LAILA - UM GERADOR DE ANALISADOR LÉXICO E SINTÁTICO ONLINE BASEADO EM FLEX E BISON

LAILA - AN ONLINE LEXICAL AND SYNTACTIC ANALYZER GENERATOR BASED ON FLEX AND BISON

Micaías Ladgelson da Silva*
Silas Silas Santiago Lopes Pereira**
Diego Rocha Lima***

RESUMO

A disciplina de Compiladores faz parte do currículo da maioria dos cursos de Ciência da Computação e graduações afins. Entretanto, a disciplina pode ser considerada desafiadora, uma vez que parte do seu conteúdo é apontado como abstrato, além de demandar a instalação e a configuração de vários programas para serem manuseados durante o seu conteúdo prático. Todavia, após pesquisas exaustivas notou-se a escassez de alternativas de soluções *online* para a prática de construção de um compilador, no qual nenhuma proposta nessa temática foi encontrada na literatura. Assim, esse trabalho apresenta a Laila, uma plataforma *web* para o apoio ao ensino da construção de um compilador, disponibilizando ao estudante a possibilidade de escrever e testar seus próprios analisadores léxicos e sintáticos em FLEX e Bison, passos imprescindíveis para o entendimento de compiladores. Dessa forma, é possível facilitar o processo educacional de construção de um compilador, melhorando a curva de aprendizado da disciplina. A Laila mostrou ser uma ferramenta efetiva ao ser testada e utilizada em sala de aula por 17 alunos da disciplina de Compiladores do Instituto Federal do Ceará - Campus Aracati, no segundo semestre letivo de 2020.

Palavras-chave: Compiladores. Ferramenta de Ensino. Analisador Léxico e Sintático.

ABSTRACT

The Compilers subject is part of the curriculum of most Computer Science courses and related graduations. However, the discipline can be considered challenging, since part of its content is pointed out as abstract, in addition to requiring the installation and configuration of several

* Graduando em Bacharelado em Ciência da Computação, Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: micaiasladgelsondasilva@gmail.com
** Silas Santiago é professor do IFCE, mestre em ciência da computação na universidade estadual do ceará (UECE) e bacharel em ciência da computação pela UECE. Endereço eletrônico: silas@lar.ifce.edu.br.
*** Diego Rocha é mestre em ciência da computação, Docente do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: diego.rocha@ifce.edu.br

programs to be handled during its practical content. However, after exhaustive research, it was noted the scarcity of alternative online solutions for the practice of building a compiler, in which no proposal on this topic was found in the literature. Thus, this work presents Laila, a web platform to support the teaching of building a compiler, providing students with the possibility of writing and testing their own lexical and syntactic analyzers in FLEX and Bison, essential steps for understanding compilers. This way, it is possible to facilitate the educational process of building a compiler, improving the learning curve of the discipline. Laila proved to be an effective tool when tested and used in the classroom by 17 students of the Compilers course at the Federal Institute of Ceará - Campus Aracati, in the second semester of 2020.

Keywords: Compilers. Teaching Tool. Lexicon and Syntactic Analyzer.

1 INTRODUÇÃO

As linguagens de programação de alto nível fornecem ao programador uma experiência mais intuitiva no desenvolvimento de *software*, contudo, para que elas sejam entendidas pela máquina, é necessário passar por um procedimento de transformação de linguagem. Este processo de transformação de uma linguagem de alto nível para linguagem de máquina, é denominado compilação e é feito por um programa chamado compilador. Um compilador é considerado um módulo elementar de *software* que transforma códigos para alguma forma que possa ser interpretada pelo computador (SETHI; ULLMAN; LAM, 2008).

A compreensão teórica e prática acerca da construção de um compilador tem evoluído nas últimas décadas. Os primeiros compiladores tinham limitações de estruturas de dados e tratamentos de erros, além de serem escritos em linguagens de baixo nível. Todavia, existem atualmente diversos *frameworks* que facilitam o desenvolvimento de uma ferramenta de compilação, a ponto do projetista não se ater aos detalhes que não sejam de sua linguagem (SETHI; ULLMAN; LAM, 2008). Ressalta-se que os fundamentos, métodos e discussões promovidos quanto ao desenvolvimento de um compilador podem ser relevantes para uma série de contextos distintos aplicados à área da computação.

Dado a importância dos conceitos de um compilador para a computação em geral, é sabido que ela é uma disciplina frequente nas grades curriculares em cursos na área da TDIC (Tecnologias Digitais da Informação e Comunicação). No entanto, apesar da sua importante contribuição na formação de um profissional de tecnologia, a mesma é considerada uma disciplina de difícil entendimento pois tem seu fundamento em conceitos de lógica, algoritmos e estruturas de dados (GESSER, 2003). Segundo o estudo de (SARAIVA; DANTAS; RODRIGUES, 2019), que busca entender a motivação das desistências em cursos dessa área, é possível notar uma proporção de evasão de mais de 40% dos estudantes. Ainda nessa pesquisa, é afirmado que uma das motivações desses cancelamentos de matrícula é a dificuldade da compreensão de conceitos computacionais e matemáticos. O trabalho de (MCGETTRICK et al., 2005) também evidencia

que as dificuldades que abrangem programação e lógica são fatores que destacam ainda mais os números com relação à evasão em cursos de computação.

Existem várias ferramentas e plataformas *online* que propõem soluções para o ensino e aprendizado de conceitos lógicos e computacionais, como por exemplo o Code.org[1] e o LeetCode[2]. O Code.org propõe uma plataforma para disseminação e desenvolvimento de raciocínio lógico e computacional, que engloba desde o nível fundamental (crianças) até níveis mais avançados. Já o LeetCode, propõe um ambiente para treino e preparação para entrevistas em grandes empresas de tecnologia, evidenciando conceitos de algoritmos, paradigmas de *web design*, estruturas de dados, entre outros. Vale destacar que as duas plataformas citadas são ferramentas completamente *online* e voltadas para ensino de conceitos concernentes à computação.

Ainda que existam muitas soluções para o apoio ao processo de ensino-aprendizagem de lógica computacional, nota-se uma escassez de ferramentas educacionais direcionadas para facilitar o ensino de compiladores, das quais poucas mostram-se eficazes em abranger todo o conteúdo e técnicas de maior relevância da disciplina (BACKES; DAHMER, 2006). Não obstante, muitas estratégias têm sido estudadas a fim de encontrar melhorias para as abordagens de ensino e aprendizagem de compiladores.

Visto isso, ferramentas para facilitar o ensino e o aprendizado são de grande interesse para muitos professores e estudantes de qualquer eixo, incluindo os da disciplina de Compiladores. Essa disciplina tem como objetivo exibir conceitos e diferentes aspectos referentes ao desenvolvimento de um compilador. Para tanto, o desenvolvimento de um projeto de construção de compilador funcional baseado em uma gramática simples pode ser proposto. Por conseguinte, é comum encontrar o uso de ferramentas capazes de demonstrar como funcionam os processos da fase de análise de um computador, mas que não tem como principal propósito gerar um analisador léxico/sintático (GESSER, 2003).

O LEX (LESK; SCHMIDT, 1975) é um gerador de analisador léxico popular no campo de estudo dos compiladores, utilizado como uma ferramenta para proporcionar uma experiência real, ou seja, ele não funciona apenas de maneira simulatória, ele gera o executável para que o projetista da linguagem em desenvolvimento possa testa-la. O LEX funciona a partir da leitura de uma especificação léxica (formada por expressões regulares) e retorna um arquivo na linguagem de programação C, onde está o analisador léxico desenvolvido. No entanto, a ferramenta é incapaz de exercer as duas categorias de analisadores (léxico e sintático). De maneira análoga, existe o YACC (*Yet Another Compiler-Compiler*) (JOHNSON et al., 1975) que é um gerador de analisador sintático criado com capacidade de receber o arquivo de saída do LEX usando-o como entrada para gerar o analisador sintático (LEVINE et al., 1992).

Do ponto de vista de usabilidade, apesar da compatibilidade entre o LEX e o YACC, o uso dos dois em conjunto ainda promove alguns desafios. Um problema evidente que pode ser citado é o uso dessas ferramentas e linha de comando, visto que muitos usuários não sabem como usá-las corretamente. Outro ponto negativo em destaque é a instalação desses programas,

[1] https://code.org/

[2] https://leetcode.com/

que não são tarefas simples para sistemas operacionais não baseados em Linux, como *Windows* e *MacOS*. Diante disso, a falta de uma *IDE (Integrated Development Environment)* agradável para identificar particularidades da linguagem LEX se torna um problema, uma vez que dificulta a identificação de erros no código do analisador escrito.

O FLEX e o Bison são ferramentas melhoradas de LEX e YACC, respectivamente. Segundo (LEVINE, 2009), um requisito para o uso dessas ferramentas é o GCC *(GNU Compiler Collection)* uma coleção de compiladores para *C*. À vista disso, a máquina utilizada para o desenvolvimento requer a instalação desses três programas para conseguir executar seus analisadores léxicos e sintáticos.

Conforme discutido, diferentes fatores tornam desafiante o processo de ensino e aprendizagem de compiladores. Para o estudante, a compreensão teórica e prática dos diferentes conceitos referentes à disciplina, podem tornar mais lento o processo de aprendizado. Uma ferramenta *online* traria muitas facilidades, visto que seria possível testar o código em qualquer máquina, independente do sistema operacional, inclusive em computadores com o processamento mais lento bastando apenas um navegador instalado e conexão com a *internet*. Contudo, após um conjunto de pesquisas realizadas em ferramentas de busca, não foram encontradas na literatura plataformas educacionais *web* para a construção de compiladores. Desse modo, uma solução educacional que facilite o projeto de construção de um compilador pode aumentar a curva de aprendizado da disciplina.

Este trabalho apresenta a Laila, uma plataforma *web* educacional para o ensino prático de compiladores. A plataforma conta com *IDEs* que disponibilizam ao estudante a possibilidade de escrever e testar seus próprios analisadores léxicos e sintáticos na linguagem de programação C, utilizando FLEX e Bison. Dessa maneira, a Laila permite simplificar o processo educacional de construção de um compilador funcional, melhorando o ensino e aprendizado do curso de Compiladores. Vale destacar que essa ferramenta foi feita buscando acessibilidade e produtividade na criação de compiladores. Outro ponto forte do trabalho está no fato da Laila se mostrar efetiva como um instrumento de apoio ao ensino, após utilizada e testada em sala de aula pelos alunos da disciplina de Compiladores do IFCE (Instituto Federal do Ceará), Campus Aracati, no segundo semestre letivo de 2020.

O presente trabalho está estruturado da seguinte maneira: A Seção 2 revisa todos os conceitos chaves relacionados à temática; já a Seção 3 apresenta alguns trabalhos relacionados à esta pesquisa; a Seção 4 descreve a metodologia utilizada para desenvolvimento da solução proposta; na Seção 5 são apontados os resultados prévios. Na Seção 6 estão contidas as conclusões e discussões sobre fatos importantes percebidos já apresentados, e por fim, as referências bibliográficas.

2 O PROCESSO DE COMPILAÇÃO

Um compilador é um programa capaz de gerar outro programa com semelhanças semânticas à linguagem usada na criação, consequentemente, cria um *software* que entende outra

linguagem. Independente da linguagem utilizada, um compilador primeiro analisa o código-fonte escrito procurando erros e informações importantes. Com as informações já obtidas, o processo de compilação gera um programa correspondente em linguagem de máquina, no qual é entendido e executado pelo processador (DAS, 2007). Um processo normal de compilação compreende duas fases, sendo elas a de análise e a de síntese. Neste trabalho, vamos focar apenas na fase de análise para manter o foco nos objetivos do trabalho.

A fase de análise, corresponde à avaliação do código-fonte em diferentes perspectivas, considerando um fluxo para o sucesso da compilação. Esse fluxo (Figura 1) é composto pelas etapas de análise léxica, análise sintática e análise semântica, respectivamente. A maior parte dos erros de compilação são encontrados durante a execução dessa fase. Em função disso, o processo é abortado e todos os erros são destacados para análise e correção do código. Em particular, cada tipo da análise possui um objetivo específico e claro, os quais em conjunto são capazes de verificar vários aspectos sobre o código-fonte apresentado inicialmente.

Figura 1 – Fluxo do processo de compilação.

Fonte: Elaborada pelos autores (2021).

A **análise léxica** é a primeira etapa do processo e tem como objetivo separar todas as *strings* do código-fonte, além de reconhecer as sequências de significância denominadas de lexemas, gerando um *token* para cada uma delas (FOLEISS et al., 2009). Em seguida, o analisador mapeia as saídas obtidas como: palavras reservadas, literais numéricos, literais de texto, operadores matemáticos, operadores lógicos e entre outros. Ainda nessa etapa, é efetuada também a remoção de quaisquer elementos que não pertencem diretamente ao programa, como por exemplo, os comentários adicionados podem ser usados para destacar alguma parte importante ou até mesmo ajudar a explicar partes específicas do código em questão. Contudo, não são necessários para a execução do código, tornando-os elementos não essenciais que são removidos pelo analisador.

A **análise sintática** corresponde à segunda etapa do processo e sucede a análise léxica. Logo, depois que os *tokens* são atribuídos na primeira etapa, o analisar sintático os recebe como entrada e verifica se estão seguindo as regras designadas pela linguagem. Evidencia-se que essa técnica não se difere de métodos utilizados em linguagens naturais, como o português. Por exemplo, na frase: *"lucas Não usa isso."*, existem dois erros sintáticos. O primeiro erro identificado é não iniciar nome próprio com letra maiúscula e o segundo se refere ao uso de letra maiúscula indevidamente. De maneira análoga, o código-fonte é analisado sintaticamente e em seguida gera uma AST *(Abstract Syntax Tree)*. A AST é uma estrutura de dados em formato de

árvore que representa a estrutura sintática do fluxo de *tokens*. Assim, o compilador é capaz de relatar um erro sempre que faltar algum *token* na árvore.

Na última etapa, a **análise semântica** é responsável por fornecer significado para suas construções, como *tokens* e estrutura de sintaxe. A semântica ajuda a interpretar os símbolos, seus tipos e suas relações uns com os outros. Além disso, a mesma avalia se a estrutura da sintaxe construída no programa de origem obtém algum significado.

Concluindo, esse fluxo de processo de compilação completo é referente ao conteúdo teórico e prático ensinado em disciplinas de Compiladores em seus respectivos cursos.

3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados trabalhos desenvolvidos com o objetivo de melhorar o desempenho do ensino teórico e prático de compiladores. Todos os trabalhos apresentados estão inseridos no contexto da disciplina de compiladores, contudo, nenhum destes dispõe de uma ferramenta compatível com uma experiência real, ou seja, capaz não apenas de simular mas de gerar os analisadores léxicos, sintáticos e semânticos.

Em (GESSER, 2003) o autor apresenta o GALS (Gerador de Analisadores Léxicos e Sintáticos), um simulador de analisadores léxicos e sintáticos para auxiliar didaticamente o aprendizado da construção de analisadores. O uso da ferramenta é vantajoso por ser um arquivo JAR (*Java Archive*), ou seja, pode ser executado facilmente em qualquer sistema operacional. Vale destacar que GALS também permite mostrar a árvore de recursão do analisador sintático como parte da proposta ao ensino didático. Ainda, para aprimorar o desempenho no ensino em universidades o autor propõe o uso em sala de aula. No GALS, é possível simular analisadores léxicos e sintáticos, no entanto, por se tratar de uma simulação para fins didáticos, a ferramenta priva o aluno da experiência de construir seus próprios analisadores.

A proposta principal do trabalho de (MERNIK; ZUMER, 2003) é apresentar a ferramenta *LISA* (*Language Implementation System based on Attribute grammars*), que busca facilitar o aprendizado e a compreensão conceitual da construção do compilador de maneira eficiente, direta e persistente. No ambiente de ensino *LISA* os alunos dispõem da possibilidade de experimentar, estimar e testar vários analisadores léxicos e sintáticos, bem como aprender estratégias de avaliação de atributos. Além disso, a *LISA* é um ambiente de desenvolvimento integrado no qual os usuários podem especificar, gerar, compilar e executar programas em uma linguagem recém-especificada. Para cada fase (léxica, sintática e semântica), uma animação apropriada é projetada na tela buscando aprimorar o modelo cognitivo do visualizador. Os autores relatam a experiência realizada com a LISA, no qual os seguintes benefícios didáticos foram observados: melhor compreensão de conceitos, maior motivação de aprendizagem por parte dos envolvidos, estimulação de aprendizagem de forma ativa/exploratória e suporte para o aprendizado construtivo.

No trabalho de (ARNAIZ-GONZÁLEZ et al., 2018), é descrito o *Seshat*, uma plataforma *web* para o ensino de linguagens regulares, expressões regulares e autômatos finitos. O *Seshat*

Tabela 1 – Comparativo dos Trabalhos Relacionados.

Trabalho	Autor/Ano	Simulador	Experiência real	Ambiente
GALS	Gesser et al. (2003)	X		Programa multiplataforma
LISA	Mernik, Zumer (2003)	X		Programa desktop para Windows
Seshat	Arnaiz Gonzáles el al. (2018)	X		Plataforma Web
Laila	Ladgelson (2021)		X	Plataforma Web

Fonte: Elaborada pelos autores (2021).

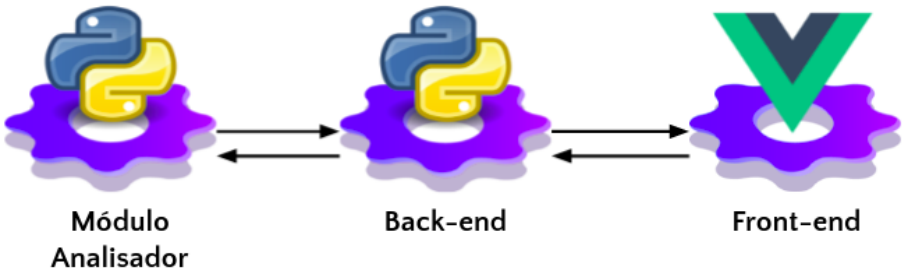
permite que os usuários possam interagir de maneira visual com os algoritmos mais comuns da área de análise léxica e da teoria dos autômatos. Essa solução, permite também que o usuário possa visualizar o passo-a-passo da execução de cada um desses algoritmos, tornando mais simples e visual um conteúdo geralmente classificado como complexo. Outro ponto em destaque é a possibilidade de ver esses exemplos quantas vezes forem necessárias para o bom entendimento do conteúdo. Por se tratar de uma plataforma *web*, o *Seshat* pode ser acessado de qualquer dispositivo, dispensando a necessidade de instalações. O Seshat é uma ferramenta moderna e relevante para aprender a parte teórica, no entanto, a mesma não abrange uma experiência real quanto à construção de compiladores.

A Tabela 1, exibe um comparativo entre os trabalhos abordados nessa seção, com destaque ao trabalho em questão. Apesar dos trabalhos apresentados, ainda há uma carência de soluções concretas que facilitem a prática de construção de compiladores. Observa-se que, dentre os citados, todos se tratam de simuladores. O trabalho proposto, se mostra único pois oferece uma maneira de impulsionar o estudo prático de compiladores de maneira integral, não como simulador, mas sim como um gerador de analisador léxico e sintático *online*. Outro ponto que vale destacar é o ambiente de produção, o que pode limitar a utilização dos trabalhos, como no caso de (GESSER, 2003) e (MERNIK; ZUMER, 2003). No caso do ambiente de plataforma *web*, onde o Laila está inserido, o mesmo pode ser usado de qualquer dispositivo em qualquer sistema operacional.

4 LAILA

Esta seção descreve a arquitetura de execução da Laila, um sistema *web* para apoio ao ensino de compiladores que permite o desenvolvimento de analisadores léxicos e sintáticos totalmente *online*. A interface *web* possui uma IDE com a qual o aluno pode escrever suas especificações léxicas (para FLEX), assim como descrever suas próprias gramáticas regulares para a sintaxe da sua linguagem (escrita para o Bison interpretar). É importante destacar que foram usadas as ferramentas FLEX e Bison por serem manuseados para a construção de compiladores na disciplina de Compiladores no IFCE - Campus Aracati, a qual foi utilizada como base para a elaboração do trabalho.

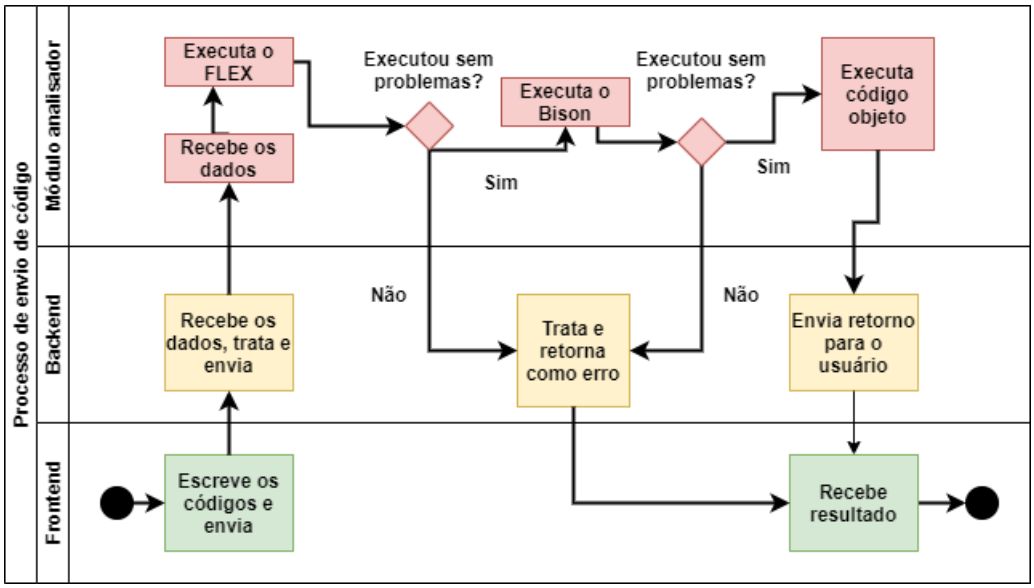
Figura 2 – Arquitetura da Laila.



Fonte: Elaborada pelos autores (2021).

A Laila foi estruturada para suportar outros analisadores léxicos e sintáticos posteriormente, todavia, para ter uma aplicação e avaliação mais rápida da proposta, foi implementado para as bibliotecas citadas anteriormente. A Figura 2 apresenta a arquitetura da proposta, composta por 3 módulos que serão detalhados durante a exibição desta seção. De antemão, o *front-end* é o componente da aplicação projetado para visualização e utilização do usuário final, neste trabalho, representado por alunos e/ou professores. Sua interface foi idealizada para ser simples, intuitiva e clara, a fim de tornar agradável suas experiências de usabilidade. O *front-end* se comunica diretamente com o *back-end*, solicitando a execução dos analisadores léxico e sintáticos. O *back-end*, por sua vez, recebe essas requisições e se comunica com o módulo analisador, se comportando como um interventor entre os dois processos. O módulo analisador recebe os dados do *back-end* e executa os analisadores léxicos e sintáticos e os retorna, dessa forma, é possível evitar que ocorram sobrecargas nesse módulo.

Figura 3 – Fluxo do processo de compilação.



Fonte: Elaborada pelos autores (2021).

A Figura 3 mostra o passo-a-passo do fluxo de envio das estruturas necessárias para a execução da linguagem que está sendo projetada pelo usuário. No início do fluxo, está o

front-end. Nele, é possível que o usuário escreva a nova linguagem projetada e a envie para ser processada. Em seguida, os dados enviados são recebidos e tratados pelo *back-end*, e então, enviados para o módulo de análise. Nesse módulo, as bibliotecas são executadas e caso algum erro seja identificado, as mensagens de erro são retornadas, caso contrário, o módulo envia o resultado da execução da nova linguagem. A comunicação entre esses módulos captura os erros de várias partes do fluxo de execução, como: erros na especificação léxica, erros na especificação sintática, erros no código teste da linguagem em construção (relativo a erros de sintaxe). Após essa captura, é retornado para o módulo *front-end* para que o usuário possa entender e tentar resolver o problema encontrado.

A arquitetura demonstrada no presente artigo foi projetada para modularizar os processos a fim de que cada componente tenha um objetivo claro e de fácil de manutenção.

4.1 Módulo Analisador

O Módulo Analisador é o primeiro e mais importante dos serviços, o FLEX e o Bison estão instalados nele. Esse componente tem como objetivo receber especificações da linguagem em desenvolvimento e suas respectivas entradas, executá-las e retornar a saída da nova linguagem emergente.

Para a construção desse componente, foi utilizada a linguagem de programação Python[3], com o auxílio do *microframework* Flask[4] para o desenvolvimento *web*. Segundo (GRUS, 2019) Python já se mostrou como uma das linguagem mais populares, além de ser simples e de fácil codificação, fornece diversas bibliotecas úteis, as quais possibilitam a execução de ferramentas de linha de comando, a exemplo do módulo chamado 'os' (Operational System), uma biblioteca permite a execução de comandos do sistema operacional. Em suma, o Módulo Analisador foi construído como uma *API REST* (Application Programming Interface Representational State Transfer) que recebe em uma requisição POST contendo o seguinte conjunto de dados: a especificação para FLEX, a especificação para Bison, o código exemplo da linguagem que está sendo especificada e um conjunto de entrada de teste.

```
1 flex -i mylexx.l 2$>$ erroLex
2 bison myBison.y 2$>$ erroBison
3 gcc myBison.tab.c -o analisador -lfl -lm 2$>$ erroC
4 ./analisador $<$ entrada
```

Script 1 – Execução do FLEX e Bison no Módulo de Análise.

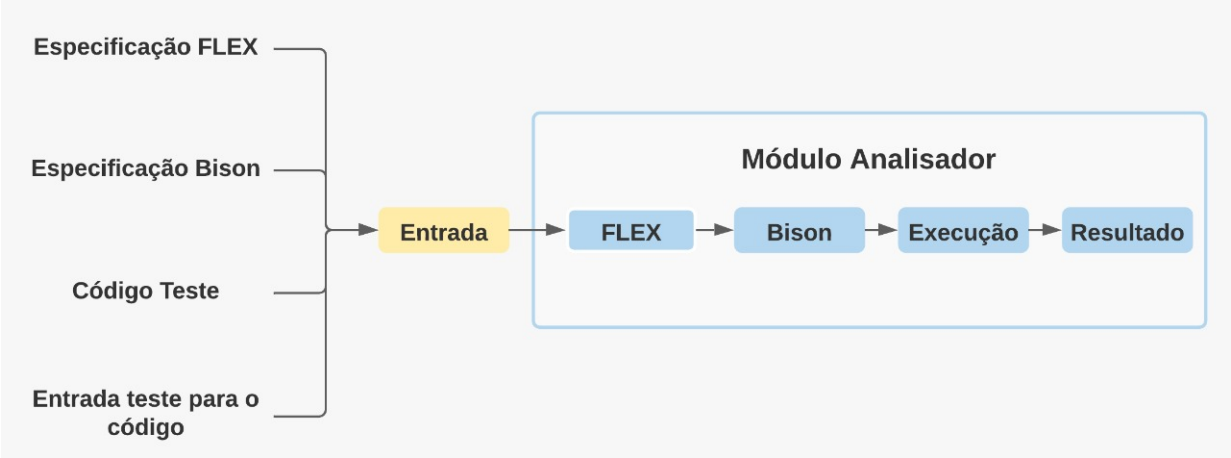
O *script* 1 se refere aos comandos que são executados no *shell* pelo módulo analisador. Esse *script* descreve como são executados os programas após o recebimento de todas as especificações necessárias que compõem cada parte da estrutura da Laila. Na linha 1, é executado o FLEX e redirecionada a saída padrão do erro para um arquivo, caso ocorra algum erro nessa fase da execução. A execução do FLEX gera um arquivo de especificação léxica chamado *lex.yy.c*.

[3] https://www.python.org/
[4] https://flask.palletsprojects.com/en/2.0.x/

Na linha 2 é usada essa especificação léxica gerada pelo FLEX como entrada para a execução do Bison. Na linha 3 é executado em C o código gerado pelo Bison. Por fim, na linha 4 é executado o programa final gerado, aplicando como entrada do programa as entradas recebidas.

A Figura 4 exibe o fluxo do processo no momento em que o usuário solicita a execução de seus analisadores léxicos e sintáticos. Continuamente, é recebida uma requisição com as informações dos códigos de especificação léxica (FLEX) e sintática (Bison), um trecho de código da linguagem que está sendo especificada e uma possível entrada para o código, caso esta tenha sido efetuada.

Figura 4 – Processo no Módulo de Análise.



Fonte: Elaborada pelos autores (2021).

A viabilidade da proposta se dá a partir do conjunto dos processos desse serviço. Isto é, o módulo analisador exige o *deploy* em uma máquina, uma vez que, é essencial a instalação das ferramentas de linha de comando FLEX e Bison. Além disso, a *API REST* faz uso do módulo 'os' (nativa na linguagem *Python*), que executa os comandos apresentados no *script* 1, trata os dados e os retorna.

Além desses processos, para verificar a exequibilidade desse módulo após sua prototipação, foram criados cenários para o envio de analisadores. Para tanto, foram consideradas todas as entradas possíveis disponíveis ao usuário, a fim de identificar possíveis erros em sua execução. Os casos simulados foram:

- **Caso 1:** O usuário envia o analisador léxico com erros e o código teste.
- **Caso 2:** O usuário envia o analisador léxico com erros, o analisador sintático e o código teste.
- **Caso 3:** O usuário envia o analisador léxico, o analisador sintático com erros e o código teste.
- **Caso 4:** O usuário envia o analisador léxico, o analisador sintático e o código teste contendo uma solicitação de entrada do teclado, mas o usuário não envia nenhum dado dessa entrada.

- **Caso 5:** O usuário envia o analisador léxico, o analisador sintático e o código teste contendo uma solicitação de entrada do teclado, mas o usuário não envia todos os dados dessa entrada.

A relevância desses casos de teste surgem da necessidade do sistema responder adequadamente em caso de erros de entrada do usuário. Essa necessidade surge em conformidade com o cenário do usuário estudante, onde ele naturalmente estará em processo de aprendizado e poderá tentar executar o seu compilador mesmo contendo erros. Dentre os casos testados, os casos 4 e 5 acarretaram falhas inesperadas.

O problema encontrado pode ser explicado da seguinte forma: suponhamos que o aluno crie corretamente seu compilador, então, este terá o código léxico (para FLEX) e o sintático (para Bison). Para testar a linguagem recém criada, o aluno elabora um código teste contendo os termos e palavras da sua linguagem. No entanto, se esse código de teste requisitar uma entrada do teclado, como a leitura de números ou *strings*, todos esses dados devem ser informados corretamente na aba de *input* da plataforma. Caso contrário, o sistema operacional aguarda a entrada desses dados do teclado, contudo, não há como acessar aquele processo na máquina do modulo analisador e escrever no teclado as entradas necessárias, já que ele é apenas um serviço REST. Consequentemente, o resultado é um fenômeno conhecido na computação como *deadlock*, o qual ocorre quando dois ou mais processos estão esperando indefinidamente por um evento que só pode ocorrer por um dos processos em espera (CHANDY; MISRA; HAAS, 1983).

Considerando esse problema, foi implementada a seguinte estratégia para evitá-lo: cada vez que o módulo analisador tenta executar o código objeto gerado, um processo do sistema operacional é criado. Esse processo tem um identificador único também chamado de PID (Process Identifier). Para evitar o *deadlock*, é necessário descobrir o PID dessa execução, interrompe-lo e retornar uma mensagem de TLE *(Time Limit Exceeded)*. Para essa finalidade, foi usada uma biblioteca nativa do *Python* chamada *multiprocessing*, com a qual pode-se obter o PID do processo, e então parar a sua execução. Dessa maneira, é possível evitar o problema que poderia sobrecarregar o servidor e trazer confusão para o aluno quanto à plataforma.

4.2 *Back-end*

O *back-end* foi idealizado para diminuir a sobrecarga do módulo analisador e evitar falhas e conflitos em caso de duas ou mais chamadas feitas ao mesmo tempo. Ou seja, o *back-end* recebe as requisições que seriam enviadas diretamente para o analisador e as trata antes de repassá-las. Esse módulo funciona com a execução dos seguintes passos: 1) o *back-end* recebe uma requisição; 2) gera um identificador único para essa requisição; 3) por fim, envia a requisição com identificador para o módulo analisador. Em contrapartida, o analisador recebe a requisição com o identificador, em seguida, é criada uma pasta para cada requisição ser processada separadamente. Após esse processo, a pasta é deletada automaticamente, evitando possíveis conflitos futuros.

4.3 *Front-end*

O *front-end* é o módulo responsável por disponibilizar uma interface para o usuário final e é composta por duas telas, a tela inicial e a tela de IDEs. Para o desenvolvimento desse módulo, foi utilizado o *framework* VueJS[5], uma biblioteca em Javascript escolhida por sua capacidade de componentização, facilidade e usabilidade, além de possuir uma grande comunidade que auxilia no apoio à dúvidas, dicas e padrões de projeto.

A criação das IDEs foram planejadas para garantir uma boa experiência de usabilidade ao usuário final, para isso, foi escolhida a biblioteca *CodeMirror*[6], um editor de texto implementado em *Javascript* com foco em linguagens de programação. Essa poderosa ferramenta unida ao *framework* VueJS tornam a Laila simples e confortável para a edição dos códigos.

O *Script* 2 expõe o código da função **main()** padrão a ser colocado dentro do analisador sintático. A linha 3 define o nome do arquivo que é necessário para o funcionamento do projeto. Caso o usuário mude essa linha, o analisador não consegue ler o código teste, dificultando reconhecer o erro gerado. Para evitar esse problema, é lido todo o código do analisador sintático e encontrado a função **main()**, então a mesma é removida e a função padrão é posta no lugar, garantindo o funcionamento da arquitetura.

```
1 #include "lex.yy.c"
2 int main() {
3   yyin=fopen("file.newlanguage","r");
4   yyparse();
5   yylex();
6   fclose(yyin);
7 return 0;
8 }
```

Script 2 – Função main() padrão para os analisadores sintáticos

5 RESULTADOS E DISCUSSÕES

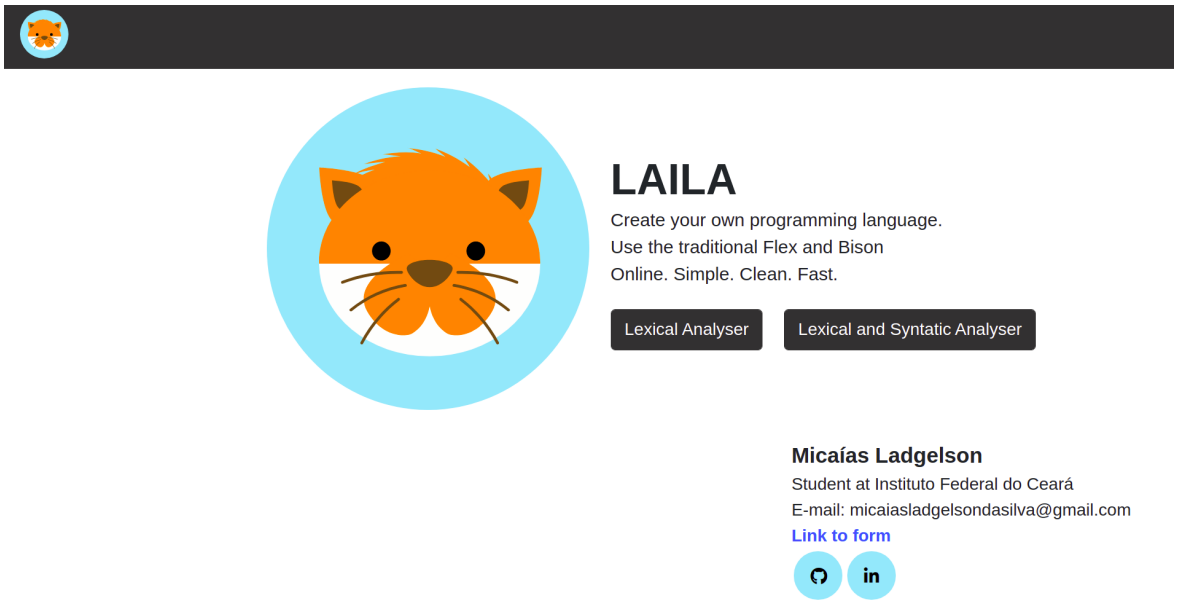
A visualização inicial da plataforma Laila[7] é apresentada na Figura 5, onde o usuário tem a opção de selecionar entre utilizar apenas a ferramenta FLEX para análise léxica ou para análise léxica e sintática.

Esta ferramenta possibilita que os usuários possam construir e testar um compilador a partir de uma interface amigável e de simples uso. O propósito da plataforma *web* é impulsionar a experiência do usuário na aprendizagem de compiladores, dispensando a necessidade de baixar módulos individuais que ocupam memória desnecessária.

A Figura 6 apresenta a *IDE* disponibilizada na plataforma para a inserção dos códigos de construção de um compilador. Em caso análogo, após o usuário escrever os seus analisadores e

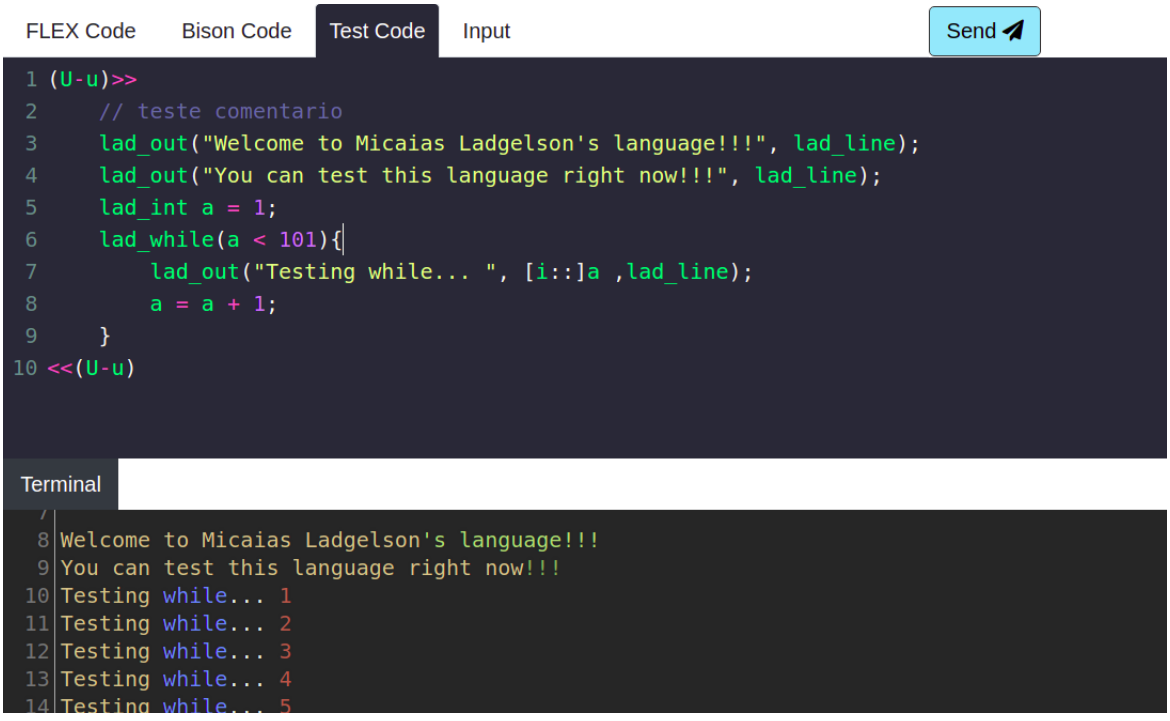
[5] https://vuejs.org/
[6] https://codemirror.net/
[7] http://200.129.3.5:8080//

Figura 5 – Tela *Home* da Laila.



Fonte: Elaborada pelos autores (2021).

Figura 6 – Tela de IDEs.



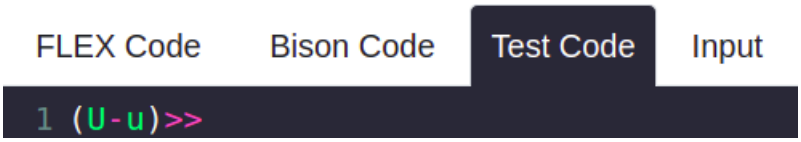
Fonte: Elaborada pelos autores (2021).

clicar no botão *send* (enviar), é requisitada a execução e ele poderá visualizar as saídas e *logs* na parte inferior da interface.

A Figura 7 dá enfoque às abas de funcionalidades da tela de IDEs. Esse menu possui quatro abas, o "FLEX Code", "Bison Code", "Text Code"e "Input". Nas duas primeiras abas, é escrito código C com algumas particularidades do FLEX e Bison. Na aba "Test Code"o usuário escreve o código da linguagem que ele está desenvolvendo, enquanto o "Input"está disponível

para entradas do teclado solicitadas. Destaca-se que a plataforma também consegue reconhecer as palavras reservadas do FLEX, Bison e C. Desta maneira, quando o usuário estiver programando, este poderá reconhecer erros de digitação e sintaxe antes de enviar o código para execução.

Figura 7 – Abas de funcionalidades.



Fonte: Elaborada pelos autores (2021).

5.1 Estudo de caso

Este estudo de caso foi realizado com a finalidade de analisar e avaliar a capacidade interativa dos estudantes com a ferramenta apresentada, a saber: a plataforma Laila. A aplicação desse experimento teve duração de um mês, de 7 de Abril à 7 de maio de 2021, e contou com o apoio de uma equipe composta por: dois professores, estudantes do curso de Bacharelado em Ciência da Computação (BCC), do IFCE (Instituto Federal de Ciência e Tecnologia do Ceará), Campus Aracati. A aplicação aconteceu por meio de três tarefas simples, sendo elas: (1) apresentação da solução para a turma de Compiladores, (2) experimentação da Laila por parte dos alunos com seus compiladores em desenvolvimento e (3) avaliação da plataforma através de um formulário no *Google Forms*[8].

O primeiro passo do estudo de caso ocorreu em 9 de Abril de 2021, em uma aula de Compiladores na plataforma *Google Meet*[9]. O encontro foi de caráter *online* por conta da situação pandêmica atual do país. Os grupos alcançados na apresentação da plataforma foram: alunos do semestre corrente da disciplina de Compiladores, ex-alunos da disciplina e professores do IFCE. O encontro se dividiu em duas partes. Em um primeiro momento, a plataforma foi apresentada com demonstrações de exemplos. Após o término desta, foi aberto um espaço para dúvidas e *feedbacks*. Finalizando, o *link* da Laila foi disponibilizado para os alunos e ex-alunos da disciplina para que testassem e, posteriormente, respondessem ao questionário avaliativo.

O questionário teve como objetivo obter um *feedback* desses estudantes após sua interação com o *software* educacional. Desse modo, o formulário foi preenchido de acordo com o nível de concordância do estudante sobre as questão apresentadas. Ou seja, para cada afirmação, o aluno selecionou de (1) a (5), onde (1) significa que o aluno discordou fortemente e (5) que o aluno concordou fortemente com a declaração. As afirmações elaboradas para o formulário foram baseadas no estudo de caso do trabalho de (ARNAIZ-GONZÁLEZ et al., 2018). As questões aplicadas podem ser visualizados logo abaixo:

1. A ferramenta é usual para entender como os algoritmos funcionam.

[8] https://www.google.com/intl/pt-BR/forms/about/
[9] https://meet.google.com/

2. A ferramenta é bem projetada e fácil de usar.
3. É bem melhor usar essa ferramenta *online* do que rodar por linha de comando.
4. Eu gostaria de ter tido oportunidade de usar a ferramenta quando cursei a disciplina.*
5. O uso dessa ferramenta da engajamento no processo de aprendizado dada sua praticidade.

É de suma importância destacar a questão quatro. Esse questionário foi dado para alunos e ex-alunos da disciplina de compiladores, no entanto, os alunos do semestre corrente ainda não finalizaram a disciplina. No entanto, durante o formulário é explicado que para quem já cursou intérprete-se como está escrito, porém, para quem não cursou ainda, entende-se que eles respondem isso pensando que receberam a ferramenta na segunda metade do semestre, não podendo usar durante toda a disciplina.

Além da apresentação em sala, houve um acompanhamento para auxiliar os alunos da disciplina quanto ao uso da Laila. Por fim, foi realizada a coleta das respostas obtidas no questionário, utilizadas para análise de usabilidade da plataforma.

5.2 Avaliação do *Feedback* dos estudantes

A avaliação do nível didático de um recurso ou ferramenta pode ser dada através do *feedback* de usuários (LECKEY; NEILL, 2001), no questionário da Laila esses usuários são alunos. Estudantes de corrente semestre (2020.2) e semestres anteriores (de 2016.1 à 2020.1) responderam a pesquisa de forma optativa. Ao todo, foram obtidas 17 respostas de alunos e ex-alunos da disciplina. A Tabela 2 mostra as afirmações sobre a ferramenta.

Tabela 2 – Resultados da pesquisa em termos de porcentagem.

Questões	Discordo Fortemente	Discordo	Neutro	Concordo	Concordo Fortemente
Questão 1	0 (0%)	0 (0%)	2 (11,80%)	5 (29,40%)	10 (58.80%)
Questão 2	0 (0%)	0 (0%)	0 (0%)	7 (41,20%)	10 (58,80%)
Questão 3	0 (0%)	0 (0%)	0 (0%)	6 (35,3%)	11 (64,70%)
Questão 4	0 (0%)	0 (0%)	0 (0%)	3 (17,60%)	14 (82,40%)
Questão 5	0 (0%)	0 (0%)	0 (0%)	6 (35,3%)	11 (64,70%)

Fonte: Elaborada pelos autores (2021).

Alguns destaques interessantes podem ser inferidos através dos resultados coletados. Para melhor entendimento, destacamos indicadores obtidos sobre as opiniões dos alunos:

- Os alunos acharam a Laila útil para aprender a parte prática de construção de analisadores léxicos e sintáticos.
- O *design* simples da interface significa que a interação com o recurso é fácil e intuitivo.
- O recurso é tão útil que a maioria dos alunos teria gostado de ter acesso a ele durante toda a disciplina, o que indica suas capacidades pedagógicas.

- Os alunos acham o processo de aprendizagem mais envolvente e atraente quando a ferramenta é usada.

O questionário foi utilizado para verificar os pontos fortes como recurso didático. Os resultados do questionário sugeriram uma melhor compreensão dos alunos sobre os conceitos da disciplina após o uso da Laila. Além disso, no questionário também havia um espaço para opiniões de *design*, experiência de usuário e funcionamento. Então, após a coleta das opiniões dos alunos sobre a ferramenta, foram feitas algumas pequenas alterações que deixaram a interface do site mais amigável, como troca de posições de componentes e troca da paleta de cores do sistema. As respostas dos alunos em relação à aplicação foram fortemente positivas, o que expressa a viabilidade da utilização de recursos adicionais semelhantes a Laila. Esses recursos podem tornar as aulas mais dinâmicas e facilitar no ensino e aprendizado da construção de um compilador.

6 CONSIDERAÇÕES FINAIS

Este trabalho descreveu o desenvolvimento e utilização da Laila, uma plataforma para auxílio ao ensino de compiladores, visando a carência de soluções *online* nesse campo de estudo. Dada essa lacuna existente, esse trabalho explica como essa plataforma consegue preencher a mesma. Com o módulo de análise e a ferramenta *web*, o aluno pode projetar, escrever e testar seus analisadores léxicos e sintáticos. As linguagens de programação Python e Javascript foram usadas para o desenvolvimento de toda a arquitetura, dada a eficiência e a natureza distribuída de aplicações em que elas podem ser usadas.

Os resultados apresentados neste trabalho são promissores. A ferramenta já está em execução e sendo utilizada por alunos da disciplina de Compiladores do IFCE, Campus Aracati. Dessa forma, espera-se auxiliar o ensino de compiladores deste e de outros cursos de computação com o uso de uma ferramenta tecnológica que auxilia no desenvolvimento de um compilador funcional *online*. Salienta-se também que com os resultados obtidos no estudo de caso, a Laila se mostrou uma ferramenta capaz de cumprir esses objetivos.

Como trabalhos futuros, pretende-se tornar a plataforma *open-source*, para que a mesma seja capaz de comportar melhorias também efetivadas pela comunidade. Também pretende-se adicionar módulos com conteúdos para tornar a plataforma mais didática aos usuários, através de exemplos e exercícios. Ainda, objetiva-se a implementação de projeto em contas, para que o aluno possa continuar seus projetos salvos na plataforma sempre que quiser. Outro ponto, é a implementação para diferenciar contas de alunos e professores, para que possam haver tipos de avaliações e atividades na própria plataforma.

REFERÊNCIAS

ARNAIZ-GONZÁLEZ, Á. et al. Seshat — A Web-based Educational Resource for Teaching the most Common Algorithms of Lexical Analysis. **Computer Applications in Engineering Education**, Wiley Online Library, v. 26, n. 6, p. 2255–2265, 2018.

BACKES, J.; DAHMER, A. C-gen: Ferramenta de Apoio ao Estudo de Compiladores. In: **XIV WEI-Workshop sobre Educação em Computação, Campo Grande**. [S.l.: s.n.], 2006.

CHANDY, K. M.; MISRA, J.; HAAS, L. M. Distributed Deadlock Detection. **ACM Trans. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 1, n. 2, p. 144–156, maio 1983. ISSN 0734-2071. Disponível em: <https://doi.org/10.1145/357360.357365>.

DAS, V. V. **Compiler Design using FLEX and YACC**. [S.l.]: PHI Learning Pvt. Ltd., 2007.

FOLEISS, J. H. et al. Scc: Um Compilador C como Ferramenta de Ensino de Compiladores. In: **WEAC2009-Workshop Educação em Arquitetura de Computadores**. [S.l.: s.n.], 2009. p. 15–22.

GESSER, C. E. GALS - Gerador de Analisadores Léxicos e Sintáticos. **Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)–Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis**, 2003.

GRUS, J. **Data Science do Zero: Primeiras Regras com o Python**. [S.l.]: Alta Books, 2019.

JOHNSON, S. C. et al. **Yacc: Yet Another Compiler-Compiler**. [S.l.]: Bell Laboratories Murray Hill, NJ, 1975. v. 32.

LECKEY, J.; NEILL, N. Quantifying Quality: the Importance of Student Feedback. **Quality in Higher Education**, Taylor & Francis Group, v. 7, n. 1, p. 19–32, 2001.

LESK, M. E.; SCHMIDT, E. **Lex: A Lexical Analyzer Generator**. [S.l.]: Bell Laboratories Murray Hill, NJ, 1975.

LEVINE, J. **Flex & Bison: Text Processing Tools**. [S.l.]: "O'Reilly Media, Inc.", 2009.

LEVINE, J. R. et al. **Lex & Yacc**. [S.l.]: "O'Reilly Media, Inc.", 1992.

MCGETTRICK, A. et al. Grand Challenges in Computing: Education—a Summary. **The Computer Journal**, OUP, v. 48, n. 1, p. 42–48, 2005.

MERNIK, M.; ZUMER, V. An Educational Tool for Teaching Compiler Construction. **IEEE Transactions on Education**, IEEE, v. 46, n. 1, p. 61–68, 2003.

SARAIVA, J.; DANTAS, V.; RODRIGUES, A. Compreendendo a Evasão em uma Década no Curso Sistemas de Informação à Luz de Fatores Humanos e Sociais. In: SBC. **Anais do IV Workshop sobre Aspectos Sociais, Humanos e Econômicos de Software**. [S.l.], 2019. p. 21–30.

SETHI, R.; ULLMAN, J. D.; LAM, M. S. **Compiladores: Princípios, Técnicas e Ferramentas**. [S.l.]: Pearson Addison Wesley, 2008.