UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RICARDO LUIS SOUZA ARDISSONE

GANITA: INTERFACE GRÁFICA PARA ENSINO DO MÉTODO DOS TABLEAUX ANALÍTICOS

CURITIBA

2025

versão 11.0 (abr. 25)

RICARDO LUIS SOUZA ARDISSONE

GANITA: INTERFACE GRÁFICA PARA ENSINO DO MÉTODO DOS TABLEAUX ANALÍTICOS

GANITA: Graphical interface for teaching the method of analytic tableaux

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Prof. Dr. Adolfo Gustavo Serra Seca Neto

CURITIBA

2025

RICARDO LUIS SOUZA ARDISSONE

GANITA: INTERFACE GRÁFICA PARA ENSINO DO MÉTODO DOS TABLEAUX ANALÍTICOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 1 / Agosto / 2025

Adolfo Gustavo Serra Seca Neto
Doutorado
Universidade Tecnológica Federal do Paraná

Maria Claudia Figueiredo Pereira Emer
Doutorado
Universidade Tecnológica Federal do Paraná

Cesar Augusto Tacla
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA

2025

Dedicado à memória de
José Luis María Ardissone Nunes (1940-2025)
Arquiteto, Ator, Avô

AGRADECIMENTOS

Agradeço ao Prof. Dr. Adolfo Gustavo Serra Seca Neto pela orientação, compreensão e apoio.

Agradeço aos professores e monitores que fizeram comentários e sugestões sobre o trabalho pela contribuição.

Agradeço ao Prof. Dr. Davi Romero de Vasconcelos pela criação do ANITA, fundamento essencial a este trabalho.

Agradeço a todos os contribuidores para projetos de software livres e abertos que foram utilizados neste trabalho.

Agradeço a todos meus professores por tudo que aprendi com eles.

Agradeço a todos os acadêmicos e pensadores cujas ideias serviram direta ou indiretamente como base para este trabalho, pois, como disse Sir Isaac Newton, eu só pude ver mas longe pois estou sobre o ombro de gigantes.

Agradeço a UTFPR pelo apoio institucional.

Agradeço aos contribuintes pelo financiamento da UTFPR.

Agradeço ao Mestre Larry Gonick, autor de The Cartoon Guide to Computer Science, e Vilmar Pedro Votre e Eduardo Lamounier Barbieri, tradutores, por me inspirarem a entrar no caminho que me levou até aqui.

Agradeço a todos os outros artistas que com sua arte me inspiraram a ser o melhor que eu poderia ser.

Agradeço a meus amigos e colegas pelo apoio.

Agradeço muito a toda a minha família por tudo que fizeram por mim.

Agradeço a todos os que contribuíram para a realização deste projeto.

Agradeço a todos os que me apoiaram nesta jornada, os que ainda estão aqui e os que já partiram.

“Logic is the beginning of wisdom,
Valeris, not the end.“
“A lógica é o começo da sabedoria,
Valeris, não o fim.”
(Spock, Star Trek VI: The Undiscovered
Country)

RESUMO

O GANITA é um aplicativo Web para desenhar interativamente e verificar a corretude de tableaux analíticos com fins didáticos. Este software é auxiliar no ensino do método dos tableaux analíticos para uso em disciplina de introdução à lógica de nível de graduação. Esse software consiste em um sistema de desenho da árvore no computador de uma maneira semelhante ao papel, que é convertido para a forma textual a la Fitch e enviado para o software ANITA preexistente de processamento de tableaux analíticos. O ANITA então faz a análise do tableau, e o retorno do ANITA é apresentado ao usuário. O GANITA é um aplicativo Web dinâmico apenas do lado do cliente, executando todo o código no navegador Web do usuário, com o pacote Python ANITA rodando dentro do ambiente especial para Web Pyodide. O software foi desenvolvido utilizando TypeScript, HTML e CSS. O software foi avaliado por professores de lógica e seus comentários registrados.

Palavras-chave: lógica; tableaux analíticos; web; ensino; software.

ABSTRACT

GANITA is a Web application to interactively design and check the correctness of analytic tableaux for educative purposes. This software is supplementary to the teaching of the method of analytic tableaux in the context of introduction to logic courses at the undergraduate level. This software consists of a system for tree drawing on a computer in a similar way to how it is drawn on paper, which is then converted to the textual a la Fitch format and sent to the preexisting analytic tableaux processing software ANITA. ANITA then does the analysis of the tableau, and the result of the analysis is presented to the user. GANITA is a dynamic Web application only on the client side, running all the code in the user's Web browser, with the ANITA Python package running within the special environment for the Web Pyodide. The software was developed using TypeScript, HTML and CSS. The software was evaluated by logic professors and their comments were registered.

Keywords: logic; analytic tableaux; web; teaching; software.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DOM	Document Object Model
JS	JavaScript
HTML	HyperText Markup Language
TS	TypeScript
CSS	Cascading Style Sheets
ANITA	Analytic Tableau Proof Assistant
GANITA	Graphical ANITA
LLVM	LLVM Compiler
Wasm	WebAssembly
API	Application Programming Interface
IA	Inteligência Artificial
PWA	Progressive Web Apps
RWD	Responsive Web Design
GUI	Graphical User Interface

SUMÁRIO

1 INTRODUÇÃO

Este relatório técnico trata de um sistema auxiliar ao ensino do método dos tableaux analíticos (ver seção 2.2) (Smullyan, 1995) de forma digital, baseando-se no software preexistente de ensino ANITA (ver seção 1.2.1) que só permite o uso da forma textual a la Fitch (ver seção 2.2.1) de representação de árvore, desenvolvemos um novo software que permite o desenho gráfico da árvore de maneira próxima ao papel, mas permitindo retorno imediato ao aluno se este realizou ou não o desenvolvimento do método corretamente.

O sistema está disponível para uso online[1], e o seu código-fonte está disponível no site GitHub[2].

Em comparação a outros softwares semelhantes, além de permitir o uso de desenho gráfico próximo ao feito com papel, também buscamos construir um sistema que permita ao aluno fazer exercícios e corrigi-los com propósito didático, ao invés de gerar a árvore automaticamente a partir das premissas. Acreditamos que um sistema interativo tem potencial de ser instrutivo para os estudantes com base no discutido na seção 2.3.

Projetamos este sistema para ser viável para uso como parte de disciplinas de lógica de nível de graduação. Para tal, implementamos a interface do sistema para Web.

1.1 Estrutura do texto

Este texto está organizado da seguinte forma: O capítulo 1 introduz o trabalho, seus objetivos, justificativa e discute e compara com outros trabalhos anteriores. O capítulo 2 apresenta a fundamentação teórica do projeto, discutindo tanto aspectos mais abstratos ou teóricos quanto da implementação técnica do sistema. O capítulo 3 apresenta o sistema em si, seu funcionamento, seus componentes e o processo de desenvolvimento. O capítulo 4 apresenta e discute os principais resultados objetivos e subjetivos do projeto. O capítulo 5 discute potenciais melhoramentos e trabalhos futuros e conclui o relatório.

1.2 Trabalhos anteriores

Provadores de teorema interativos, também chamados de assistentes de prova, checam se provas formais fornecidas por um usuário humano em uma linguagem especializada são válidas. Esses softwares podem serem utilizados no ensino de lógica (Minh; Gonnord; Narboux, 2025). Trataremos aqui de apenas alguns exemplos de assistente de prova mais relevantes. Para uma introdução mais ampla do assunto, ver Minh, Gonnord e Narboux (2025).

[1] https://rardiol.github.io/ganita/
[2] https://github.com/rardiol/ganita

1.2.1 ANITA: Analytic Tableau Proof Assistant

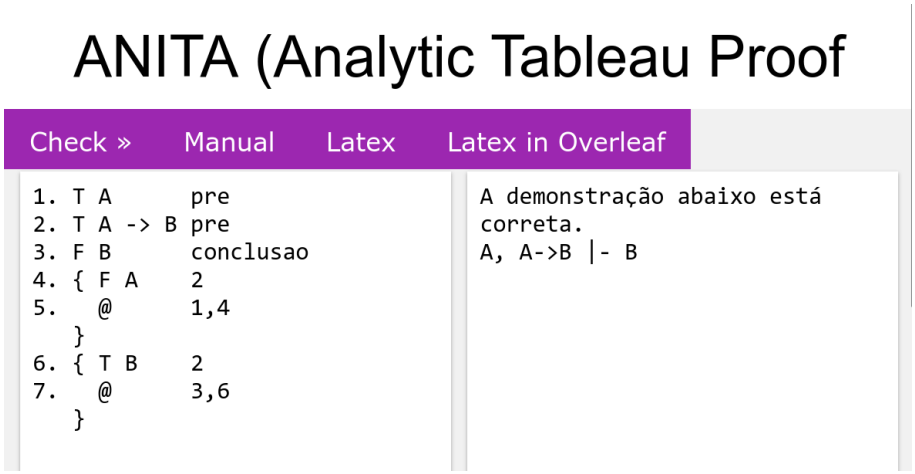
O ANITA (*Analytic Tableau Proof Assistant* ou Assistente de Provas com Tableau Analítico) é um sistema para ensino do método de tableaux analítico. Ele é um software livre escrito em Python e pode ser utilizado na Web ou como um executável. Uma característica importante do ANITA é que ele opera apenas na forma textual das fórmulas, recebendo a entrada do estudante na chamada forma à la Fitch (ver seção 2.2.1) (Vasconcelos, 2023).

A interface Web do software está disponível para uso online[3], e também está disponível uma demostração do seu uso pelo Prof. Dr. Adolfo Neto[4].

O presente projeto foi concebido como uma melhoria do ANITA que permite o desenho gráfico dos tableaux, e utiliza o ANITA como analisador dos tableaux.

O nome GANITA vem da combinação de G de gráfico ou *graphical* com ANITA, que é uma abreviação de *Analytic Tableau Proof Assistant*, ou Assistente de Provas de Tableau Analítico em tradução livre. *Gaṇita* também é a transliteração latina da palavra sânscrita गणित, significando matemática.

Gráfico 1 – Demonstração de uso do ANITA.



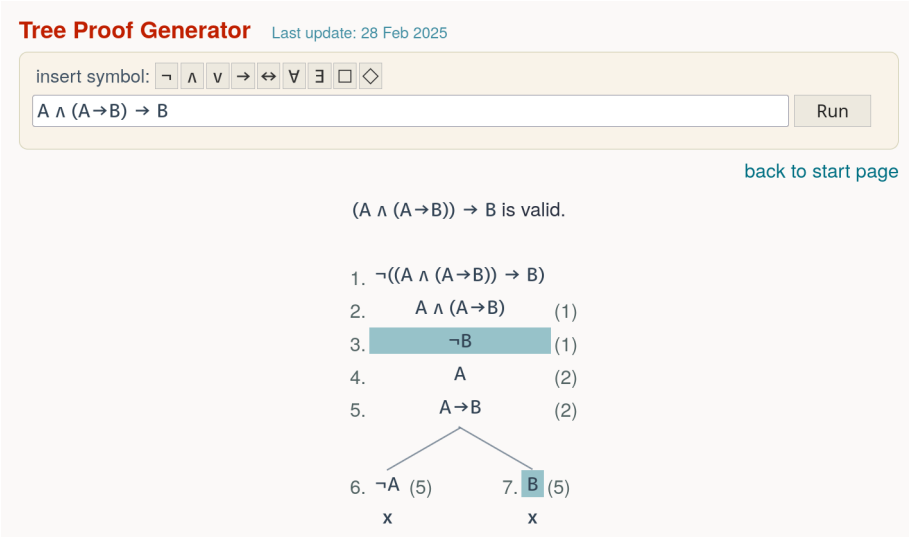
Fonte: Autoria Própria.

1.2.2 Tree Proof Generator

O Tree Proof Generator[5] é um aplicativo Web para lógica formal proposicional, de primeira ordem e para algumas lógicas modais normais. O usuário utiliza o sistema inserindo a fórmula que deseja analisar, e o sistema retorna ou um contramodelo ou uma demonstração na forma de um tableau analítico. Ele não permite o desenvolvimento da árvore pelo usuário (Schwarz, 2024).

[3] https://sistemas.quixada.ufc.br/anita/
[4] https://www.youtube.com/watch?v=CbjYClMLiFI
[5] https://www.umsu.de/trees/

Gráfico 2 – Exemplo de uso do Tree Proof Generator.



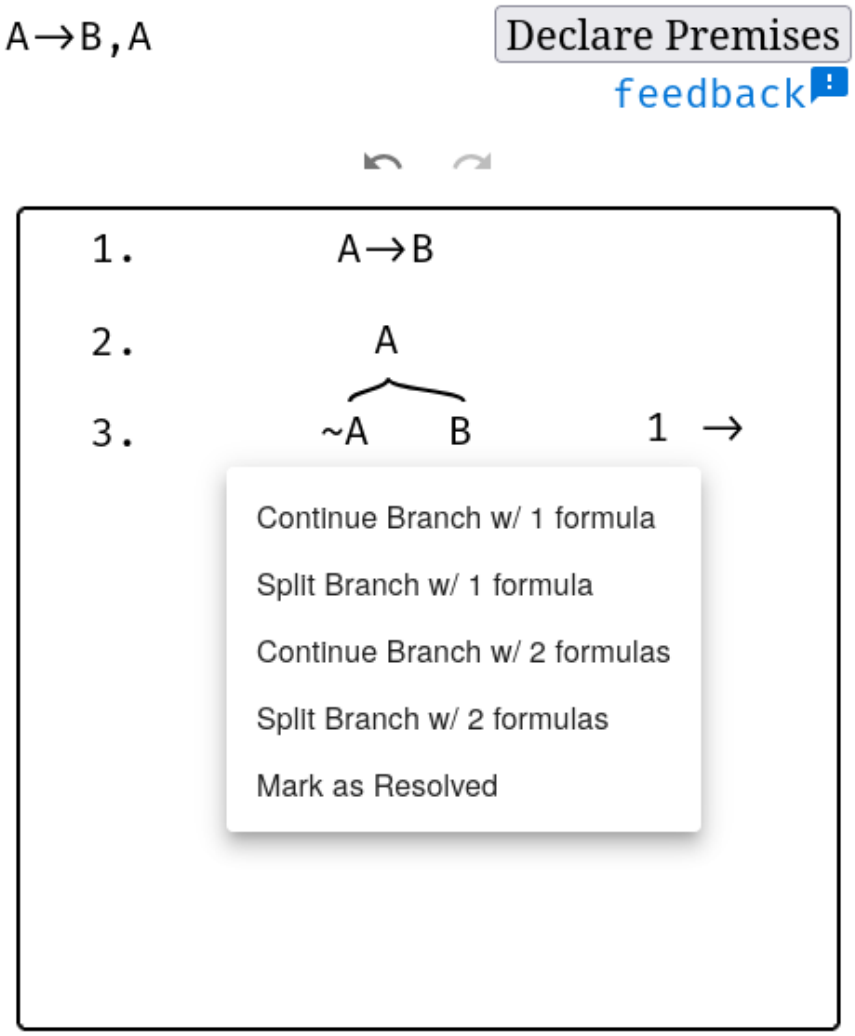
Fonte: Autoria Própria.

1.2.3 Carnap

Carnap[6] é um *framework* para o estudo e ensino de lógica formal desenvolvida em Haskell e utilizável na Web. Entre várias outras funções e utilidades, ela pode ser usada para estudar tableaux analíticos. O aluno inicialmente declara as premissas e subsequentemente interage de forma estruturada, mediante uma lista de opções de aplicação de regras de desenvolvimento, que o aluno precisa complementar corretamente incluindo as fórmulas filhas e as regras aplicadas (Leach-krouse, 2018).

[6] https://carnap.io/srv/doc/truth-tree.md

Gráfico 3 – Exemplo de uso do Carnap.



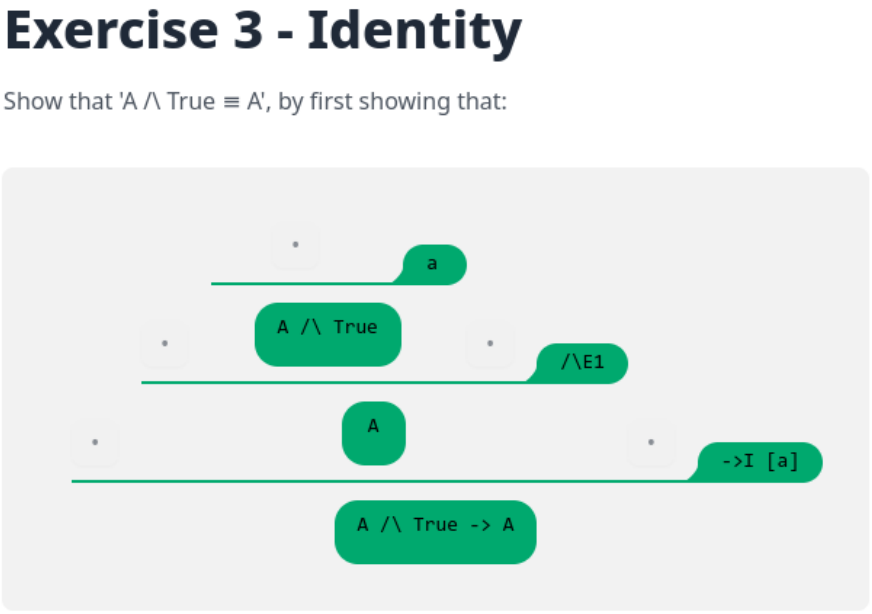
Fonte: Autoria Própria.

1.2.4 OnlineProver

O OnlineProver[7] é um aplicativo Web para ensino de lógica formal, com os alunos realizando um processo iterativo de desenvolvimento da provas de teoremas. Tem objetivos semelhantes ao GANITA com os alunos tendo que especificar as derivações e constructos de regra, mas sem seleção de regras interativas, se aproximando do ensino no papel. A diferença mais significativa é que ao invés dos tableaux analíticos ou de provas ao estilo Fitch, o OnlineProver usa provas derivativas estilo Gentzen sobre sistemas dedutivos definidos pelo usuário (Perháč *et al.*, 2025).

[7] http://onlineprover.com/

Gráfico 4 – Exemplo de uso do OnlineProver.



Fonte: Autoria Própria.

1.3 Justificativa e objetivo

A justificativa principal do projeto são os benefícios educacionais esperados do sistema, que deve se aproximar mais da forma como o método é ensinado tradicionalmente, via desenho da árvore, o que é mais intuitivo e visual, comparado ao estilo textual a la Fitch que é necessário ensinar separadamente para usar o sistema ANITA. O uso de ferramentas Web é desejável para permitir uma máxima facilidade de uso por parte dos alunos, podendo ser utilizado tanto dentro quanto fora do ambiente de aula sem a necessidade de instalação. O uso da ferramente preexistente ANITA vai permitir acelerar o projeto e desenvolver áreas mais relevantes ao ensino. A proposta se baseia em parte em proposta de ex-professor de disciplina de introdução à lógica. Uma justificativa adicional pessoal é o aprendizado de tecnologias importantes como desenvolvimento Web, adicionado ao uso de outros conhecimentos adquiridos ao longo do curso.

O objetivo geral do sistema é criar um sistema online didático que auxilie no ensino do método dos tableaux analíticos. Esse objetivo pode ser separado em dois objetivos específicos:

1. Desenvolver um software para Web que permita desenhar e analisar tableaux analíticos.

2. Obter avaliações do sistema por professor de disciplinas de lógica.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Lógica proposicional e de primeira ordem

O GANITA tem suporte para a lógica proposicional e para a lógica de primeira ordem. A lógica proposicional é um sistema de lógica formal que trata de sentenças formadas por proposições, que podem ser verdadeiras ou falsas, e por conectivos lógicos entre essas proposições. As proposições podem ser afirmações como "O céu é azul", mas em geral são representadas abstratamente por uma letra como A, B, etc. Os conectivos da lógica proposicional são o ou lógico ($\vee$), o e lógico ($\wedge$), a negação ($\neg$) e a condicional material ($\Rightarrow$) (Silva; Finger; Melo, 2006).

Já a lógica de primeira ordem é uma lógica formal que trata de sentenças formadas por termos, subdivididos entre constantes e variáveis, e predicados que são aplicados a esses termos, que também podem ser interpretados como conjuntos. Assim como na lógica proposicional os predicados são conectados para formar fórmulas mais complexas. Além dos conectivos da lógica proposicional que também são aplicáveis à lógica de primeira ordem, usa-se o quantificador existencial ($\exists$) e o quantificador universal ($\forall$) (Shapiro; Kissel, 2024).

2.2 Método dos tableaux analíticos

O GANITA tem o propósito de ensinar o método dos tableaux analíticos, que é um sistema dedutivo que é aplicável à lógica proposicional e à lógica de primeira ordem (ver seção 2.1). Uma descrição informal do uso desse método para a lógica proposicional é que desenha-se uma árvore onde cada nó é uma fórmula lógica. Um sequente é um objeto da forma $\Gamma \vdash A$, onde Γ é o antecedente ou hipótese, A é o consequente ou conclusão, e que significa que o antecedente permite inferir o consequente. Para iniciar a análise de um sequente, escrevemos todas as premissas como filhas em sequência, e ao final insere-se uma negação do que se está tentando provar. O método prossegue seguindo as chamadas regras alfa, beta, delta e gama, que determinam como expandir a árvore. A árvore é expandida até que todos os caminhos desde a raiz até uma folha contenham alguma contradição, caso em que o sequente foi provado, ou que não seja mais possível expandir a árvore, caso em que o caminho desde a raiz sem contradição fornece um contra-exemplo.

Existem algumas variantes no uso do método. Uma delas é o uso de fórmulas marcadas, que são fórmulas precedidas de um simbolo T/V ou F indicando se são verdadeiras ou falsas, ou de fórmulas não marcadas, que usam a negação lógica ($\neg$) para transmitir a mesma informação como parte da fórmula (Smullyan, 1995) (Silva; Finger; Melo, 2006) (Howson, 2005).

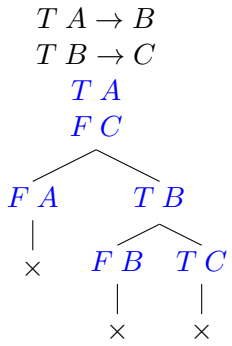
Regras alfa e beta (ver tabela 1) são aplicáveis à lógica proposicional. As regras beta bifurcam a árvore, enquanto que as regras alfa não. A figura 5 é um exemplo de um tableau que prova sua conclusão, e a figura 6 é um exemplo de um tableau que fornece um contramodelo.

Tabela 1 – Regras α e β

α regra	$T\ \varphi \wedge \psi$ | $T\ \varphi$ $T\ \psi$	$F\ \varphi \vee \psi$ | $F\ \varphi$ $F\ \psi$	$F\ \varphi \to \psi$ | $T\ \varphi$ $F\ \psi$	$T\ \neg\varphi$ | $F\ \varphi$	$F\ \neg\varphi$ | $T\ \varphi$
β regra	$F\ \varphi \wedge \psi$ $F\ \varphi \quad F\ \psi$	$T\ \varphi \vee \psi$ $T\ \varphi \quad T\ \psi$	$T\ \varphi \to \psi$ $F\ \varphi \quad T\ \psi$		

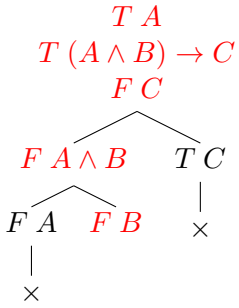
Fonte: Vasconcelos (2023) (Modificado).

Gráfico 5 – Exemplo Tableau $A \to B, B \to C, A \vdash C$



Fonte: Vasconcelos (2023) (Modificado).

Gráfico 6 – Exemplo Tableau $A, A \wedge B \to C \nvdash C$



Contramodelo:
$v(A) = T$ and $v(B) = v(C) = F$

Fonte: Vasconcelos (2023) (Modificado).

Regras gama e delta (ver tabela 2) são aplicáveis à lógica de primeira ordem. Regras gama tratam de substituições enquanto que regras delta instanciam novas variáveis. As figuras 7 e 8 são exemplos de tableaux para lógica de primeira ordem.

Aqui φ_t^x é a expressão obtida da fórmula φ substituindo a variável x, onde quer que ela ocorra livremente em φ, pelo termo t. Por exemplo, $(H(x) \to \forall x M(x))_y^x = (H(y) \to \forall x M(x))$. Podemos dizer que um termo t é substitutível por x em φ se não há uma variável y em t que é capturada por um $\forall y$ (ou $\exists y$) quantificador de φ_t^x. Por exemplo, o termo z é substitutível por y em $\forall x P(x,y)$. Por outro

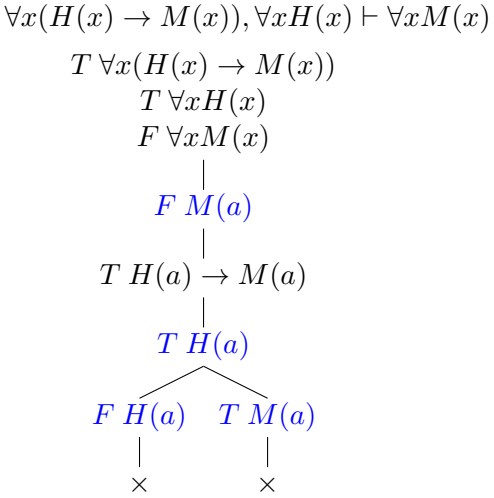
lado, o termo x não é substitutível por y em $\forall x P(x,y)$ (Vasconcelos, 2023)(Tradução própria).

Tabela 2 – Regras γ e δ

γ regra	$T\ \forall x\varphi$ | $T\ \varphi_t^x$ x é substitutível por t em φ	$F\ \exists x\varphi$ | $F\ \varphi_t^x$ x é substitutível por t em φ
δ regra	$F\ \forall x\varphi$ | $F\ \varphi_a^x$ a é uma nova variável	$T\ \exists x\varphi$ | $T\ \varphi_a^x$ a é uma nova variável

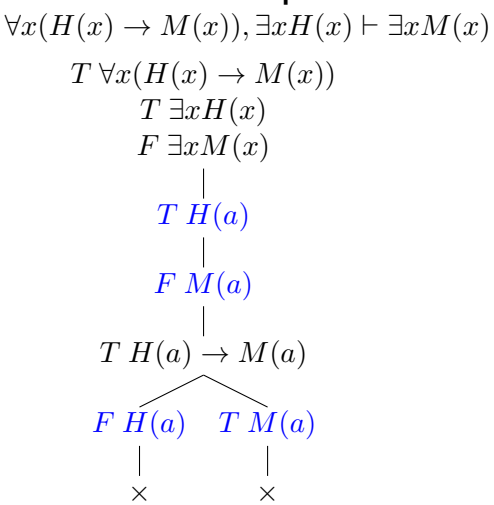
Fonte: Vasconcelos (2023) (Modificado).

Gráfico 7 – Exemplo Tableau



Fonte: Vasconcelos (2023) (Modificado).

Gráfico 8 – Exemplo Tableau



Fonte: Vasconcelos (2023) (Modificado).

2.2.1 Forma a la Fitch

Além da forma gráfica, também é possível utilizar o método dos tableaux analíticos na forma textual a la Fitch. Cada nó da árvore é substituído por uma linha de texto, contendo o número de linha da fórmula, a fórmula em si, e a identificação de que tipo de fórmula se trata. Formulas podem ser premissas, conclusões, aplicações de regras ou fechos. Nesses dois últimos casos, se identifica a linha de origem que justifica a aplicação de regra ou o fecho. Símbolos de abre e fecha chaves são utilizados para denotar bifurcações e a estrutura da árvore (Pelletier; Hazen, 2024).

Segue um exemplo da demonstração de *modus ponens* na forma a la Fitch utilizada pelo ANITA (Vasconcelos, 2023).

```
1.    T A     pre
2.    T A->B  pre
3.    F B     conclusao
4.  { F A     2
5.    @       4,1
    }
6.  { T B     2
7.    @       6,3
    }
```

2.3 Ensino interativo e prático

O GANITA é um software de ensino interativo para uso acadêmico. Existem evidências que a aprendizagem ativa tem resultados melhores do que o ensino passivo no ensino superior nas áreas de ciência, tecnologia, engenharia e matemática (Freeman *et al.*, 2014). Para utilizar o ensino ativo, é necessário um planejamento e preparação adequada da aula, e pode-se utilizar tanto práticas não digitais como no computador.

Na área da matemática, é comum a forma de ensino na qual o professor pede aos alunos que resolvam exercícios de fixação e treino após o ensino de alguma técnica matemática. Esses exercícios também podem tomar uma forma digital, que permite a correção automática e imediata, inclusive fora do ambiente de aula.

O ensino da ciência da computação teórica normalmente tem como pré-requisito o ensino de matemática, o que costuma apresentar dificuldades para alunos de computação. Existem evidências que o uso de técnicas pedagógicas específicas podem ser mais eficazes, como o uso de tentativa e erro como parte do ensino (Perháč *et al.*, 2025).

2.4 Desenvolvimento Web

Desenvolveremos este trabalho para a plataforma Web, ou *World Wide Web* por extenso. A Web normalmente opera no modelo cliente-servidor, com navegadores Web fazendo o papel de cliente e se comunicando com servidores Web, dos quais recebem sites Web.

Sites Web podem serem classificados de várias formas, uma das quais é entre sites estáticos, que não tem suporte a interação do usuário e são fornecidas pelo servidor como estão, sem processamento ou com processamento mínimo no servidor, e sites dinâmicos, em que a página Web é modificada e não meramente acessada como foi armazenada (Malik, 2025). Em sites dinâmicos, podemos fazer a distinção entre aspectos dinâmicos no lado do cliente *(client-side)*, que fornecem interatividade e possivelmente comunicação com o servidor com o código rodando no navegador Web cliente, e do lado do servidor*(server-side)*, que são sites que são modificados no servidor antes de serem enviados ao cliente (Cloudflare, Inc., 2025). Um site pode ter aspectos dinâmicos do lado do cliente, do lado do servidor, de ambos os lados, ou de nenhum.

Existe também o conceito de aplicativo Web, ou *Web app*, que se refere a software desenvolvido sobre a plataforma Web, e é aplicado a sites dinâmicos particularmente complexos, interativos e que podem substituir software nativo tradicional (Contribuidores MDN, 2025d) (Amazon Web Services, Inc. ou afliadas, 2025).

2.5 Tipagem estática, dinâmica e gradual

Utilizamos a linguagem de programação TypeScript para desenvolvimento, que é essencialmente o JavaScript com a adição de tipagem gradual. Os sistemas de tipagem das linguagens de programação podem ser classificados entre os de tipagem dinâmica e os de tipagem estática, com sistemas de tipagem gradual como intermediários entre os dois. Linguagens estáticas tem sua consistência checada em tempo de compilação, detectando alguns tipos de erro antes da execução. Linguagens dinâmicas são executadas sem as mesmas verificações, facilitando o uso interativo. Esses dois extremos tem vantagens e desvantagens, levando aos sistemas graduais, que permitem misturar ambos os sistemas na mesma base de código. Um dos métodos para permitir isso é a "tipagem opcional", que consiste em checar as partes estáticas em tempo de compilação, e posteriormente executar o programa resultante como se fosse escrito em uma linguagem dinâmica. Para um tratamento mais detalhado do assunto, ver New, Licata e Ahmed (2021).

Outra conexão interessante dos sistemas de tipagem com o trabalho é o isomorfismo de Curry-Howard, que permite encontrar uma equivalência entre as formulas da lógica e os tipos de linguagens de programação (Clarkson *et al.*, 2025).

2.6 Sistemas de construção

Utilizamos o webpack para realizar o empacotamento do projeto. Sistemas de construção (*build systems*) são softwares que facilitam o processo de transformar código fonte e outros recursos que compõem um certo sistema e preparar o lançamento de uma certa versão deste sistema (Google; Contribuidores Bazel, 2025). Sistemas de construção normalmente executam tarefas como automaticamente compilar múltiplos arquivos, linkar o código objeto resultante, pré-processar outros tipos de arquivo conforme necessário, copiar arquivos do repositório fonte para o local correto de destino, entre outras muitas tarefas.

3 DESENVOLVIMENTO

Neste capitulo apresentaremos os principais aspectos de como o sistema foi desenvolvido e como ele funciona.

3.1 Escopo

Desenvolvimento da interface Web GANITA e aspectos relacionados, em especial a parte de desenho gráfico dos tableaux, e integração com o ANITA. O ANITA, com suporte para tableaux analíticos a la Fitch (textual), já existe e é usado como parte do GANITA. Como um instrumento auxiliar ao ensino de disciplina de graduação, o sistema deve ser possível de usar com um nível pequeno de instrução por parte do professor, mas não tem a intenção de ensinar lógica ou o método dos tableaux analíticos de forma independente. O sistema não é destinado ao uso em dispositivos móveis ou com tela pequena.

3.2 Metodologia

Desenvolvemos o projeto iterativamente e usando prototipação, orientando o desenvolvimento seguindo *feedback* do professor orientador ao longo do processo para elicitar requisitos e possíveis melhorias.

Ao final do desenvolvimento principal, para obtermos mais diversidade de perspectivas e possíveis melhorias e requisitos finais, buscamos outros professores da área de lógica. Realizamos uma entrevista presencial com professor, onde apresentamos o sistema, observamos o uso e coletamos comentários gerais e sugestões. Nós, incluindo o professor orientador, também solicitamos comentários abertos via email sobre o sistema de outros professores da área.

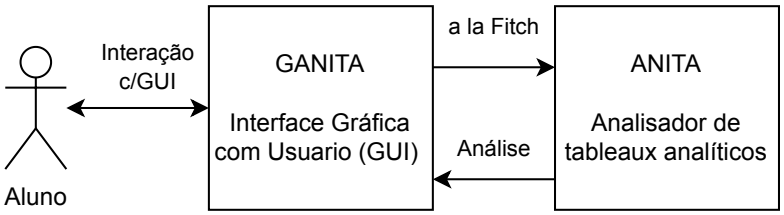
Um primeiro protótipo do sistema foi desenvolvido utilizando o software de desenho de diagramas Dia (Contribuidores Dia, 2023), um conversor escrito em Python e a interface Web do ANITA. O desenvolvimento efetivo do GANITA foi para a plataforma Web, se integrando ao ANITA, que não teve que ser modificado. Utilizamos bibliotecas e ferramentas para suporte ao desenvolvimento Web.

3.3 Modelagem do sistema

O sistema consiste primordialmente de uma interface Web, compreendendo HTML, JavaScript e CSS (ver seção 3.5.1), que permite desenhar interativamente um tableau analítico na forma de árvore, e um conversor dessa árvore, também executando em JavaScript, para a forma a la Fitch textual. Esse texto é utilizado como entrada no ANITA, e o resultado da análise deste é apresentado na interface. A figura 9 é um esquema simplificado do sistema, e a figura 11 é um esboço simplificado da interface deste.

Também compõe o projeto um pequeno manual para o usuário integrado à interface, explicando como utilizar a interface e algumas peculiaridades do uso do sistema.

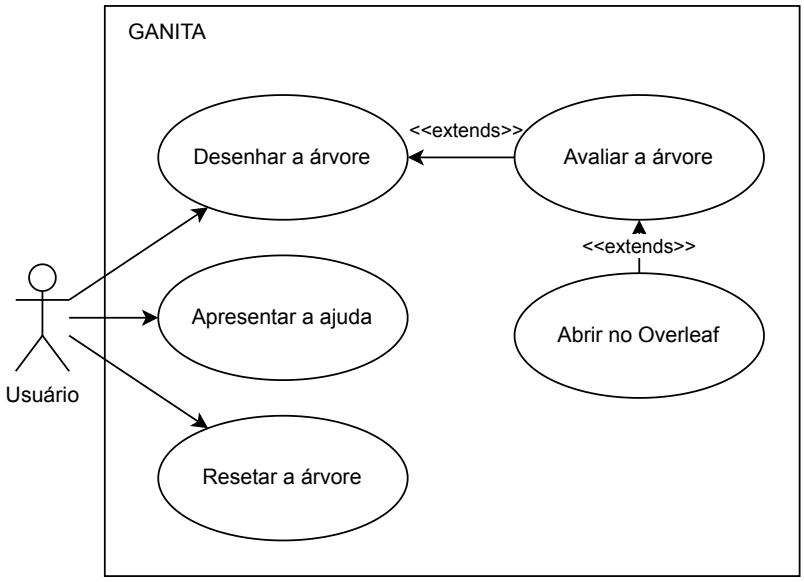
Gráfico 9 – Esquema do sistema



Fonte: Autoria própria (2025).

Os casos de uso do sistema são apresentados na forma de diagrama na figura 10. O principal caso de uso do sistema é o de desenhar a árvore, que permite desenhar o tableau graficamente, arrastando o mouse para formar as conexões entre nós/formulas e digitando para inserir as fórmulas individuais, como apresentado mais extensamente na seção 3.4. Esse caso de uso é estendido pela funcionalidade de realizar a análise do tableau, cujo resultado é apresentado ao usuário. Após a análise, também é possível exportar a árvore para o Overleaf. Também são casos de uso a função de resetar a árvore, que retorna o sistema ao seu estado inicial, e o de apresentar a ajuda do sistema ao usuário.

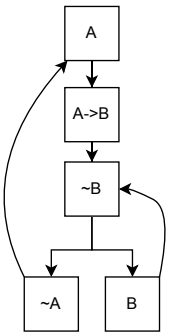
Gráfico 10 – Diagrama de casos de uso do sistema



Fonte: Autoria própria (2025).

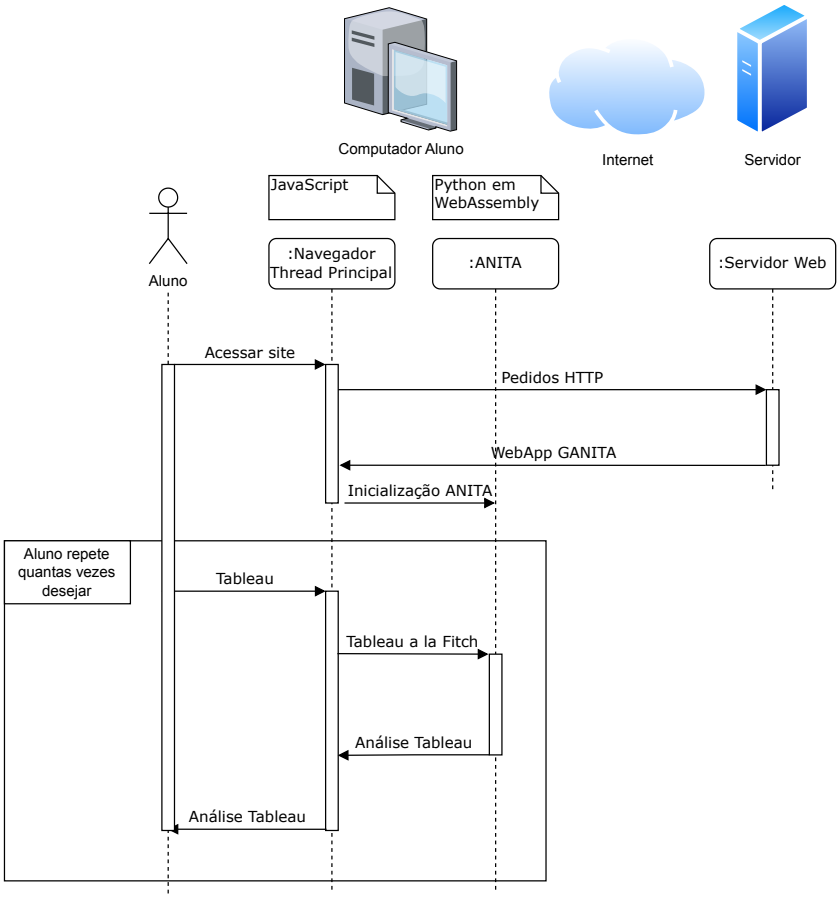
A última parte do projeto é o novo contexto de execução do ANITA, que deixa de ser um aplicativo Web dinâmico do lado do servidor e passa a rodar dentro do navegador Web. O GANITA é portanto um aplicativo Web dinâmico apenas do lado do cliente, sendo do ponto de vista do servidor um site estático, não havendo comunicação do cliente com o servidor após o carregamento da página. O funcionamento do GANITA e do ANITA são esquematizados nas figuras 12 e 13 respectivamente para comparação.

Gráfico 11 – Esboço inicial da interface do sistema

	1. T A pre 2. T A->B pre 3. F B conclusao 4. { F A 2 5. @ 1,4 } 6. { T B 2 7. @ 3,6 }	A demonstração abaixo está correta. A, A->B |- B
Entrada Gráfica	Entrada ANITA textual (autogerada)	Resultado ANITA textual

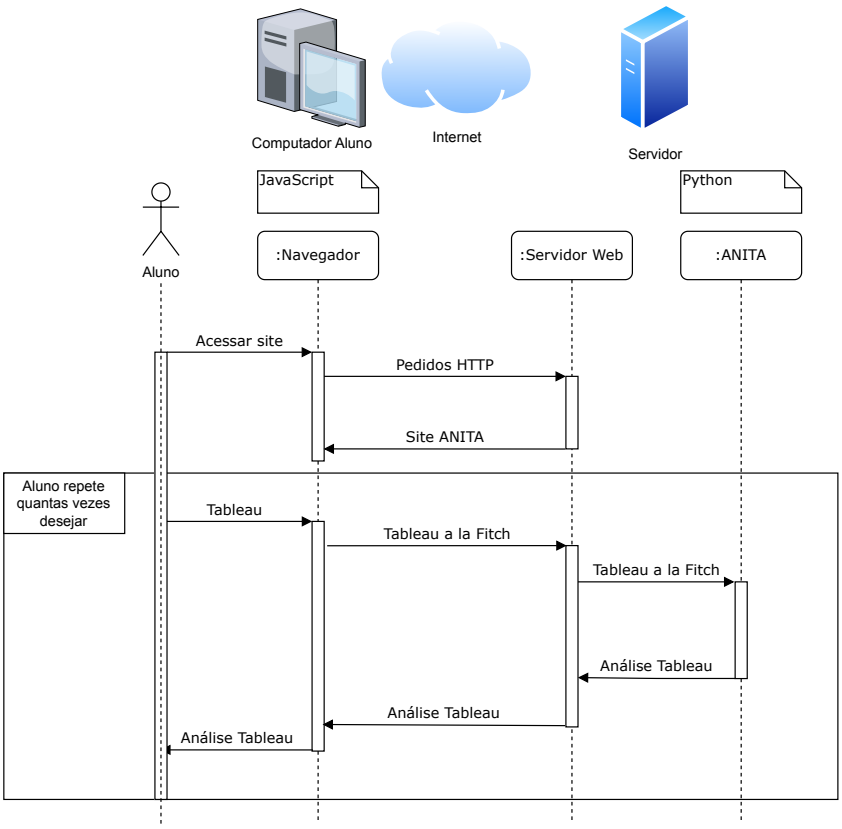
Fonte: Autoria própria (2025).

Gráfico 12 – Funcionamento do GANITA



Fonte: Autoria própria (2025).

Gráfico 13 – Funcionamento do ANITA para Web para comparação



Fonte: Autoria própria (2025).

O ANITA também permite exportar a árvore para o formato LaTeX (The LaTeX Project, 2025) e abrir ela automaticamente no site de edição e colaboração com arquivos LaTeX Overleaf[1], funcionalidade que também implementamos no GANITA.

3.3.1 Requisitos

3.3.1.1 Requisitos funcionais

1. **RF1** Permitir desenhar graficamente uma árvore de tableau analítico.
2. **RF2** Permitir corrigir erros e alterar a árvore.
3. **RF3** Mostrar a forma a la Fitch da árvore atual.
4. **RF4** Informar textualmente se a árvore inserida está correta ou incorreta.
5. **RF5** Informar textualmente o erro da árvore inserida, caso exista.
6. **RF6** Obter a forma LaTeX da árvore.

[1] https://www.overleaf.com/

7. **RF7** Exportar a forma LaTeX da árvore para o Overleaf.
8. **RF8** Permitir reinicializar a árvore para seu estado inicial.

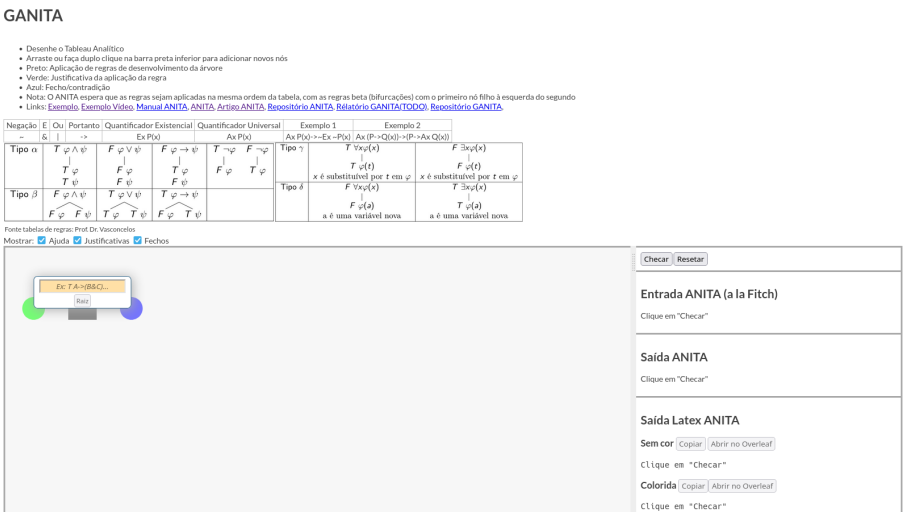
3.3.1.2 Requisitos não funcionais

1. **RNF1** Ser usável via interface Web.
2. **RNF2** Ser possível de instalar em servidor de maneira simples.
3. **RNF3** Ser livre para uso.
4. **RNF4** Não ter custos para uso além dos de hospedagem.
5. **RNF5** Ser integrado ao sistema ANITA.
6. **RNF6** Conter instruções de uso básicas.

3.4 Apresentação do sistema

O sistema inicia mostrando uma tela semelhante a figura 14, e tem uma seção de ajuda inicial (ver figura 15), instruindo o usuário sobre como usar o sistema. Reiterando o que foi dito na seção de escopo, essas instruções tratam primordialmente de como usar o GANITA, não do método dos tableaux analíticos em si. Abaixo da ajuda, a área principal do GANITA se divide entre a área de desenho da árvore à esquerda (ver figura 17), e a área de entrada e saída textual do ANITA à direita (ver figura 18).

Gráfico 14 – Tela inicial ao carregar o sistema



Fonte: Autoria própria (2025).

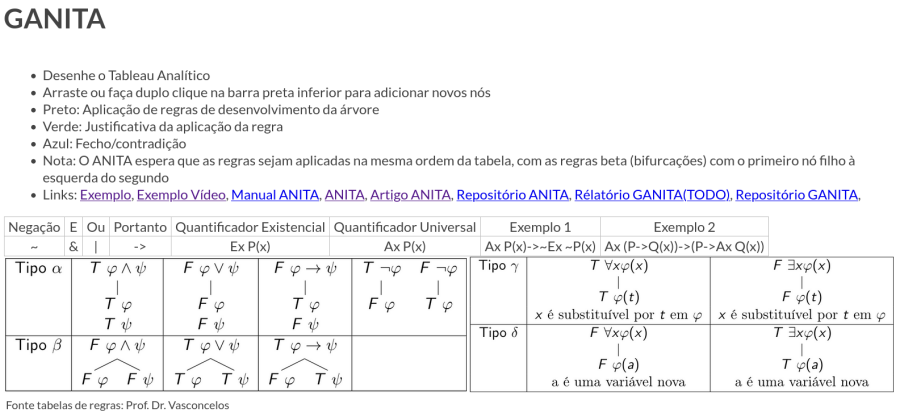
A árvore contem nós, com esses nós possuindo "pontos finais", ou *endpoints* na terminologia do jsPlumb (ver seção 3.5.4.1), que são onde as flechas ou conexões se conectam.

Tabela 3 – Símbolos utilizados

Não	E	Ou	Se então	Para todo	Existe um
∼	&	|	->	Ax P(x)	Ex P(x)

Fonte: Autoria própria (2025).

Gráfico 15 – Detalhe da seção de ajuda ao usuário



Fonte: Autoria própria (2025).

Essas flechas ou conexões então ligam os nós. Tanto as conexões quanto os pontos finais são de 3 tipos diferentes e correspondentes, cada um com sua cor.

A árvore inicia apenas com o nó raiz na tela. Cada nó tem:

- Caixa de texto a ser preenchida com a fórmula desse nó. Ver tabela 3 para os símbolos utilizados.
- Botão de fechar o nó.
- Linha(s) da entrada do ANITA a que esse nó corresponde.
- Tipo de linha a que esse nó corresponde: premissa, conclusão, fecho ou aplicação de regra.
- 3 pontos finais destino, um de cada tipo.
- 3 pontos finais fonte, um de cada tipo.

A exceção é o nó raiz, que já está presente na inicialização. O nó raiz não tem pontos finais fonte pois ele é necessariamente uma premissa, e também não pode ser fechado.

Os nós se conectam através de flechas ou conexões de três tipos diferentes, identificados por suas cores. Flechas pretas indicam a estrutura principal da árvore se desenvolvendo, de cima para baixo. Flechas verdes, as justificativas, indicam que nó é a fonte da regra que está sendo aplicada, indo de baixo para cima. Flechas azuis indicam os fechos, ou contradições, ligando dois nós contraditórios. As conexões só podem ser formadas entre pontos finais de tipos iguais.

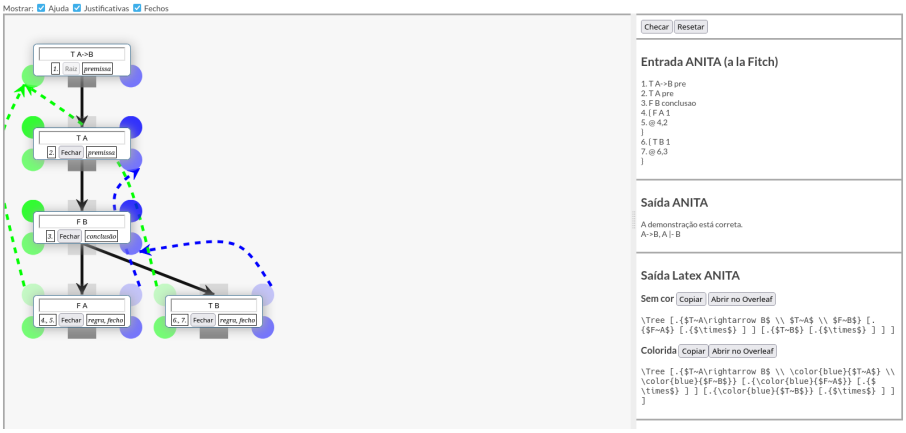
O usuário então constrói a árvore, puxando e arrastando para formar as conexões corretas. Novos nós vazios são automaticamente criados quando se arrasta uma nova conexão desde um ponto final fonte preto, ou quando se faz duplo clique em um ponto final fonte preto.

O site também tem um exemplo de uso em imagem[2] e vídeo[3] para demonstração.

Depois de desenhar a árvore, o aluno pode clicar no botão de checar, que executa a verificação da árvore, e ver a resposta na coluna à direita. Também é possível clicar no botão de resetar para limpar a árvore.

A figura 16 apresenta um exemplo simples de uso do GANITA, a demonstração da regra de *Modus Ponens*. As figuras 17 e 18 apresentam as áreas de desenho e de entrada/saída do ANITA respectivamente. A figura 19 demonstra a funcionalidade de esconder algumas conexões para maior clareza.

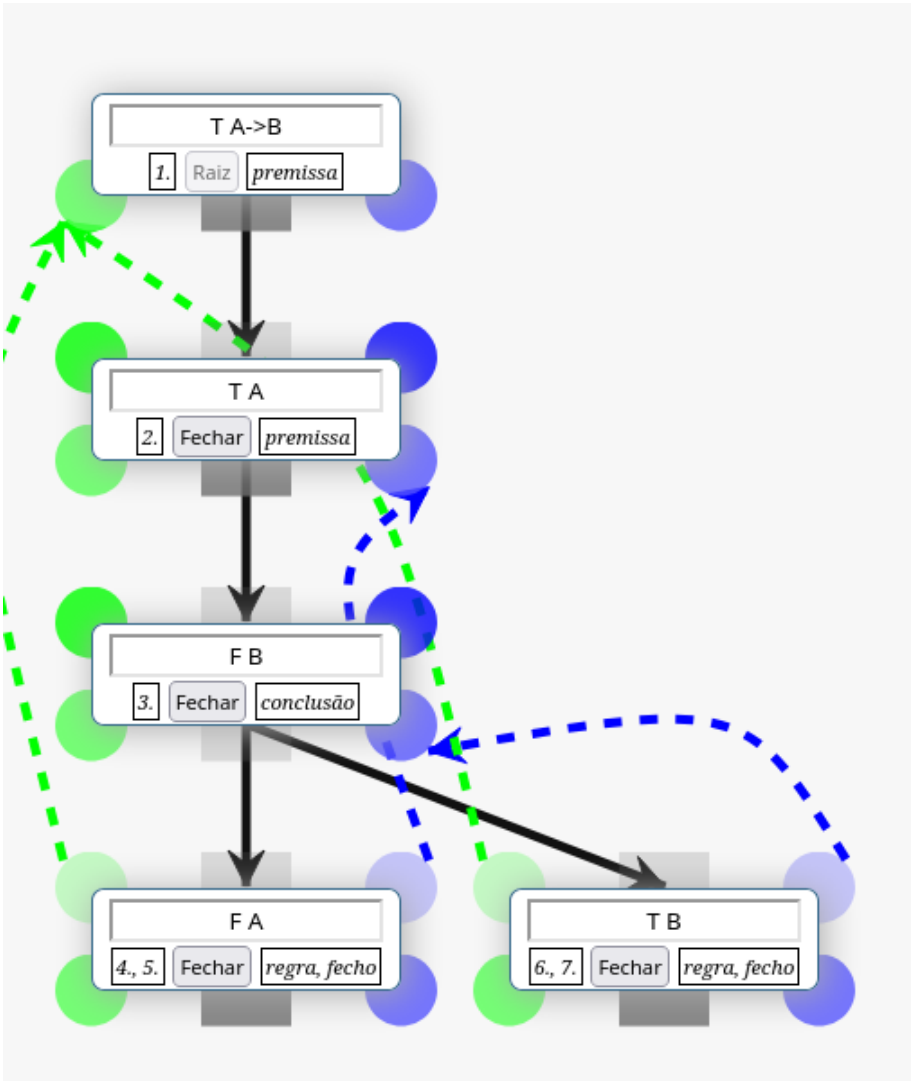
Gráfico 16 – Demonstração de *Modus Ponens*



Fonte: Autoria própria (2025).

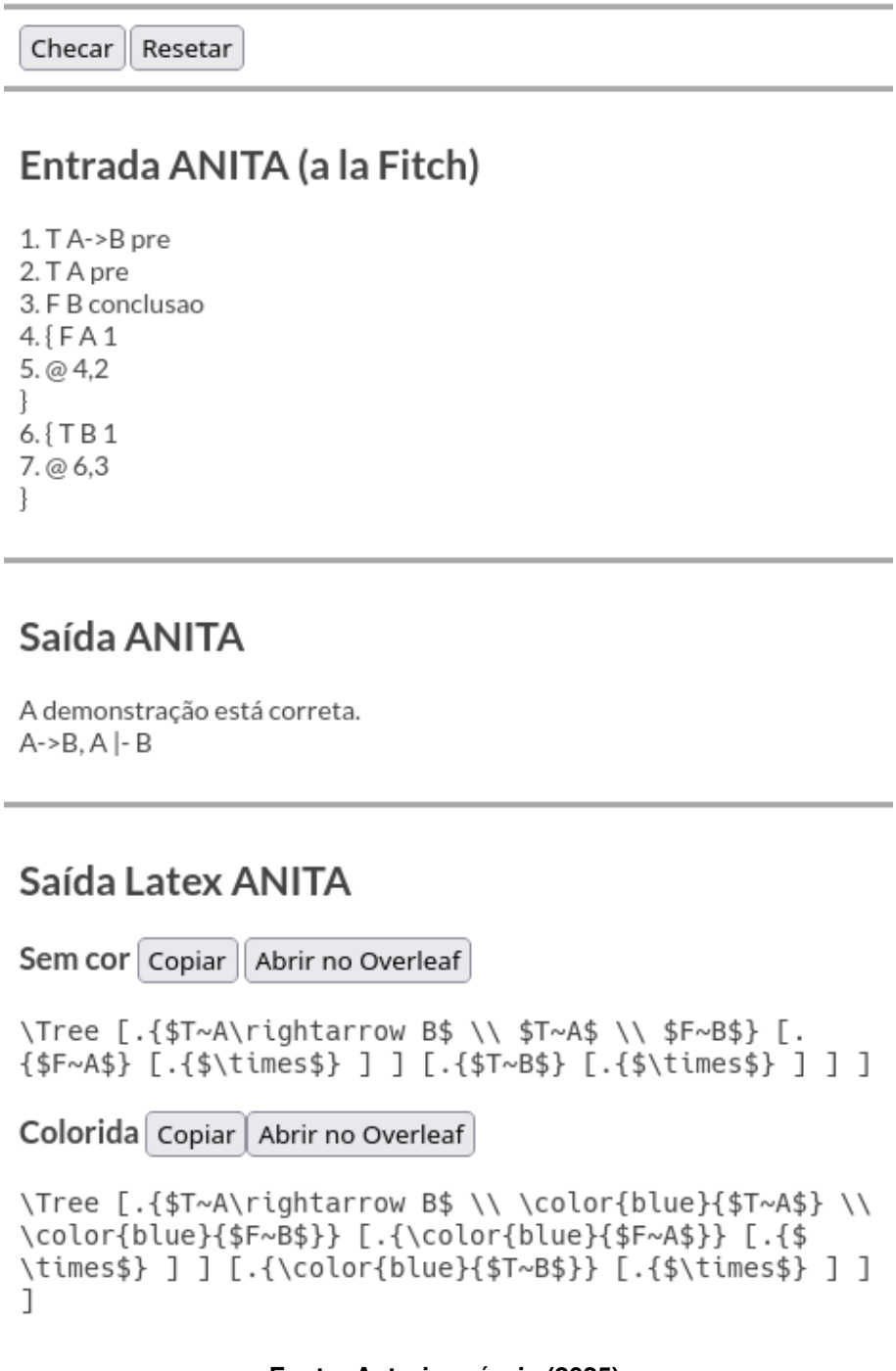
[2] https://rardiol.github.io/ganita/exemplo.png
[3] https://rardiol.github.io/ganita/exemplo.mp4

Gráfico 17 – Detalhe da árvore de *Modus Ponens*



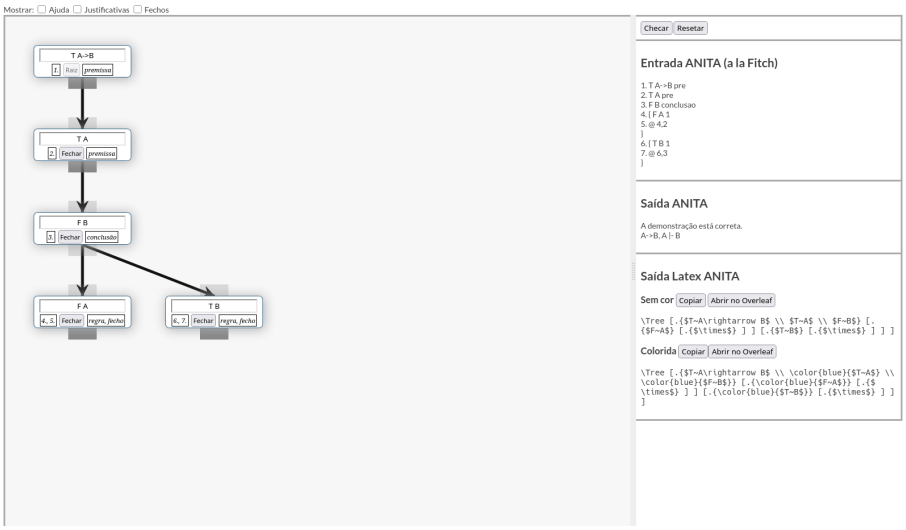
Fonte: Autoria própria (2025).

Gráfico 18 – Detalhe da seção de entrada e saída do ANITA, mostrando a forma a la Fitch, o resultado e as formas LaTeX da árvore



Fonte: Autoria própria (2025).

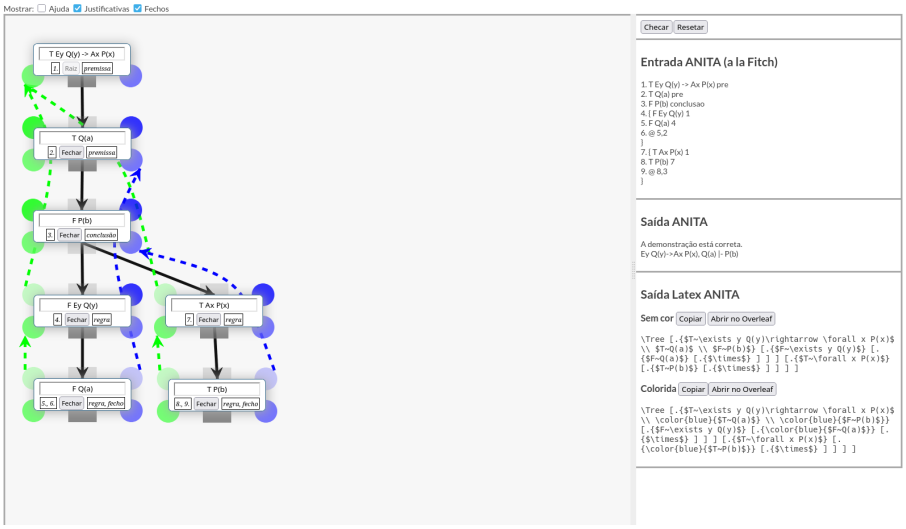
Gráfico 19 – Demonstração da capacidade de esconder as flechas de justificativa e fecho para maior clareza do desenho



Fonte: Autoria própria (2025).

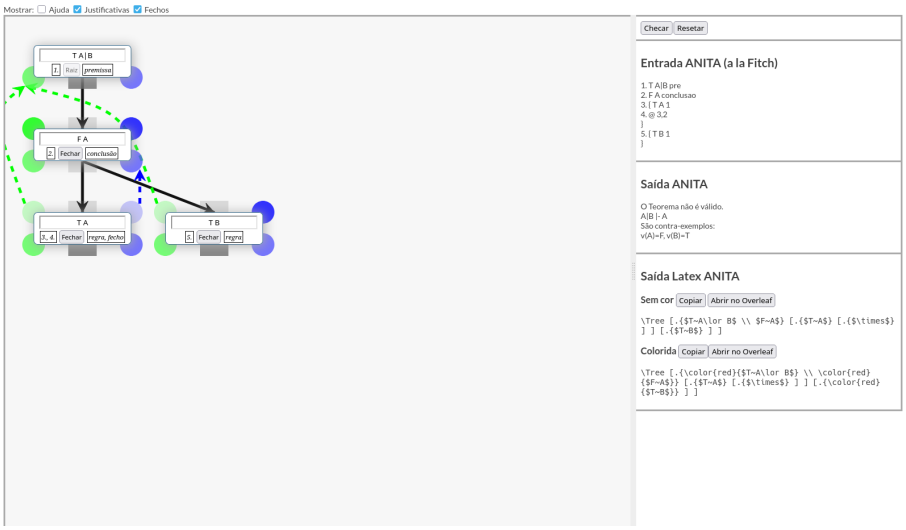
A figura 20 demonstra o uso do GANITA para lógica de primeira ordem. A figura 21 demonstra um caso onde o GANITA foi utilizado com um tableau que não prova sua conclusão e fornece um contraexemplo do tableau.

Gráfico 20 – Demonstração do uso de lógica de primeira ordem



Fonte: Autoria própria (2025).

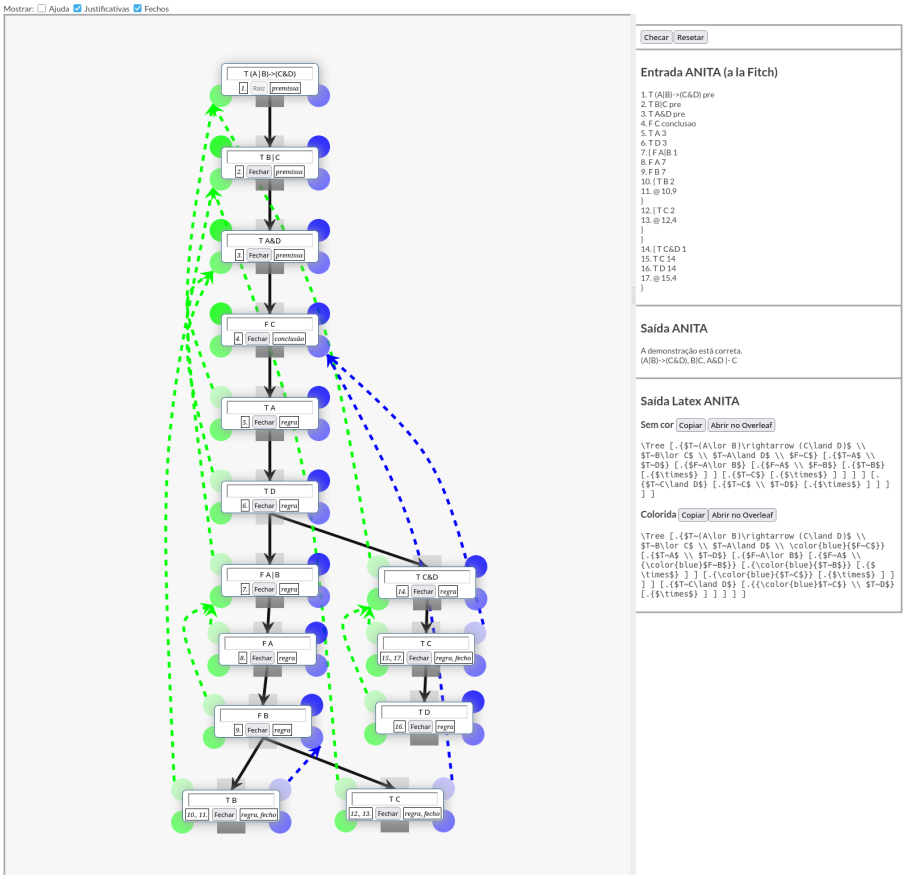
Gráfico 21 – Demonstração de um tableaux com contraexemplo



Fonte: Autoria própria (2025).

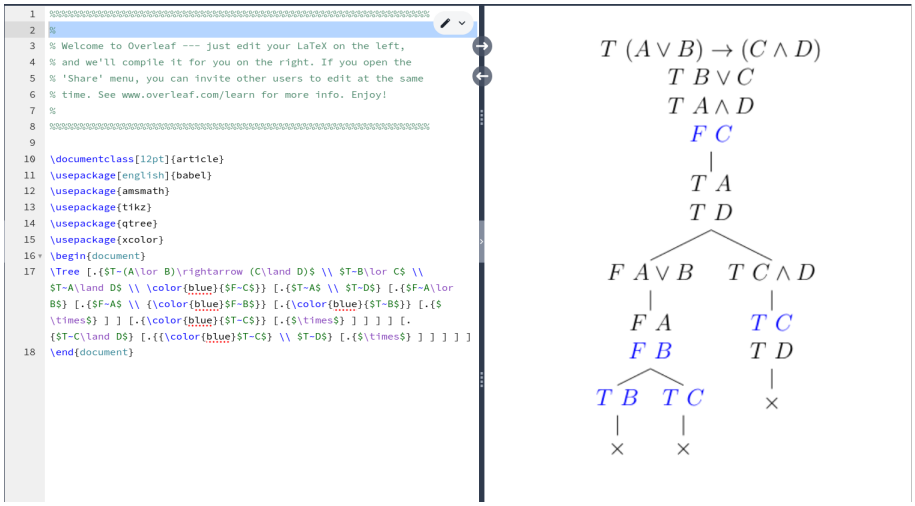
A figura 22 mostra um tableau relativamente grande sendo demonstrado no GANITA. A figura 23 mostra o mesmo tableau na forma LaTeX aberto no Overleaf, utilizando os botões da interface do GANITA que permitem copiar ou abrir diretamente no Overleaf as versões LaTeX da árvore geradas pelo ANITA.

Gráfico 22 – Demonstração de um tableau analítico grande



Fonte: Autoria própria (2025).

Gráfico 23 – Demonstração do gráfico 22 aberto no Overleaf



Fonte: Autoria própria (2025).

3.5 Implementação do sistema

Nesta seção discutiremos os detalhes técnicos de como o GANITA foi desenvolvido, incluindo as funcionalidades, ferramentas, bibliotecas e tecnologias envolvidas.

3.5.1 Tecnologias e linguagens básicas Web

O GANITA é uma aplicação que desenvolvemos para a plataforma Web, utilizando as tecnologias padrão desta plataforma, além de várias outras tecnologias adicionais. A plataforma Web básica utiliza três linguagens principais: HTML, a linguagem de marcação que define o conteúdo da página, CSS, a linguagem de folhas de estilos que determina a forma da apresentação do conteúdo, e o JavaScript, a linguagem de programação nativa original da Web. A isso se somam um grande número de Web APIs, como a *Document Object Model* (DOM) que é utilizado para navegar e manipular código HTML (Contribuidores MDN, 2025b). Para interagir com essas linguagens, existem dois softwares principais, o navegador Web do usuário, que no caso do GANITA pode ser qualquer navegador moderno, e um servidor, sobre o qual o GANITA, por não ser dinâmico do lado do servidor, não impõe exigências adicionais além do simples fornecimento de arquivos solicitados.

Optamos por não utilizar um *framework* Web para o desenvolvimento com o intuito de manter o projeto mais simples e sob controle direto, além de ser mais instrutivo para nós. Consequentemente, escrevemos diretamente os códigos HTML e CSS do GANITA, utilizando diretamente as funcionalidades destes, incluindo funcionalidades modernas.

3.5.2 Split.js

Split.js é uma pequena biblioteca JavaScript que com o conteúdo da página separado entre várias linhas e colunas permite ao usuário controlar divisores flexíveis e mudar o tamanho de cada área subdividida (Cahill, 2021). Utilizamos o Split.js para fazer a separação entre a área da árvore e da saída e entrada do ANITA de forma flexível, permitindo aumentar e diminuir o tamanho conforme necessidade do usuário.

3.5.3 Ferramentas de desenvolvimento

Nesta seção discutiremos algumas ferramentas que foram utilizadas no processo de desenvolvimento, mas de um ponto de vista estrito não fazem parte do produto final.

3.5.3.1 TypeScript

O TypeScript é uma linguagem de programação que é um superconjunto do JavaScript e que é transpilada para JavaScript no momento de compilação. O objetivo principal do TypeScript é a adição de tipagem forte e estática, ou mais propriamente tipagem forte, gradual e opcional ao JavaScript (Microsoft, 2025b). O uso desse tipo de tipagem tem a intenção de minimizar o número de bugs, facilitar o desenvolvimento e simplificar mudanças no código sem criar novos bugs, ao mesmo tempo mantendo o máximo de semelhança e compatibilidade com o JavaScript. O TypeScript é altamente popular com a pesquisa State of JS 2024 indicando que 67% dos respondentes utilizam mais TypeScript que JavaScript (Devographics, 2024).

Utilizamos o TypeScript no projeto para acelerar o desenvolvimento e evitar bugs triviais que podem ser descobertos via análise estática. Apenas algumas seções triviais de código foram implementadas em JavaScript diretamente.

3.5.3.2 Visual Studio Code

Utilizamos a *Integrated Development Environment* (IDE) Visual Studio Code, que oferece suporte para as outras tecnologias empregadas como webpack e TypeScript (Microsoft, 2025a).

3.5.3.3 ESLint

O ESLint é uma ferramente do tipo *lint*, que executa análise estática em código JavaScript (OpenJS Foundation; Contribuidores ESLint, 2025). Utilizamos o typescript-eslint, com suporte para TypeScript para buscar por alguns tipos de erro que são programaticamente encontráveis no código e evitar bugs.

3.5.3.4 Node.js e npm

Node.js é um ambiente de execução de JavaScript, criado para permitir o uso do JavaScript fora de um navegador Web, particularmente em servidores. Não utilizamos esse ambiente no projeto, mas o seu gerenciador de pacotes, o npm, que é o gerenciador de pacotes mais utilizado também para desenvolvimento Web *frontend*, foi utilizado para gerenciar os pacotes JavaScript e TypeScript (OpenJS Foundation, 2025).

3.5.4 Desenho da árvore

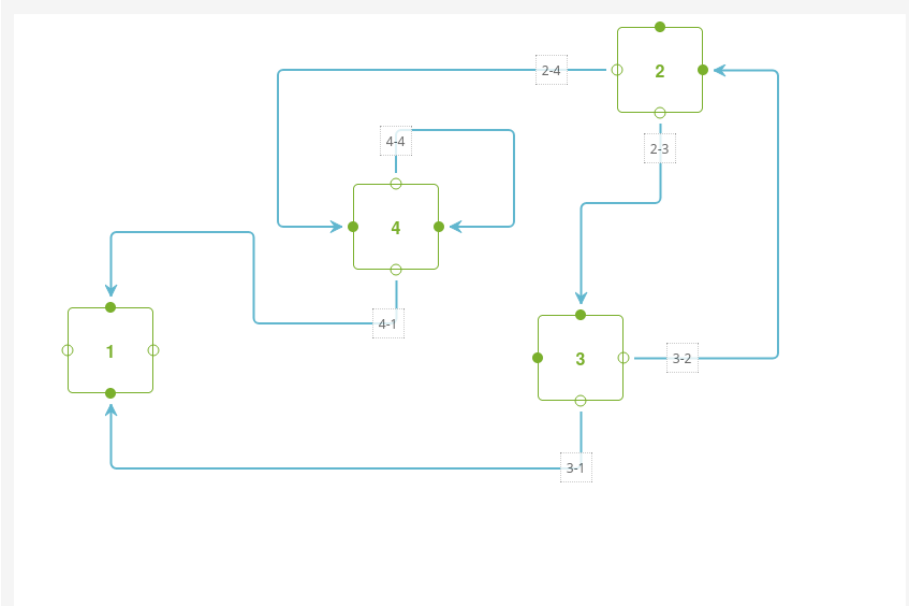
Existem uma enorme variedade de bibliotecas para desenho e visualização de esquemas, gráficos, árvores, fluxogramas, etc para a plataforma Web. A principal biblioteca que decidirmos utilizar é a jsPlumb Community Edition.

3.5.4.1 JsPlumb

A jsPlumb Community Edition é uma biblioteca de software livre para desenvolvimento de interfaces Web com elementos que se movem e se conectam e desconectam, facilitando o desenvolvimento do projeto, que envolve elementos HTML formando uma árvore (jsPlumb, 2024).

Esta é a biblioteca principal que gerencia todos os aspectos da área principal de desenho da árvore, como os nós, pontos finais e conexões.

Gráfico 24 – Exemplo de uso da jsPlumb Community Edition



Fonte: jsPlumb (2024).

O jsPlumb é configurado com um misto de definição de *callbacks*, chamadas de função e de definição de dados de configuração, como por exemplo a listagem de código 1. Nesse

exemplo, pode-se observar a escolha da cor, características estéticas do ponto final, que tipo de conexão são aceitas nele, definição de um *callback*, além de outros detalhes.

Listagem 1 – Definição do ponto final (*Endpoint)* fonte de fecho (Azul)

```
const closureSourceEndpoint: EndpointOptions = {
    endpoint: { type: "Dot", options: { radius: 30 } },
    paintStyle: { fill: "#00F" },
    source: true,
    target: false,
    scope: "closure",
    connectorClass: "closure",
    connectionsDirected: true,
    maxConnections: 1,
    connectorStyle: {
        strokeWidth: 5,
        stroke: "#00F",
        dashstyle: "2 2"
    },
    connector: {
        type: StateMachineConnector.type,
        options: {
            margin: 15,
            curviness: 60,
            proximityLimit: 120,
        }
    },
    cssClass: "closure source",
    // @ts-expect-error
    beforeDrop: myBeforeDrop,
};
```

Fonte: Autoria própria (2025).

3.5.4.2 Bibliotecas e ferramentas não utilizadas

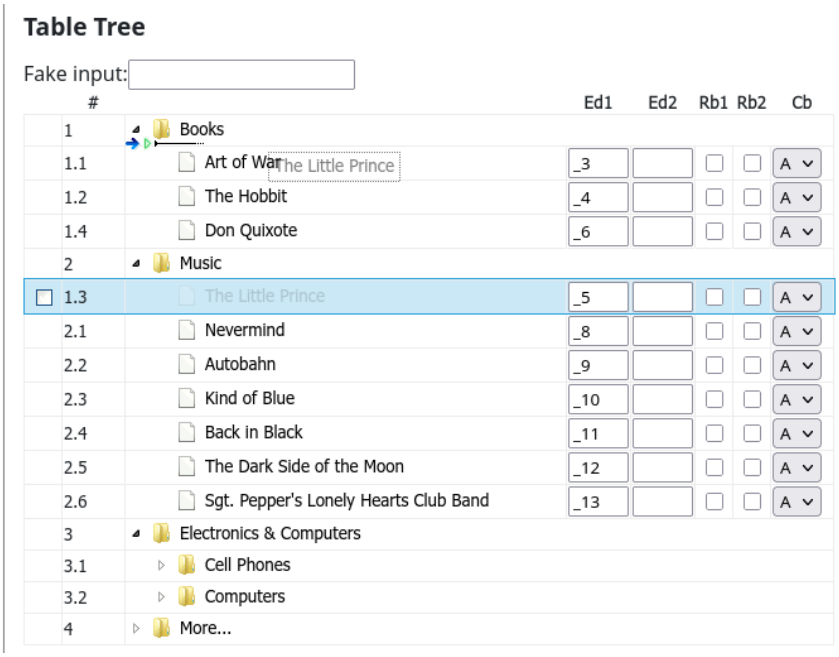
Além do jsPlumb, avaliamos outras bibliotecas para uso no projeto para cumprir a função de desenho da árvore, que não foram utilizadas. Segue uma análise rápida destas e a razão pela qual se decidiu utilizar o jsPlumb.

Um grupo de bibliotecas, bastante semelhantes ao jsPlumb, são as de interação arrastar e soltar (*drag-and-drop*), que tem muitas funções assemelhadas ao jsPlumb, mas nenhuma dessas oferece o mesmo suporte do jsPlumb ao desenho de ligações configuráveis entre os nós da árvore. São estas bibliotecas o interact.js (Taye Adeyemi, 2025), o Draggrable JS (Shopify, 2025) e o Fabric.js (Contribuidores Fabric.js, 2025).

Outro grupo de bibliotecas são as específicas para a criação ou visualização de árvores. Muitas destas, como a Treant.js, (Peručić, 2021) são apenas para visualização, sendo inadequadas para esse projeto. Outras destas bibliotecas, como por exemplo a Fancytree (Wendt, 2025), são específicas para o modelo de árvore modelada como uma lista com vários níveis de indentação indicando a forma da árvore, semelhante ao funcionamento das listas indentadas

nativas ao HTML. Consideramos esse modelo muito inflexível e distante de como os tableaux analíticos são desenhados.

Gráfico 25 – Exemplo de uso avançado da Fancytree

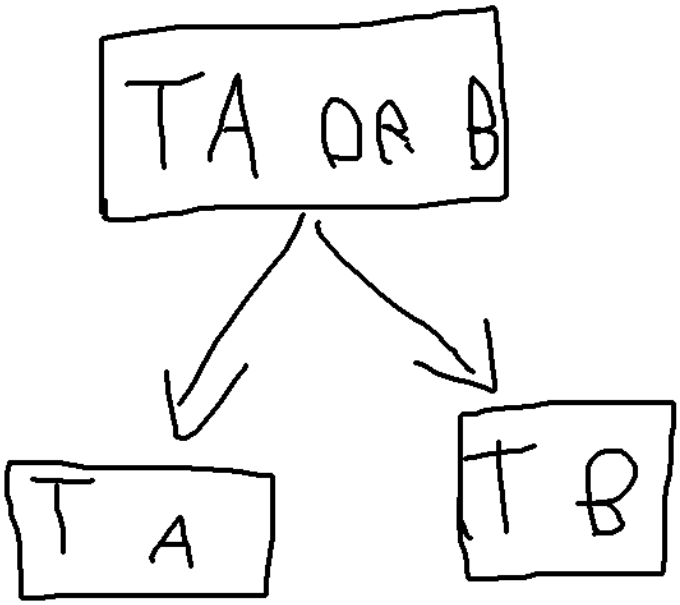


Fonte: Wendt (2025).

Por último, poderíamos utilizar uma biblioteca genérica que funciona sobre o *canvas* HTML, como a Konva (Konva, 2025). Essa opção ofereceria a maior flexibilidade para desenhar a árvore, porém seria necessário implementar um sistema para desenhar a árvore de maneira estruturada.

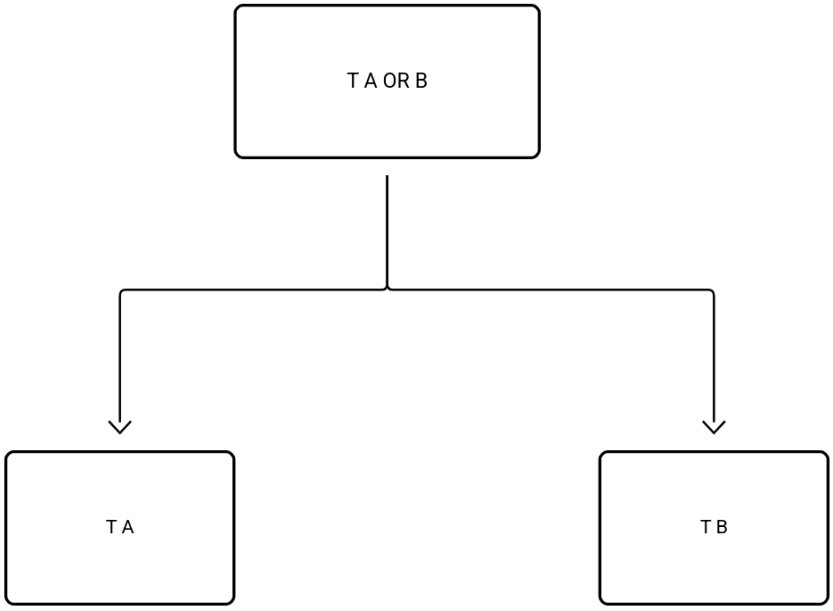
Também existem algumas ferramentas utilizando IA que permitem transformar desenhos livres de várias formas de gráfico em gráficos estruturados. Poderíamos utilizar essas ferramentas no projeto para implementar um sistema onde o aluno desenha livremente a árvore, no computador ou até mesmo no papel, e o sistema posteriormente processaria e avaliaria a árvore. Porém, todas as ferramentas são proprietárias e tem acesso à API pago, inviabilizando o uso no projeto. Além disso, é possível que o uso de um desenho guiado seja mais instrutivo para os alunos (Diagramly, 2025) (dAIgram, 2024). Um exemplo do uso do dAIgram é a figura 27, resultado estruturado convertido a partir da figura 26.

Gráfico 26 – Desenho livre de um tableau



Fonte: Autoria própria (2025).

Gráfico 27 – Gráfico estruturado convertido pelo dAlgram



Fonte: Autoria própria/dAlgram.

3.5.5 Conversor

A árvore da biblioteca jsPlumb com seus nós e conexões é convertida para o formato textual a la Fitch usando um algoritmo recursivo implementado em TypeScript. O algoritmo percorre a árvore começando pelo nó raiz, numera sequencialmente cada nó, e determina algoritmica-

mente se o nó é uma premissa, conclusão, aplicação de regra ou fecho. A cada nó encontrado, uma linha de saída textual na forma a la Fitch, com exceção dos fechos, que geram 2 linhas, e das bifurcações, que geram as linhas de abre e fecha chaves. A geração do texto na forma a la Fitch é feita de forma intuitiva, sem utilizar explicitamente a gramática formal utilizada pelo ANITA, e a conversão é feito em uma só passagem, sem passos intermediários ou múltiplas passagens. Em caso de erro do usuário, o conversor pode cometer erros de interpretação, caso em que o usuário tem que ler a saída e entrada do ANITA para corrigir o erro. Como a testagem do sistema é manual e não realizando uma prova formal do sistema e da conversão, é possível que existam casos de tableaux corretos que não sejam interpretados corretamente pelo GANITA.

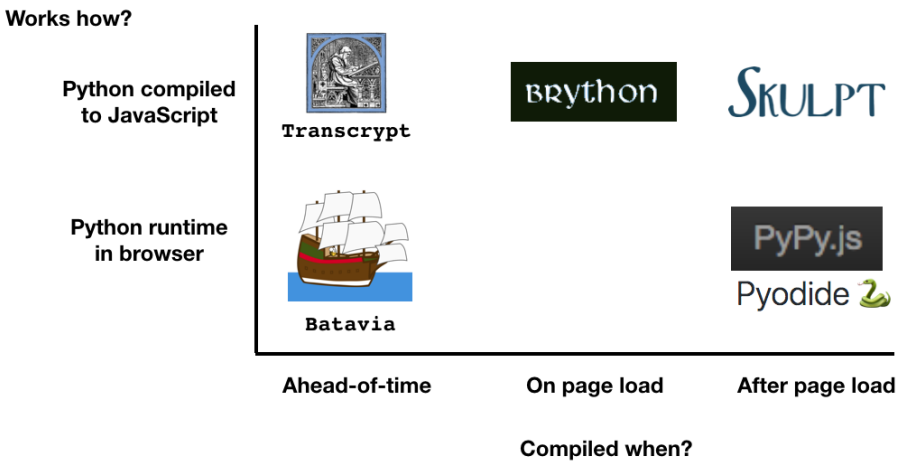
3.5.6 Pyodide, ANITA, WebAssembly e WebWorkers

O GANITA foi desenvolvido como uma extensão do ANITA e usa este para a análise das árvores. Consequentemente, os detalhes de como o ANITA foi utilizado foram centrais ao desenvolvimento do projeto. Além do ANITA em si, foram empregadas outras tecnologias associadamente ao ANITA.

Além do JavaScript, navegadores Web modernos oferecem suporte à programação usando WebAssembly (Wasm), um formato de instruções binário baseado em pilha que normalmente é executado no navegador Web, mas também pode ser executado em outras plataformas. Até o momento o Wasm não possui acesso direto ao DOM, sendo necessário uso de código JavaScript para integrar o Wasm ao resto da Web (WebAssembly, 2024) (Contribuidores MDN, 2025e). Múltiplas linguagens podem serem compiladas para Wasm, usando um compilador como o Emscripten (Contribuidores Emscripten, 2015), que se integra com o compilador LLVM.

A implementação de referência da linguagem de programação Python é o CPython (Python Software Foundation, 2025). O Pyodide é um *port* do CPython para WebAssembly/Emscripten, que permite executar pacotes Python dentro do navegador Web. O Pyodide também tem suporte para gerenciamento de erros, async/await, e mais importantemente uma interface de função estrangeira para comunicação com o JavaScript (Contribuidores Pyodide, 2024a). Essa interface entre o Python e o JavaScript dentro do Pyodide é altamente desenvolvida, permitindo acesso direto a objetos JavaScript do Python e também de objetos Python do JavaScript, além de fazer traduções automáticas ou customizadas de tipo (Contribuidores Pyodide, 2024b). A figura 28 permite comparar o Pyodide com outros sistemas para uso do Python na Web, com o Pyodide incluindo uma *runtime* Python no navegador e executando o código Python após o carregamento da página Web.

Gráfico 28 – Comparativo de sistemas para Python na Web



Fonte: Shaun, 2019.

O ANITA como desenvolvido originalmente tem duas partes. A primeira parte é uma página Web dinâmica operando no modelo cliente-servidor, que interage com a segunda parte, que é um pacote Python rodando no servidor que faz a análise sintáxica e semântica dos tableaux analíticos e analisa se estão corretos ou não, e também complementa com o *feedback* ao usuário. A página Web do ANITA não é utilizada no GANITA, com o GANITA tendo sua própria interface Web. O ANITA também pode ser utilizado localmente como uma biblioteca, que é como o GANITA o utiliza. O ANITA também oferece suporte ao uso com IPython Widgets, funcionalidade que não foi utilizada no GANITA.

Esse pacote Python tem uma interface primordialmente textual, recebendo a árvore na notação a la Fitch, utilizando o pacote RPLY (Gaynor, 2025) para processar a entrada de texto e retornando a análise também textualmente. O pacote RPLY permite processar a gramatica formal da linguagem da forma a la Fitch, fazendo tanto análise sintáxica como léxica, sendo uma porta para RPython (PyPy Project, 2024) do pacote PLY (Beazley, 2020), que é essencialmente uma reimplementação em Python dos tradicionais utilitários geradores de análise lexical Lex e de análise sintáxica Yacc (Hubert, 2001). A análise lexical é o primeiro passo, e consiste em percorrer o texto de entrada e juntar sequencias de caracteres em conjuntos com algum significado chamados de *tokens*. Esses tokens são utilizados como entrada para a análise sintáxica, que cria uma estrutura de forma de árvore que representa o resultado da análise (Louden, 1997).

No GANITA, o pacote Python do ANITA no formato *wheel* é transferido para o navegador Web do usuário, e roda dentro do ambiente Pyodide, e se comunica com o JavaScript da página principal. O carregamento do Pyodide, incluindo o carregamento das bibliotecas e do próprio interpretador CPython, é lento, demorando entre 5 e 30 segundos. Por isso foi utilizada um WebWorker, tecnologia empregada no navegador que funciona semelhantemente ao *multithreading* (Contribuidores MDN, 2025c). O uso de um WebWorker permite que o Pyodide carregue no plano de fundo enquanto que a *thread* principal que controla a interface da aplicação conti-

nua a funcionar, permitindo ao aluno desenhar a árvore e ao ANITA carregar ao mesmo tempo. O WebWorker e a *thread* principal se comunicam via troca de mensagens assíncronas.

3.5.7 Construção e implantação

Utilizamos o webpack como o sistema de construção do projeto, tanto localmente para desenvolvimento quanto para implantação final. O webpack é um empacotador de módulos para aplicativos JavaScript que pode juntar vários recursos, analisar o grafo de dependências entre eles, pré-processar arquivos e empacotá-los em um pequeno número de arquivos processados prontos para serem hospedados em um servidor (OpenJS Foundation; Contribuidores webpack, 2024), fazendo o papel de um sistema de construção para a plataforma Web e melhorando o desempenho do sistema. O webpack é modular, e tem suporte para o Pyodide, TypeScript, CSS e a inclusão de arquivos arbitrários, todas funcionalidades utilizadas no GANITA.

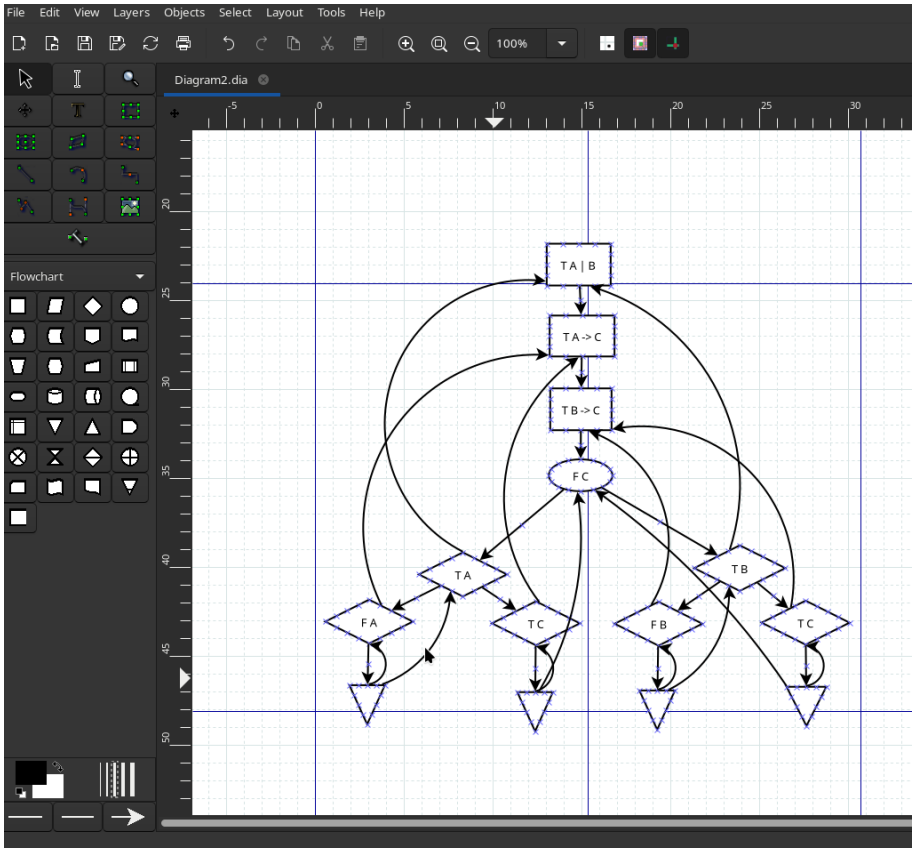
Para facilitar o desenvolvimento e acelerar o download, quase todos os arquivos necessários ao uso do sistema foram agregados como parte do repositório, sendo baixados do mesmo servidor do qual a página HTML principal é obtida. Também foi necessário realizar o reempacotamento do ANITA no formato *wheel*.

O webpack tem um servidor Web integrado para desenvolvimento. Para implantação final do sistema, também utilizamos o GitHub. Usamos o GitHub Actions (GitHub, Inc., 2025a) para executar o webpack no servidor GitHub, preparar o projeto e fazer implantação deste no serviço de publicação de páginas Web GitHub Pages (GitHub, Inc., 2025b). Como o GANITA é do ponto de vista do servidor um site estático, é possível do ponto de vista técnico utilizar o GitHub Pages para implantação do sistema. E como um projeto não comercial, o GANITA também se encaixa nos casos de uso autorizados do GitHub Pages. Com um limite não rígido de 100 GB por mês de transferência de dados, e usando uma sobre-estimativa de 20 MB de download por acesso a página, são 5000 acessos por mês antes de ser necessário o uso de outro serviço de hospedagem (GitHub, Inc., 2025c).

3.6 Iterações e processo de desenvolvimento

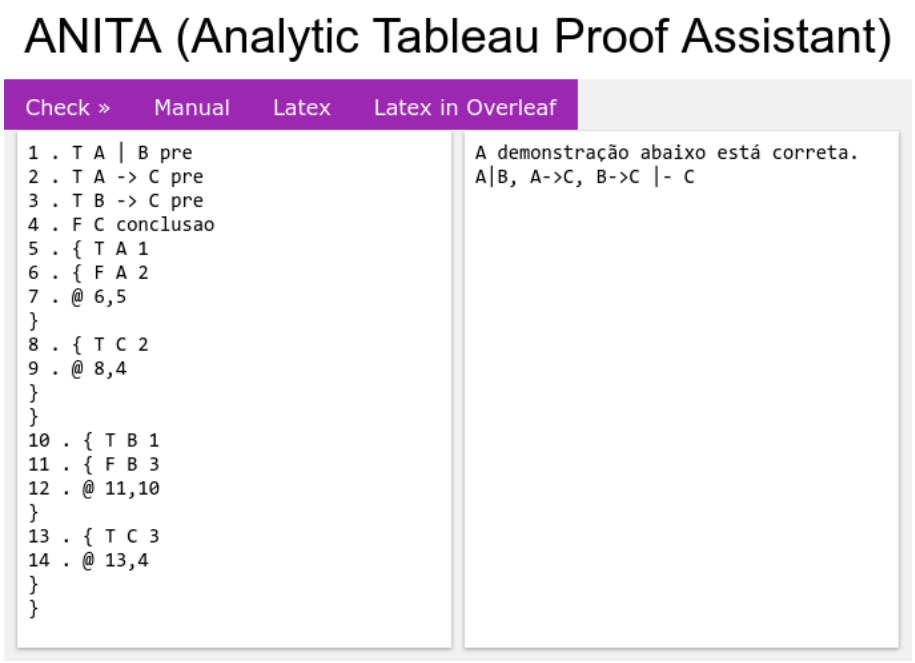
Desenvolvemos o primeiro protótipo do sistema para avaliar a viabilidade do projeto e se este atenderia ao desejado pelo professor orientador que sugeriu o projeto. Desenvolvemos este protótipo como um conversor em Python do formato do software de desenho de diagramas Dia (Contribuidores Dia, 2023) para a a forma a la Fitch. O desenho de um caso exemplo foi feito no Dia, como na figura 29, o conversor foi executado com o arquivo Dia como entrada e sua saída foi inserida na interface Web do ANITA (ver figura 30). Este protótipo e teste foi desenvolvido e realizado em algumas horas.

Gráfico 29 – Desenho da árvore no software Dia para o primeiro protótipo



Fonte: Autoria própria (2025).

Gráfico 30 – Exemplo de uso do primeiro protótipo

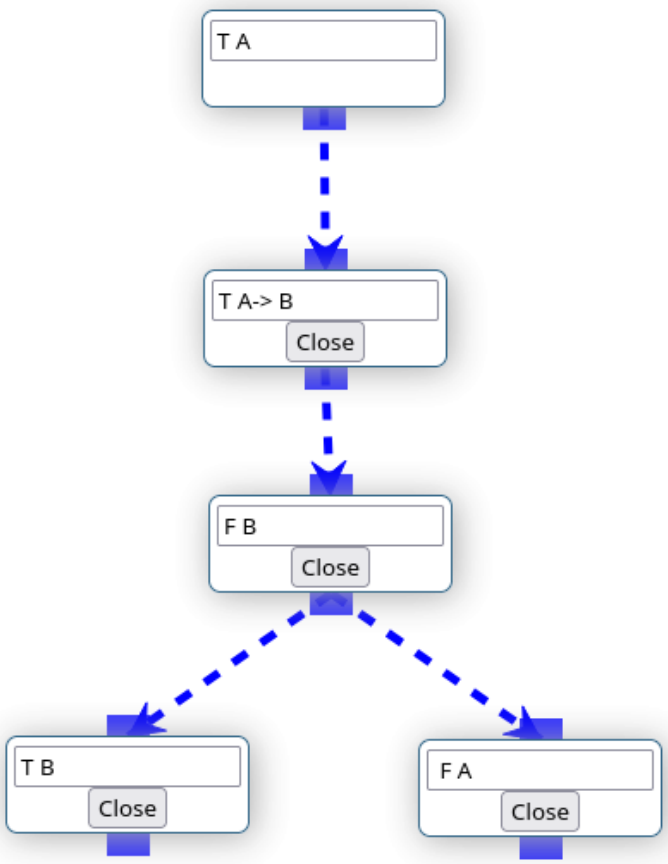


Fonte: Autoria própria (2025).

Posteriormente a este protótipo, passamos a tratar da questão de como seria feito o desenho da árvore no sistema final. Após selecionarmos inicialmente a biblioteca jsPlumb, cria-

mos um novo protótipo do sistema de desenho da árvore para avaliação do professor orientador para avaliar se o sistema para desenhar o tableau estava de acordo com o desejado, com resultados positivos. Este protótipo pode ser observado em funcionamento na figura 31. Nessa etapa também avaliamos e em alguns casos testamos as bibliotecas e ferramentas para desenho da árvore discutidas na seção 3.5.4. A avaliação das bibliotecas e a criação do protótipo para demonstração demorou algumas semanas.

Gráfico 31 – Protótipo inicial do sistema utilizando jsPlumb



Fonte: Autoria própria (2025).

Após esses 2 protótipos relacionados a conversão e ao desenho, iniciamos o desenvolvimento principal do projeto, que demorou aproximadamente 3 semanas. Ao final dessa etapa, realizamos outra reunião de acompanhamento com o professor orientador com a primeira versão plenamente funcional do GANITA para obter algumas pequenas alterações dos requisitos como a adição da funcionalidade de exportar para o Overleaf e outras sugestões de melhorias e mudanças. Após obter alguns *feedbacks* de outros professores discutidos na seção 4.5, também realizamos algumas pequenas melhorias, mudanças e a correções de bugs. Essas correções finais demoraram algumas semanas, sendo realizadas paralelamente à escrita deste relatório.

4 RESULTADOS

Nessa seção discutiremos os principais resultados do sistema, como características de desempenho, usabilidade e avaliação por professores.

4.1 Custos e hospedagem

Não houve custos financeiros no projeto. Todas as bibliotecas e softwares utilizadas são software livres e gratuitos. Como discutido na seção 3.5.7, devido ao GANITA ser, do ponto de vista do servidor hóspede, uma página Web estática acompanhada de recursos também estáticos, pode-se utilizar qualquer hospedagem de sites estáticos gratuita ou de baixíssimo custo para implementação efetiva do sistema, além do GitHub Pages onde ele está presencialmente hospedado. O sistema pode ser trivialmente movido para outro repositório GitHub via *forking*, ou pode-se exportar os artefatos de construção do projeto para um outro servidor.

4.2 Código e licença

O código foi hospedado em um repositório GitHub[1], o mesmo onde está hospedado o sistema em execução[2]. Licenciamos o código feito por nós e outros arquivos originais (ver o apêndice B) sob a licença GNU General Public License versão 2.0 ou posterior, de acordo com a resolução conjunta COPPG/COGEP Nº 01/2021, de 10 de novembro de 2021.

O sistema não coleta nenhum dado dos usuários e não tem restrições de uso, além dos da plataforma GitHub Pages onde ele é hospedado.

4.3 Desempenho

Nesta seção discutiremos algumas características de desempenho do sistema.

4.3.1 Tamanho do download

O tamanho dos arquivos baixados ao usar o sistema estão na tabela 4. O tamanho total descompactado destes arquivos é 14 MB, e o tamanho da transferência compactada é 6 MB, demorando no total aproximadamente 0.5s em uma conexão doméstica. Utilizando a funcionalidade de limitação de conexão do navegador Web Firefox configurada para "DSL"(2 Mbps de download, 1 Mbps de upload, 5ms de latência), os tempos de carregamento variam ao redor de 30 a 60s.

[1] https://github.com/rardiol/ganita
[2] https://rardiol.github.io/ganita/

Tabela 4 – Lista de arquivos transferidos

Arquivo	Tipo	Tamanho da transferência	Tamanho do arquivo
pyodide.asm.wasm	wasm	3,19 MB	10,11 MB
python_stdlib.zip	zip	2,36 MB	2,36 MB
pyodide.asm.js	js	215,18 kB	1,06 MB
main.bundle.js	js	79,17 kB	305,73 kB
pyodide-lock.json	json	26,91 kB	112,21 kB
regras_gammadelta.png	png	55,30 kB	54,67 kB
anita-0.1.13-py3-none-any.whl	wheel	45,02 kB	45,93 kB
regras_alfabeta.png	png	35,49 kB	34,86 kB
805.bundle.js	js	8,86 kB	24,22 kB
arply-0.7.8-py2.py3-none-any.whl	wheel	15,81 kB	16,04 kB
appdirs-1.4.4-py2.py3-none-any.whl	wheel	9,82 kB	9,57 kB
/ganita/	HTML	2,57 kB	7,09 kB
ganita.css	CSS	2,28 kB	4,56 kB
favicon.ico	ícone	801 B	318 B
data:image/png;base64iVB	png	0 B	168 B

Fonte: Autoria própria (2025).

Como o site é integralmente estático, em caso de acessos repetidos do mesmo dispositivo, o tamanho efetivo do download é menor ou quase nulo devido ao uso de *caching*. Pode-se constatar que o maior componente do download é o Pyodide, que necessita baixar uma cópia completa do CPython antes de executar. O Pyodide também causa downloads "sequenciais", que só iniciam depois de outro arquivo ser baixado, o que em alguns casos piora o tempo até o inicio da execução.

4.3.2 Velocidade de execução

A página Web sem considerar os arquivos JavaScript é pequena e simples, portanto fica pronta rapidamente para o uso após o download da página. O atraso mais significativo é a inicialização do Pyodide. Utilizando um computador doméstico, esse prazo, sem considerar o tempo de download dos arquivos, é de aproximadamente 5 segundos. Porém, devido ao uso de um WebWorker, a interface de desenho da árvore já está disponível durante este período, tornando esse tempo de pouca importância, já que é improvável que o usuário esteja pronto para checar a árvore em poucos segundos.

Testando com um computador doméstico, a velocidade de checagem do tableau, mesmo que esse seja razoavelmente complexo, foi quase instantânea.

4.4 Compatibilidade

Utilizamos algumas funcionalidades HTML, CSS e JavaScript que requerem navegadores Web modernos e que podem eventualmente afetar seu uso por usuários utilizando sistemas muito antigos. A funcionalidade utilizada mais moderna é o seletor CSS *:has*, que é considerada *baseline* (linha base), ou seja amplamente suportada pelos navegadores Web mais utilizados,

desde 2023 (Fyrd, 2025a). Tecnologias Web normalmente são implementadas tentando buscar degradação apenas parcial em caso de problemas, e o uso deste seletor não é essencial, portanto as funcionalidades essenciais ainda devem serem preservadas mesmo com um navegador muito antigo. A funcionalidade crítica ao funcionamento do sistema que mais recentemente passou a ser considerada *baseline*, em 2020, é o sistema de módulos e importação de módulos JavaScript (Fyrd, 2025b).

Testamos rapidamente o sistema em um tablet e concluímos que é menos prático, mas possível de ser utilizado. Nossos testes em celular indicam que também é tecnicamente possível de utilizar, mas o pequeno tamanho de tela torna o uso bastante incômodo. O uso em dispositivos móveis não era um objetivo do projeto, mas pode ser de interesse para trabalhos futuros.

4.5 Avaliação por professores

Além da orientação do professor orientador, solicitamos a avaliação e comentários sobre o sistema para outros professores da área de lógica. Aqui apresentaremos e discutiremos estes comentários. Alguns levaram a modificações do sistema já implementadas, alguns podem ser consideradas para trabalhos futuros.

Um professor doutor da área de computação com interesse em algoritmos e complexidade e que atualmente leciona disciplina de introdução à lógica para computação afirmou sobre o projeto que é "uma ferramenta útil e fácil de usar e pode ajudar os alunos para treinar e os professores para usar em sala de aula". Uma melhoria requisitada por ela foi a adição de algum tipo de marcação colorida controlada pelo usuário ou em alguns casos automaticamente identificando o estado do nó, especificamente se este já foi utilizado plenamente, parcialmente ou não em todos os ramos. Outro destaque colorido pedido foi para o caso dos átomos, para facilitar encontrar os átomos contraditórios para formar fechos. Ela também indagou e pode posteriormente observar que o sistema permitia obter os contra-exemplos de sequentes inválidos, e obter informações sobre se um certo ramo ainda ficou insaturado, informações que são fornecidas pela saída textual do ANITA. Durante o teste com a professora, também pudemos constatar que a forma como o sistema permite formar bifurcações não era intuitiva, e que os pontos finais eram pequenos quando controlados com dispositivos de toque em vez de mouse.

Um professor doutor da grande área da filosofia com interesse em lógica e computação comentou "Que maravilha, Adolfo [orientador do projeto]! Sei que o pacote qtree não está preparado para isso, mas seria fantástico se fosse possível gerar código LaTeX para a versão completinha dos tableaux, incluindo as setinhas para justificativas e fechos. O que você acha?". Está funcionalidade poderia de fato em teoria ser implementada, utilizando a biblioteca *tree-dvips*[3], e modificações no ANITA para emitir a saída LaTeX modificada.

Um professor doutor da grande área da filosofia com interesse na área de lógica-matemática comentou "Que excelente ideia. Qual seria a possibilidade de implementar uma

[3] https://ctan.org/pkg/tree-dvips

aba com todas as regras do sistema para o aluno, por exemplo, ter como exemplo as aplicações das regras? Acho essa uma excelente ferramenta didática. Parabéns.". Esse comentário levou a uma expansão da seção de ajuda da interface, onde incluímos as tabelas com as regras de expansão da árvore aplicáveis.

Outro professor doutor da área de computação, com área de interesse em inteligência artificial e atualmente lecionando disciplina de introdução à lógica para computação comentou "Acho que falta documentação para que o usuário saiba representar todas as etapas da prova. Talvez um exemplo inicial que pudesse ser carregado já ajudasse neste primeiro uso. Eu vasculhei os documentos e não encontrei. Para ser sincero, não consegui construir uma prova."O que demonstra uma fraqueza do sistema, ser insuficientemente intuitivo. Como a intenção do sistema é o uso como parte auxiliar do ensino, esperamos que uma instrução inicial por parte do professor em sala seja suficiente para permitir ao aluno utilizar o sistema. A funcionalidade do sistema se carregar já com um exemplo também é possível de implementar, necessitando alterações no código para permitir algum tipo de desserialização (Standard C++ Foundation, 2015) de dados para o formato da árvore.

Um monitor de disciplina de introdução à lógica também testou o sistema e encontrou um bug relacionado ao fecho de nós que não eram nós folhas, que corrigimos.

A tabela 5 apresenta um resumo dos prós e contras observados do sistema.

Tabela 5 – Prós e contras observados do sistema

Prós	Contras
Não requer instalação	Necessidade de baixar novamente a cada uso
Compatível com quase qualquer dispositivo com navegador Web	Inicialização demorada em alguns casos
Funciona offline após carregamento inicial	Difícil de utilizar em celular
Simples e barato de hospedar	Consome mais recursos do cliente comparado com o ANITA
Livre e gratuito	Não permite compartilhar árvores
Informações adicionais disponíveis textualmente	Falta de integração da saída do ANITA com a árvore
Útil e fácil de usar	Pouco intuitivo
Excelente ferramenta didática	Requer instrução inicial para uso
Usável por alunos e professores	

Fonte: Autoria própria (2025).

5 CONCLUSÃO E TRABALHOS FUTUROS

5.1 Trabalhos futuros

O sistema como já construído parece ser de uso, mas como foi discutido na seção de avaliação (4.5) e ao longo do texto, apresenta algumas fraquezas que poderiam serem corrigidas, que discutiremos aqui.

5.1.1 Interface e usabilidade

Algumas partes do sistema poderiam ser melhoradas ou expandidas para melhorar vários aspectos do sistema, como usabilidade, flexibilidade, compatibilidade, eficiência, entre outros.

Primeiramente, a interface poderia ser melhorada para aplicar os métodos do *Responsive Web Design (RWD)* (Contribuidores MDN, 2025a). Esse é um conjunto de métodos de desenvolvimento usando as tecnologias usuais da Web (HTML, CSS, JavaScript) para que a página e os seus elementos se movam e aumentem e diminuam de tamanho flexivelmente conforme o tamanho do visor, entre outras considerações para permitir o uso do mesmo sistema em uma multitude de plataformas. Isso permitiria o sistema se tornar mais facilmente usável em dispositivos móveis, o que pode ser de interesse, já que apesar dos laboratórios universitários terem acesso a desktops, muitos alunos tem acesso mais fácil a dispositivos móveis. Outra tecnologia que poderia ser empregada são as *Progressive Web Apps (PWAs)*, que permitem o uso de sites Web de uma forma mais semelhante ao de aplicativos nativos, o que permitiria a instalação local do GANITA em dispositivos e o seu uso plenamente offline e permitindo maior controle da interface.

Outra possibilidade é implementar funções de serialização e desserialização (Standard C++ Foundation, 2015) da árvore, o que permitiria compartilhar árvores online e abrir e compartilhar demonstrações total ou parcialmente completas automaticamente para fins didáticos.

5.1.2 ANITA e Python

Outro aspecto do sistema que poderia ser reconsiderado é o uso do Pyodide e do ANITA. O Pyodide (ver seção 3.5.6) como uma porta do CPython para WebAssembly, tem que carregar e rodar um interpretador Python completo no navegador, o que tem um custo de transferência do arquivo do interpretador, o tempo de carregamento até que o interpretador esteja pronto para uso, e uma maior dificuldade de integração do código Python e por extensão do ANITA com o GANITA. Uma possibilidade que poderia minimizar alguns desses problemas é o uso de outra tecnologia para Python na Web. Uma possibilidade é o Brython (Quentel; et al, 2025), que permite o uso do Python como linguagem principal na Web. Outra é o py2wasm (Akbary, 2024),

que compila o código Python para WebAssembly sem a necessidade de interpretação em tempo de execução. Uma possibilidade mais minimalista é o uso do PyScript (PyScript Development Team, 2025), que além do suporte ao Pyodide, também tem suporte para uso do MicroPython, uma variante do Python com capacidades reduzidas mas consumo de recursos menor.

Outra possibilidade é a conversão do ANITA para JavaScript, ou de forma manual, ou usando algum tradutor como o Transcrypt (Hooge, 2024), e substituindo a biblioteca RPLY por um equivalente JavaScript. Também é possível utilizar, modificar ou criar alguma outra biblioteca de lógica para cumprir funções semelhantes ao que o ANITA desempenha atualmente.

5.1.3 Gramática formal para conversão

Como discutido na seção 3.5.5, a conversão da forma em árvore para a forma textual é feita de maneira intuitiva. Uma solução alternativa para esse processo é realizar uma conversão mais bem estruturada, utilizando a gramática formal utilizada pelo ANITA, que utiliza o gerador de análise lexical e sintáxica RPLY (Gaynor, 2025) para analisar a forma a la Fitch. Esse processo então faria uma conversão em múltiplas etapas, e poderia fazer diretamente a conexão entre cada aspecto da forma a la Fitch gerada e da árvore desenhada, de maneira semelhante à um transpilador (Tomassetti, 2020). Isso também possivelmente permitiria provar que a árvore desenhada pode corretamente representar todos os tableaux possíveis.

5.1.4 *Feedback* para o aluno

Uma outra melhoria possível é uma melhor integração do *feedback* do ANITA com a árvore desenhada no GANITA, por exemplo destacando onde está o erro encontrado e potencias maneiras de corrigir. Essas melhorias poderiam seguir de uma melhor integração do ANITA com o JavaScript, e o uso de uma interface entre os dois mais integrada do que a comunicação textual empregada atualmente.

5.1.5 Internacionalização e localização

Atualmente, o sistema está disponível apenas em português. Seria possivelmente de interesse traduzir o GANITA para outras línguas. Seria necessário realizar a internacionalização e localização da interface Web, utilizando alguma técnica de internacionalização ou de tempo de compilação ou de tempo de execução do lado do cliente para Web (Richard, 2020) e utilizando bibliotecas auxiliares como o *webpack-localize-assets-plugin* (Osame, 2025). O ANITA já tem uma versão em inglês que poderia ser utilizada.

5.2 Conclusão

O objetivo deste trabalho é o desenvolvimento de um sistema didático e gráfico para o desenho e checagem automática de tableaux analíticos. Acreditamos que este objetivo foi cumprido, e que o sistema é usável para ensino e mais intuitivo e próximo do ensino em papel do que o ANITA, não necessitando o ensino especifico da forma a la Fitch. O sistema tem ajuda integrada, apesar de não ser intuitivo o suficiente para uso puramente independente sem instrução inicial.

O uso de tecnologias relativamente inovadoras como aplicativos Web dinâmicos apenas do lado do cliente se demonstrou viável. Apesar das limitações de maior lentidão de carregamento quando comparada com o sistema cliente-servidor tradicional do ANITA, a forma de implementação do GANITA se demonstrou viável e tem suas vantagens, como simplicidade de hospedagem e a não necessidade de se construir duas bases de código e comunicação entre estas. O aplicativo Web é usável em desktops e conexões de rede comuns. Também implementamos tecnologias Web modernas de amplo uso como webpack e TypeScript com sucesso.

Os comentários dos professores demonstraram interesse, e também sugeriram modificações e melhorias de vários níveis de complexidade para implementação. os outros testes demonstraram que o sistema tem características de desempenho compatível com o uso geral, com limitações para o uso em dispositivos móveis.

Nossa conclusão é que esperamos que o sistema seja de utilidade didática para a comunidade acadêmica.

REFERÊNCIAS

AKBARY, S. **Announcing py2wasm: A Python to Wasm compiler · Blog · Wasmer**, . 2024. Disponível em: https://wasmer.io/posts/py2wasm-a-python-to-wasm-compiler.

Amazon Web Services, Inc. ou afliadas. **What is a Web App? - Web Application Explained - AWS**, . 2025. Disponível em: https://aws.amazon.com/what-is/web-application/.

BEAZLEY, D. **PLY (Python Lex-Yacc)**, . 2020. Disponível em: https://dabeaz.com/ply/.

CAHILL, N. **split/packages/splitjs at master · nathancahill/split**, . 2021. Disponível em: https://github.com/nathancahill/split/tree/master/packages/splitjs.

CLARKSON, M. R. *et al.* **The Curry-Howard Correspondence — OCaml Programming: Correct + Efficient + Beautiful**, . 2025. Disponível em: https://cs3110.github.io/textbook/chapters/adv/curry-howard.html.

Cloudflare, Inc. **What do client side and server side mean? | Client side vs. server side**, . 2025. Disponível em: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/.

Contribuidores Dia. **Apps/Dia – GNOME Wiki Archive**, . 2023. Disponível em: https://wiki.gnome.org/Apps/Dia/.

Contribuidores Emscripten. **Main — Emscripten 4.0.11-git (dev) documentation**, . 2015. Disponível em: https://emscripten.org/.

Contribuidores Fabric.js. **Fabric.js Javascript Library**, . 2025. Disponível em: https://fabricjs.com/.

Contribuidores MDN. **Responsive web design - Learn web development | MDN**, . 2025. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design.

Contribuidores MDN. **The web standards model - Learn web development | MDN**, . 2025. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model.

Contribuidores MDN. **Using Web Workers - Web APIs | MDN**, . 2025. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

Contribuidores MDN. **What is a progressive web app? - Progressive web apps | MDN**, . 2025. Disponível em: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app.

Contribuidores MDN. **WebAssembly concepts - WebAssembly | MDN**, . 2025. Disponível em: https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Concepts.

Contribuidores Pyodide. **Pyodide — Version 0.27.7**, . 2024. Disponível em: https://pyodide.org/en/stable/.

Contribuidores Pyodide. **Type translations — Version 0.27.7**, . 2024. Disponível em: https://pyodide.org/en/stable/usage/type-conversions.html#implicit-conversions.

dAIgram. **Free AI diagram generator from images | Export to JPG, PNG, PDF, JSON file**, . 2024. Disponível em: https://www.daigram.app/.

Devographics. **State of JavaScript 2024: Usage**, . 2024. Disponível em: https://2024.stateofjs.com/en-US/usage/.

Diagramly. **Diagramly.Ai**, . 2025. Disponível em: https://diagramly.ai/.

FREEMAN, S. *et al.* Active learning increases student performance in science, engineering, and mathematics. **Proceedings of the National Academy of Sciences of the United States of America**, v. 111, n. 23, p. 8410–8415, jun. 2014. ISSN 0027-8424. Disponível em: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4060654/.

Fyrd. **:has() CSS relational pseudo-class | Can I use... Support tables for HTML5, CSS3, etc**, . 2025. Disponível em: https://caniuse.com/css-has.

Fyrd. **JavaScript operator: import | Can I use... Support tables for HTML5, CSS3, etc**, . 2025. Disponível em: https://caniuse.com/mdn-javascript_operators_import.

GAYNOR, A. **alex/rply**, . 2025. Original-date: 2012-08-06T00:56:54Z. Disponível em: https://github.com/alex/rply.

GitHub, Inc. **Understanding GitHub Actions**, . 2025. Disponível em: https://docs.github.com/en/actions/get-started/understanding-github-actions.

GitHub, Inc. **What is GitHub Pages?**, . 2025. Disponível em: https://docs.github.com/en/pages/getting-started-with-github-pages/what-is-github-pages.

GitHub, Inc. **GitHub Pages limits**, . 2025. Disponível em: https://docs.github.com/en/pages/getting-started-with-github-pages/github-pages-limits.

Google; Contribuidores Bazel. **Why a Build System?**, . 2025. Disponível em: https://bazel.build/basics/build-systems.

HOOGE, J. de. **1. Transcrypt: what and why — Transcrypt 3.9.2 documentation**, . 2024. Disponível em: https://www.transcrypt.org/docs/html/what_why.html#what-is-transcrypt.

HOWSON, C. **Logic with Trees: An Introduction to Symbolic Logic**. [*S.l.*]: Routledge, 2005. Google-Books-ID: Y4WGAgAAQBAJ. ISBN 978-1-134-78550-6.

HUBERT, B. **Lex and YACC primer/HOWTO: Introduction**, . 2001. Disponível em: https://tldp.org/HOWTO/Lex-YACC-HOWTO-1.html.

jsPlumb. **community.jsplumbtoolkit.com/**, . 2024. Disponível em: https://community.jsplumbtoolkit.com/.

Konva. **Konva - JavaScript Canvas 2d Library**, . 2025. Disponível em: https://konvajs.org/.

LEACH-KROUSE, G. Carnap: An Open Framework for Formal Reasoning in the Browser. **Electronic Proceedings in Theoretical Computer Science**, v. 267„ p. 70–88, mar. 2018. ISSN 2075-2180. ArXiv:1803.03092 [cs]. Disponível em: http://arxiv.org/abs/1803.03092.

LOUDEN, K. C. **Compiler construction : principles and practice**. 1st. ed. [*S.l.*]: Cengage Learning, 1997.

MALIK, A. **Subject and Course Guides: Digital UIC: Static and Dynamic Websites**, . 2025. Disponível em: https://researchguides.uic.edu/digital-uic/static-dynamic.

MEYER, N. **Star Trek VI: The Undiscovered Country**, . 1991.

Microsoft. **Documentation for Visual Studio Code**, . 2025. Disponível em: https://code.visualstudio.com/docs.

Microsoft. **Why does TypeScript exist?**, . 2025. Disponível em: https://www.typescriptlang.org/why-create-typescript/.

MINH, F. T.; GONNORD, L.; NARBOUX, J. Proof Assistants for Teaching: a Survey. **Electronic Proceedings in Theoretical Computer Science**, v. 419„ p. 1–27, maio 2025. ISSN 2075-2180. Disponível em: http://arxiv.org/abs/2505.13472v1.

NEW, M. S.; LICATA, D. R.; AHMED, A. Gradual type theory. **Journal of Functional Programming**, v. 31„ p. e21, jan. 2021. ISSN 0956-7968, 1469-7653. Disponível em: https://www.cambridge.org/core/journals/journal-of-functional-programming/article/gradual-type-theory/2D5DC0E87301B1724C42B7E31F90DD6B.

OpenJS Foundation. **Node.js — Introduction to Node.js**, . 2025. Disponível em: https://nodejs.org/pt/learn/getting-started/introduction-to-nodejs.

OpenJS Foundation; Contribuidores ESLint. **Getting Started with ESLint - ESLint - Pluggable JavaScript Linter**, . 2025. Disponível em: https://eslint.org/docs/latest/use/getting-started.

OpenJS Foundation; Contribuidores webpack. **Concepts**, . 2024. Disponível em: https://webpack.js.org/concepts/.

OSAME, H. **privatenumber/webpack-localize-assets-plugin**, . 2025. Original-date: 2021-05-10T08:47:40Z. Disponível em: https://github.com/privatenumber/webpack-localize-assets-plugin.

PELLETIER, F. J.; HAZEN, A. Natural Deduction Systems in Logic. *In*: ZALTA, E. N.; NODELMAN, U. (Ed.). **The Stanford Encyclopedia of Philosophy**. Spring 2024. Metaphysics Research Lab, Stanford University 2024. Disponível em: https://plato.stanford.edu/archives/spr2024/entries/natural-deduction/.

PERHáč, J. *et al.* OnlineProver: Experience with a Visualisation Tool for Teaching Formal Proofs. **Electronic Proceedings in Theoretical Computer Science**, v. 419„ p. 55–74, maio 2025. ISSN 2075-2180. Disponível em: http://arxiv.org/abs/2505.05987v1.

PERUčlć, F. **Treant.js - javascript library for drawing tree diagrams**, . 2021. Disponível em: https://fperucic.github.io/treant-js/.

PyPy Project. **Welcome to RPython's documentation! — RPython Documentation**, . 2024. Disponível em: https://rpython.readthedocs.io/en/latest/.

PyScript Development Team. **Features - PyScript**, . 2025. Disponível em: https://docs.pyscript.net/2025.7.1/user-guide/features/.

Python Software Foundation. **Alternative Python Implementations**, . 2025. Disponível em: https://www.python.org/download/alternatives/.

QUENTEL, P.; et al. **Brython**, . 2025. Disponível em: https://brython.info/.

RICHARD, S. **Internationalization And Localization For Static Sites**, . 2020. Section: General. Disponível em: https://www.smashingmagazine.com/2020/11/internationalization-localization-static-sites/.

SCHWARZ, W. **wo/tpg: Tree Proof Generator**, . 2024. Disponível em: https://github.com/wo/tpg.

SHAPIRO, S.; KISSEL, T. K. Classical Logic. *In*: ZALTA, E. N.; NODELMAN, U. (Ed.). **The Stanford Encyclopedia of Philosophy**. Spring 2024. Metaphysics Research Lab, Stanford University 2024. Disponível em: https://plato.stanford.edu/archives/spr2024/entries/logic-classical/.

Shopify. **Draggable JS – JavaScript drag and drop library**, . 2025. Disponível em: https://shopify.github.io/draggable/.

SILVA, F. S. C. d.; FINGER, M.; MELO, A. C. V. d. Lógica para computação, „,. 2006. Disponível em: https://repositorio.usp.br/item/002933896.

SMULLYAN, R. M. **First-order Logic**. [*S.l.*]: Courier Corporation, 1995. Year: 1995. ISBN 978-0-486-68370-6.

Standard C++ Foundation. **Serialization and Unserialization, C++ FAQ**, . 2015. Disponível em: https://web.archive.org/web/20150405013606/http://isocpp.org/wiki/faq/serialization#serialize-overview.

Taye Adeyemi. **interact.js - JavaScript drag and drop, resizing and multi-touch gestures for modern browsers**, . 2025. Disponível em: https://interactjs.io/.

The LaTeX Project. **Introduction to LaTeX**, . 2025. Disponível em: https://www.latex-project.org/about/.

TOMASSETTI, F. **How to write a transpiler**, . 2020. Disponível em: https://tomassetti.me/how-to-write-a-transpiler/.

VASCONCELOS, D. R. de. ANITA: Analytic Tableau Proof Assistant. **Electronic Proceedings in Theoretical Computer Science**, v. 375„ p. 38–53, mar. 2023. ISSN 2075-2180. ArXiv:2303.05864 [cs]. Disponível em: http://arxiv.org/abs/2303.05864.

WebAssembly, . 2024. Disponível em: https://webassembly.org/.

WENDT, M. **mar10/fancytree: JavaScript tree view / tree grid plugin with support for keyboard, inline editing, filtering, checkboxes, drag'n'drop, and lazy loading**, . 2025. Disponível em: https://github.com/mar10/fancytree/.

APÊNDICE A – Dependências

A.1 package.json (pacotes npm)

```
{
  "devDependencies": {
    "@eslint/js": "^9.29.0",
    "@pyodide/webpack-plugin": "^1.3.3",
    "@types/split.js": "^1.4.0",
    "css-loader": "^7.1.2",
    "eslint": "^9.29.0",
    "style-loader": "^4.0.0",
    "ts-loader": "^9.5.2",
    "typescript": "^5.8.3",
    "typescript-eslint": "^8.35.0",
    "webpack": "^5.99.9",
    "webpack-cli": "^6.0.1",
    "webpack-dev-server": "^5.2.2"
  },
  "dependencies": {
    "@jsplumb/browser-ui": "^6.2.10",
    "@jsplumb/connector-bezier": "^5.13.7",
    "@jsplumb/connector-flowchart": "^5.13.7",
    "pyodide": "^0.27.7",
    "split.js": "^1.6.5"
  }
}
```

A.2 Pacotes Python

- anita-0.1.13-fork
- appdirs-1.4.4
- rply-0.7.8

APÊNDICE B – Arquivos do projeto

Este apêndice contem a listagem de arquivos contidos no repositório do projeto.

B.1 Arquivos de autoria própria

Esta lista contem os arquivos que criamos em sua totalidade ou parcialmente.

- dist/assets/exemplo.mp4
- dist/assets/exemplo.png
- dist/favicon.ico
- dist/ganita.css
- dist/index.html
- eslint.config.mjs
- .github/workflows/webpack.yml
- .gitignore
- src/custom.d.ts
- src/index.ts
- src/mainPy.py
- src/webWorker.ts
- tsconfig.json
- webpack.config.js

B.2 Arquivos gerados automaticamente

Esta é a lista de arquivos gerados automaticamente pela ferramenta npm (ver seção 3.5.3.4).

- package.json
- package-lock.json

B.3 Outros arquivos

Esta é a lista de arquivos de autoria alheia ao projeto. Realizamos o empacotamento em formato *wheel* do ANITA, mas seu conteúdo não foi modificado.

- dist/py/anita-0.1.13-py3-none-any.whl
- dist/py/appdirs-1.4.4-py2.py3-none-any.whl
- dist/py/rply-0.7.8-py2.py3-none-any.whl
- dist/assets/regras_alfabeta.png
- dist/assets/regras_gammadelta.png