e liberdade do usuário:** Usuários frequentemente escolhem funções por engano e precisam de uma "saída de emergência"claramente marcada.

4. **Consistência e padrões:** Usuários não devem ter que se perguntar se palavras, situações ou ações diferentes significam a mesma coisa.

5. **Prevenção de erros:** Ainda melhor que boas mensagens de erro é um *design* cuidadoso que previne problemas.

6. **Reconhecimento em vez de memorização:** Minimizar a carga de memória do usuário tornando objetos, ações e opções visíveis.

7. **Flexibilidade e eficiência de uso:** Aceleradores invisíveis para usuários novatos podem aumentar a velocidade de interação para especialistas.

8. ***Design* estético e minimalista:** Diálogos não devem conter informação irrelevante ou raramente necessária.

9. **Ajudar usuários a reconhecer, diagnosticar e recuperar erros:** Mensagens de erro devem ser expressas em linguagem simples, indicar o problema precisamente e sugerir soluções.

10. **Ajuda e documentação:** Embora seja melhor que o sistema possa ser usado sem documentação, pode ser necessário fornecer ajuda e documentação.

A aplicação de cada heurística ao JSimples é detalhada a seguir.

5.2.1 Pontos Fortes Identificados

A avaliação revelou conformidade plena em aspectos fundamentais de usabilidade, conforme detalhado a seguir:

H1 - Visibilidade do Status do Sistema (5/5): A plataforma fornece *feedback* imediato durante o processo de compilação e execução. Durante compilação, erros são exibidos instantaneamente. No modo passo a passo (Figura 6), a instrução MVS atual é destacada visualmente e o estado completo da máquina (registradores I, S, D e conteúdo da Pilha M) é atualizado de forma síncrona, permitindo ao estudante acompanhar cada transição de estado.

H2 - Correspondência entre Sistema e Mundo Real (5/5): A utilização de comandos em português na linguagem Simples reduz a barreira linguística comparado a ferramentas em inglês como JFLAP e ComVis. Esta escolha de *design* permite que estudantes brasileiros concentrem esforços cognitivos nos conceitos de compilação ao invés de tradução mental de termos estrangeiros. A interface também emprega terminologia educacional familiar, alinhando-se com vocabulário usado em sala de aula.

H4 - Consistência e Padrões (5/5): A adoção da biblioteca Ant Design garante padrões visuais e comportamentais consistentes em toda a interface. Botões com funções similares mantêm posicionamento, cor e estilo uniformes. A organização dos seis painéis principais (editor de código, console, árvore sintática, árvore de derivação, execução passo a passo e tabela de símbolos) permanece constante durante toda a interação, facilitando a navegação e reduzindo carga cognitiva.

H8 - *Design* Estético e Minimalista (5/5): A interface mantém foco em elementos essenciais para o fluxo de trabalho: escrita de código, visualização de estrutura sintática e acompanhamento de execução. Não há elementos decorativos supérfluos, propagandas ou funcionalidades secundárias que distraiam o estudante da tarefa principal. O *layout* de seis painéis otimiza uso do espaço de tela sem sobrecarregar visualmente o usuário.

H9 - Ajudar a Recuperar de Erros (5/5): As mensagens de erro de compilação são específicas, indicando linha e coluna do problema, tipo do erro (léxico, sintático, semântico) e descrição clara em português. Por exemplo, ao detectar declaração duplicada de variável, o sistema informa a posição e o símbolo já declarado, permitindo localização e correção rápida. Esta clareza é particularmente importante para estudantes iniciantes, que se beneficiam de *feedback* pedagógico ao invés de mensagens técnicas.

5.2.2 Oportunidades de Melhoria

Cinco heurísticas apresentaram conformidade parcial, indicando aspectos que, embora não comprometam a funcionalidade essencial, poderiam ser aprimorados:

H3 - Controle e Liberdade do Usuário (3/5): A plataforma permite pausar, retomar e interromper a execução através de botões. Entretanto, a ausência de funcionalidade para "voltar passo"limita o controle total do usuário sobre a navegação durante execução passo a passo. Atualmente, para reanalisar um trecho específico já executado, o estudante deve reiniciar a execução completa desde o início. Implementar histórico navegável de estados da MVS, permitindo retroceder para qualquer ponto anterior da execução. Esta funcionalidade facilitaria análise comparativa de estados e depuração de lógica.

H5 - Prevenção de Erros (3/5): A plataforma implementa múltiplos mecanismos preventivos, como realce de sintaxe (Figura 2), que destaca palavras-chave da linguagem Simples em cores distintas, permitindo identificação visual imediata de comandos válidos. Estas características reduzem erros básicos de sintaxe e uso inadequado da interface. Entretanto, o editor não implementa análise estática em tempo real (*linting*), ou seja, erros sintáticos só são detectados após tentativa de compilação, ao invés de serem sinalizados visualmente durante digitação. Esta limitação reduz a prevenção proativa de erros básicos.

H6 - Reconhecimento em vez de Memorização (3/5): A árvore sintática e de derivação apresentam rótulos para cada nó (comandos, expressões, declarações), e o realce de sintaxe facilita reconhecimento visual de estruturas no código-fonte. Porém, o painel de execução MVS exibe apenas mnemônicos das instruções sem explicação contextual. Estudantes em primeiro contato com arquitetura de pilha podem não reconhecer imediatamente o significado de cada instrução. Para melhorar a compreensão de novos estudantes, devem ser implementadas dicas explicativas que exibam breve descrição sobre a implicação de uma instrução MVS sobre o estado da máquina.

H7 - Flexibilidade e Eficiência de Uso (4/5): O JSimples oferece dois modos de execução adequados a diferentes necessidades: execução contínua para visualização rápida de resultado final, e execução passo a passo para análise detalhada de cada instrução. Esta dualidade atende tanto usuários novatos quanto experientes. Entretanto, a ausência de atalhos de teclado força uso exclusivo do *mouse*, reduzindo eficiência para usuários avançados.

H10 - Ajuda e Documentação (2/5): A plataforma não oferece documentação sobre o código MVS e a sintaxe da linguagem Simples. Não há tutorial interativo de primeiro uso, ajuda contextual, ou exemplos de código integrados à interface. Estudantes sem familiaridade prévia podem enfrentar curva de aprendizado inicial em relação ao uso da linguagem Simples.

5.2.3 Síntese da Avaliação de Usabilidade

A análise heurística resultou em média de 4/5 para usabilidade. Os pontos fortes concentram-se em aspectos fundamentais: *feedback* constante (H1), linguagem adequada ao público brasileiro (H2) e mensagens de erro pedagógicas (H9). A consistência visual (H4) e o minimalismo (H8) contribuem para interface limpa que não sobrecarrega estudantes durante processo de aprendizagem.

As oportunidades de melhoria identificadas (H3, H5, H6, H7, H10) referem-se predominantemente a funcionalidades que, embora desejáveis para otimizar experiência de usuários, não impedem o uso efetivo da ferramenta por seu público-alvo principal. A ausência de navegação reversa (H3) e atalhos de teclado (H7) representa limitação para cenários avançados de depuração, mas não compromete o fluxo básico de aprendizagem. Similarmente, dicas explicativos (H6) e tutorial interativo (H10) aprimorariam curva de aprendizado inicial, mas não são impeditivos. A Tabela 3 sintetiza todos os tópicos discutidos acima.

Tabela 3 – Avaliação heurística do JSimples segundo Nielsen

Heurística	**Nota**	**Observação**
H1: Visibilidade do *status* do sistema	5	*Feedback* em tempo real, atualização síncrona
H2: Correspondência mundo real	5	Linguagem Simples com comandos em português
H3: Controle e liberdade do usuário	3	Ausência de navegação reversa (voltar passo)
H4: Consistência e padrões	5	Biblioteca Ant Design garante uniformidade
H5: Prevenção de erros	3	Ausência de *linting* no editor de código
H6: Reconhecimento vs memorização	3	Faltam *tooltips* sobre instruções MVS
H7: Flexibilidade e eficiência de uso	4	Ausência de atalhos de teclado
H8: *Design* estético e minimalista	5	Interface limpa focada em essenciais
H9: Recuperação de erros	5	Mensagens específicas com linha e coluna
H10: Ajuda e documentação	2	Documentação básica, pode expandir
Média Geral	**4**	

Fonte: Autor do texto (2025).

5.3 AVALIAÇÃO PEDAGÓGICA (LORI)

O instrumento LORI é um *framework* estabelecido para avaliação de objetos de aprendizagem. Este instrumento avalia nove dimensões pedagógicas em escala Likert de 1 a 5 pontos, onde 1 representa qualidade inadequada e 5 representa qualidade excelente. As dimensões avaliadas são:

1. **Qualidade do Conteúdo:** Veracidade, precisão, apresentação equilibrada de ideias e nível apropriado de detalhe.
2. **Alinhamento com Objetivos de Aprendizagem:** Correspondência entre objetivos, atividades, avaliações e características dos aprendizes.
3. ***Feedback* e Adaptação:** Capacidade de adaptar-se às respostas do aprendiz e fornecer *feedback* significativo.
4. **Motivação:** Capacidade de motivar a população-alvo identificada.
5. ***Design* de Apresentação:** Qualidade do *design* da informação visual para facilitar processamento cognitivo.
6. **Usabilidade da Interação:** Facilidade de navegação, previsibilidade da interface e qualidade dos recursos de ajuda.
7. **Acessibilidade:** *Design* de controles e formato de apresentação para acomodar aprendizes com deficiências.
8. **Reusabilidade:** Capacidade de uso em diferentes contextos de aprendizagem e com diferentes alunos.
9. **Conformidade com Padrões:** Adesão a padrões e especificações internacionais.

A aplicação de cada dimensão ao JSimples é detalhada a seguir.

5.4 PONTOS FORTES IDENTIFICADOS

A avaliação revelou excelência em aspectos fundamentais pedagógicos, conforme detalhado a seguir:

D1 - Qualidade do Conteúdo (5/5): A plataforma apresenta implementação fiel à especificação original da linguagem Simples e da MVS. O conteúdo apresentado nas visualizações corresponde ao comportamento esperado conforme documentação original. A terminologia utilizada é consistente com a literatura de compiladores. Não há simplificações excessivas que comprometam a correção técnica, nem detalhamentos que sobrecarreguem cognitivamente o estudante.

D2 - Alinhamento com Objetivos de Aprendizagem (5/5): A ferramenta foi projetada especificamente para atender aos objetivos de aprendizagem da disciplina de teoria de linguagens e compiladores. Cada funcionalidade mapeia-se diretamente a um objetivo pedagógico específico: a visualização da árvore sintática e árvore de derivação permite compreender análise sintática descendente; a execução passo a passo possibilita entender arquitetura de pilha e ciclo *fetch-decode-execute*; o código MVS gerado demonstra tradução de alto nível para instruções de máquina.

D4 - Motivação (5/5): A visualização interativa da árvore sintática e da árvore de derivação transforma conceito abstrato em representação visual. O modo de execução passo a passo permite que estudantes vejam o resultado de suas ações, proporcionando sensação de descoberta ativa. A possibilidade de escrever, compilar e executar programas próprios em linguagem com comandos em português reduz barreiras linguísticas. O *feedback* visual imediato mantém engajamento durante exploração. Esta abordagem alinha-se com preferências estudantis por métodos ativos de aprendizagem identificadas por Stamenković et al (Stamenković; Jovanović; Chakraborty, 2020).

D5 - *Design* de Apresentação (5/5): A interface utiliza hierarquia visual para organizar informações. O *layout* de seis painéis principais (editor de código, console, árvore sintática, árvore de derivação, execução passo a passo da MVS e tabela de símbolos) distribui informações relacionadas sem competição visual. O realce de sintaxe diferencia elementos sintáticos, facilitando reconhecimento de padrões. A árvore sintática e árvore de derivação utilizam representações hierárquicas espaciais que refletem a estrutura de derivação gramatical. Durante execução passo a passo, a instrução atual é destacada, direcionando atenção do estudante. Os registradores e conteúdo da pilha são apresentados de forma organizada, reduzindo carga cognitiva necessária para interpretação do estado da máquina.

D8 - Reusabilidade (5/5): O JSimples demonstra reusabilidade em múltiplos contextos educacionais. A plataforma pode ser utilizada tanto em disciplinas introdutórias de lógica de

programação (focando apenas na escrita e execução de programas Simples) quanto em cursos de compiladores (explorando análise sintática e geração de código). A arquitetura *web* multi-plataforma reduz restrições de sistema operacional ou *hardware*. A conformidade com padrão SCORM possibilita integração com sistemas de gerenciamento de aprendizagem, facilitando adoção institucional.

5.5 OPORTUNIDADES DE MELHORIA

Quatro dimensões apresentaram desempenho satisfatório ou inferior, porém revelaram margem para evolução em aspectos específicos:

D3 - *Feedback* e Adaptação (4/5): A plataforma oferece *feedback* imediato e específico durante compilação, indicando tipo de erro (léxico, sintático, semântico), linha, coluna e descrição em português. Durante execução passo a passo, o estudante visualiza mudanças de estado em tempo real. Entretanto, o sistema não conta um um sistema de *linting* para o editor de código, o que poderia facilitar o entendimento para novos estudantes.

D6 - Usabilidade da Interação (4/5): A navegação entre modos de execução (contínua e passo a passo) é intuitiva. A interface mantém estado consistente durante transições entre funcionalidades. Porém, estudantes sem conhecimento prévio da sintaxe Simples precisam consultar fontes externas. A falta de exemplos pré-carregados ou *templates* de código aumenta a curva de aprendizado inicial. Implementar galeria de programas exemplo diretamente na interface facilitaria exploração guiada e aprendizagem por exemplos.

D7 - Acessibilidade (2/5): Não há suporte para leitores de tela, impedindo uso por estudantes com deficiência visual. A visualização da árvore sintática e árvore de derivação depende exclusivamente de representação gráfica visual, sem alternativa auditiva. Contraste de cores não foi testado para daltonismo. Navegação por teclado é limitada, prejudicando usuários com dificuldades motoras. A interface não oferece opções de ajuste de tamanho de fonte ou alto contraste. A fim de melhorar questões de acessibilidade, seria necessário implementar descrições textuais alternativas para elementos visuais; suporte completo a navegação por teclado com indicadores visuais de foco; e compatibilidade com tecnologias assistivas.

D9 - Conformidade com Padrões (4/5): A plataforma implementa conformidade com padrão SCORM para integração com sistemas de gerenciamento de aprendizagem, facilitando distribuição de uso em contextos institucionais. O código HTML5 e CSS3 segue padrões *web*

modernos, garantindo compatibilidade entre navegadores. Entretanto, ausência de metadados educacionais estruturados dificulta catalogação em repositórios educacionais.

5.6 SÍNTESE DA AVALIAÇÃO PEDAGÓGICA

A avaliação LORI demonstrou que o JSimples apresenta qualidade pedagógica com média de 4,3/5. Os pontos fortes concentram-se em dimensões fundamentais para software educacional: qualidade e precisão do conteúdo (D1), alinhamento direto com objetivos de aprendizagem da disciplina (D2), e *design* de apresentação que facilita processamento cognitivo (D5). A capacidade motivacional (D4) é particularmente notável, transformando conceitos abstratos de compiladores em experiências visuais e interativas concretas. A elevada reusabilidade (D8) confirma versatilidade da ferramenta para múltiplos contextos educacionais.

As oportunidades de melhoria identificadas (D3, D6, D7, D9) referem-se principalmente a funcionalidades de acessibilidade. A ausência de *feedback* adaptativo (D3) e documentação integrada (D6) representa limitação moderada que pode ser contornada através de mediação docente em contexto de sala de aula. Por outro lado, as limitações de acessibilidade (D7) constituem barreira mais significativa para inclusão de estudantes com deficiências, aspecto que deveria ser priorizado em desenvolvimentos futuros. A conformidade parcial com padrões educacionais (D9) não compromete funcionalidade atual, mas dificulta descoberta e compartilhamento da ferramenta em repositórios acadêmicos mais amplos. A Tabela 4 sintetiza todo o conteúdo discutido nos tópicos acima.

Tabela 4 – Avaliação pedagógica do JSimples segundo LORI

Dimensão LORI	**Nota**	**Observação**
D1: Qualidade do Conteúdo	5	Implementação fiel à especificação
D2: Alinhamento com Objetivos de Aprendizagem	5	Mapeamento direto para objetivos da disciplina
D3: *Feedback* e Adaptação	4	*Feedback* claro, falta *linting*
D4: Motivação	5	Visualização interativa, experimentação ativa
D5: *Design* de Apresentação	5	Hierarquia visual clara, realce de sintaxe
D6: Usabilidade da Interação	4	Falta documentação integrada
D7: Acessibilidade	2	Limitações para usuários com deficiências
D8: Reusabilidade	5	Multiplataforma, padrão SCORM
D9: Conformidade com Padrões	4	SCORM implementado, faltam metadados
Média Geral	**4,3**	

Fonte: Elaborado pelo autor.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e avaliação do JSimples, uma plataforma *web* interativa para ensino de conceitos de teoria de linguagens e compiladores. A ferramenta foi concebida para preencher três lacunas identificadas na revisão bibliográfica: a predominância de aplicações desktop que limitam acessibilidade, a ausência de ferramentas *web* com visualização interativa de árvores sintáticas combinada com execução passo a passo, e a carência de recursos didáticos em português contextualizados ao ensino brasileiro.

A implementação utilizou *stack* tecnológico moderno baseado em JavaScript e React, resultando em aplicação multiplataforma que executa diretamente no navegador sem dependências externas. A avaliação através de *frameworks* complementares (Heurísticas de Nielsen para usabilidade e LORI para qualidade pedagógica) resultou em médias de 4,0/5 e 4,3/5 respectivamente, identificando pontos fortes em aspectos como *feedback* visual, correspondência linguística e *design* de apresentação, além de oportunidades de melhoria em documentação, acessibilidade e *feedback* adaptativo.

As limitações identificadas referem-se à ausência de validação empírica com usuários reais, restrições de acessibilidade para estudantes com deficiências e falta de documentação integrada. Trabalhos futuros prioritários incluem condução de estudo experimental para mensurar impacto na aprendizagem, implementação de recursos de acessibilidade e desenvolvimento de funcionalidades de *linting* dentro do editor de códigos. A plataforma está disponível[1] para uso educacional e representa base para evolução incremental através das melhorias identificadas nas avaliações conduzidas.

[1] Disponível em https://jsimples.joaoragazzo.dev/. Acessado em 03 de Dezembro de 2025.
Código fonte disponível em https://github.com/joaoragazzo/jsimples. Acessado em 03 de Dezembro de 2025.

REFERÊNCIAS

ALFRED, V. A.; MONICA, S. L.; JEFFREY, D. U. Compilers principles, techniques tools. **pearson Education**, 2007.

Ant UED. **Ant Design: A UI Design Language and React UI library**. 2015. https://ant.design/. Acessado em: 8 de outubro de 2025.

ARNAIZ-GONZáLEZ Álvar *et al.* Seshat—a web-based educational resource for teaching the most common algorithms of lexical analysis. **Computer applications in engineering education**, v. 26, n. 6, p. 2255–2265, 2018. Disponível em: https://onlinelibrary.wiley.com/doi/epdf/10.1002/cae.22036.

BAïNA, K.; BENATALLAH, B. Is it still worth the cost to teach compiling in 2020? a pedagogical experience through hortensias compiler and virtual machine. **EasyChair**, 2021. Disponível em: https://vixra.org/pdf/2007.0063v2.pdf.

CARSTENS, K. J. *et al.* Effects of technology on student learning. **Turkish Online Journal of Educational Technology-TOJET**, v. 20, n. 1, p. 105–113, 2021. Disponível em: https://eric.ed.gov/?id=EJ1290791.

CHESNEVAR, C. I.; GONZáLEZ, M. P.; MAGUITMAN, A. G. Didactic strategies for promoting significant learning in formal languages and automata theory. **ACM SIGCSE Bulletin**, v. 36, n. 3, p. 7–11, 2004. Disponível em: https://dl.acm.org/doi/10.1145/1026487.1008002.

Free Software Foundation. **Bison - GNU parser generator**. [S.l.], 2025. Disponível em: https://www.gnu.org/software/bison/. Acessado em: 8 de outubro de 2025.

HAVERBEKE, M. **CodeMirror: versatile text editor implemented in JavaScript for the browser**. 2018. https://codemirror.net/. Acessado em: 7 de outubro de 2025.

JOHNSON, S. C. **Yacc: Yet Another Compiler-Compiler**. Murray Hill, New Jersey, 1975.

KOWALTOWSKI, T. Implementação de linguagens de programação. **Guanabara Dois**, 1983.

KREMER, B. **React D3 Tree**. 2017. https://bkrem.github.io/react-d3-tree/. Acessado em: 12 de outubro de 2025.

MADDERN, G.; STOIBER, M. **Styled Components: CSS for the Component Age**. 2016. https://styled-components.com/. Acessado em: 8 de outubro de 2025.

MERNIK, M.; ZUMER, V. An educational tool for teaching compiler construction. **IEEE Transactions on Education**, v. 46, n. 1, p. 61–68, 2003. Disponível em: https://ieeexplore.ieee.org/document/1183668/.

Meta Platforms, Inc. **React 19.1**. 2025. https://react.dev/. Acessado em: 8 de outubro de 2025.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. **Proceedings of the SIGCHI conference on Human factors in computing systems**, p. 249–256, 1990.

OLIVEIRA, H. B. "um compilador, uma linguagem de programação e uma máquina virtual simples para o ensino de lógica de programação, compiladores e arquitetura de computadores". 2014.

RODGER, S. H.; FINLEY, T. W. Jflap: an interactive formal languages and automata package. **Jones Bartlett Learning**, 2006. Disponível em: https://jflap.org/csed/jflap/jflapbook/jflapbook2006.pdf.

SILVA, L. G. D. *et al.* Avaliação da experiência de uso do jflap como recurso pedagógico no ensino de linguagens formais. **Simpósio Brasileiro de Informática na Educação (SBIE)**, p. 995–1006, 2023. Disponível em: https://sol.sbc.org.br/index.php/sbie/article/view/26729.

STAMENKOVIć, S.; JOVANOVIć, N. A web-based educational system for teaching compilers. **IEEE Transactions on Learning Technologies**, p. 143–156, 2023. Disponível em: https://ieeexplore.ieee.org/document/10190165.

STAMENKOVIć, S.; JOVANOVIć, N.; CHAKRABORTY, P. Evaluation of simulation systems suitable for teaching compiler construction courses. **Computer Applications in Engineering Education**, v. 28, n. 3, p. 606–625, 2020. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22231.

TAN, L.; WANG, N. Future internet: The internet of things. **IEEE**, v. 5, n. 376, 2010. 3rd international conference on advanced computer theory and engineering (ICACTE).

VARGO J.C. NESBIT, K. B. J.; ARCHAMBAULT, A. Learning object evaluation: Computer-mediated collaboration and inter-rater reliability. **International Journal of Computers and Applications**, p. 198–205, 2003.

Zach Carter. **Jison Documentation**. 2009. https://gerhobbelt.github.io/jison/docs/. Acessado em: 23 setembro de 2025.