

Uma introdução à lógica assistida por computador

Flávio L. C. de Moura
Departamento de Ciência da Computação
Universidade de Brasília¹

28 de agosto de 2023

¹flaviomoura@unb.br

Capítulo 1

Introdução

Este material foi desenvolvido para dar suporte às disciplinas que tenham lógica (proposicional e/ou de primeira ordem) e uma introdução ao estudo de algoritmos como parte do seu programa. Neste contexto, podemos citar diversas disciplinas de graduação da Universidade de Brasília que podem utilizar este material integralmente ou em parte como, por exemplo, Fundamentos Teóricos da Computação, Lógica Computacional e Projeto e Análise de Algoritmos do Departamento de Ciência da Computação, Lógica Matemática do Departamento de Matemática e Lógica do Departamento de Filosofia. Normalmente, o público que cursa estas disciplinas pertence aos cursos de Computação, Matemática e Engenharias em geral, mas acreditamos que este material seja útil para todos que tenham interesse nos temas de lógica e algoritmos.

A primeira parte deste material apresenta os conceitos básicos de lógica partindo da lógica proposicional e chegando na lógica de primeira ordem. A lógica de primeira ordem pode ser vista como a "lógica padrão" utilizada, ainda que informalmente, em Matemática e Computação. Este estudo inicial de lógica será útil para a segunda parte que utiliza os conceitos estudados na resolução de problemas computacionais. Tanto o estudo de lógica quanto o de algoritmos dá especial atenção à construção de provas (matemáticas). Neste contexto, as provas são inicialmente feitas em papel e lápis (provas informais), e posteriormente em computador (provas formais). A construção de provas é um tema que costuma ser espinhoso, de forma que aqui tentaremos facilitar o processo de familiarização com este tema partindo de provas simples, para em seguida explorarmos situações mais complexas.

Linguagens naturais, como o Português por exemplo, são ambíguas por natureza, e para evitarmos possíveis dúvidas na leitura de fórmulas ou propriedades utilizaremos linguagens mais restritas. Iniciaremos com a linguagem da lógica proposicional (LP), que nos permitirá resolver diversos problemas interessantes. Estes problemas serão estudados também no contexto computacional. Com isto, queremos dizer que resolveremos problemas manualmente, isto é, em papel e lápis, e também no computador.

Apesar da LP possuir limitações de expressividade, ela será útil para que possamos entender a dinâmica da construção de provas, mas a lógica efetivamente usada no dia a dia do matemático ou do cientista da computação é a Lógica de Primeira Ordem (LPO), que nos permitirá expressar propriedades de algoritmos de forma mais natural. Durante esta caminhada, estudaremos um assunto fundamental que está presente em diversos contextos: *indução*. Intuitivamente, o conceito de indução é bastante simples, mas a sua aplicação em situações específicas costuma gerar muita dúvida.

A construção de provas mecânicas, ou seja, provas feitas em computador, é uma atividade que tem despertado interesse crescente nas últimas décadas em função da forma como a computação tem se infiltrado no nosso dia a dia. Mas aqui precisamos de uma pequena pausa para explicarmos o que queremos dizer com provas feitas por computador. Esta explicação se faz necessária porque existem pelo menos duas abordagens distintas no que se refere a este assunto: os provadores automáticos de teoremas por um lado, e os assistentes de prova por outro.

Um provador automático de teoremas é um programa munido de uma heurística que recebe um

teorema como argumento e tenta, de forma automática, encontrar uma prova para o teorema dado [30, 19, 22]. Um assistente de provas por outro lado, consiste em um programa que requer a orientação/interação do usuário para poder construir uma prova. Ou seja, o usuário vai guiando o sistema na construção de prova, enquanto o sistema verifica se cada passo dado/sugerido pelo usuário está correto. São exemplos de assistentes de prova o PVS[26], o Isabelle/HOL[24], o Lean[11] e o Coq[33]. Neste material trabalharemos com o assistente de provas Coq, que é um sistema de código aberto e que pode ser instalado em sistemas Linux, MacOS e Windows, ou até mesmo ser executado via browser[1].

Existem materiais muito interessantes que servem como tutoriais do Coq, como por exemplo, [29], ou [6]. Este material não é um tutorial do Coq, mas a sua utilização enriquecerá bastante nosso estudo. Nosso foco é o estudo da lógica proposicional e de primeira ordem, assim como a sua utilização/aplicação no estudo de algoritmos. Para isto utilizaremos o Coq como ferramenta de apoio mostrando como um assistente de provas pode ser útil nesta caminhada. Aqui é importante observar também que um assistente de provas é basicamente uma linguagem de programação juntamente com uma linguagem de especificação, ou seja, além da linguagem de programação existem uma camada lógica adicional, chamada de linguagem de especificação, que nos permite expressar os lemas e teoremas, por exemplo. A camada lógica do Coq é baseada em um formalismo conhecido como *cálculo de construções indutivas* [27] que é muito mais expressivo do que a lógica de primeira ordem que estudaremos aqui. Neste sentido, utilizaremos apenas uma pequena parte do poder de computacional do Coq.

Não assumimos nenhum conhecimento prévio de Coq, e sua utilização é opcional. Ou seja, é perfeitamente possível utilizar apenas a parte teórica deste material. A ideia aqui é que você possa reproduzir os temas abordados em Coq a partir do zero: simplesmente abra o Coq com a interface de sua preferência, e siga as orientações das atividades propostas. Nem tudo que será abordado aqui terá uma versão correspondente em Coq, já que alguns temas serão puramente teóricos e foram pensados para serem feitos apenas em papel e lápis. E mesmo a etapa de construção de provas em um assistente de provas deve ser precedida de um esboço em papel e lápis. As atividades a serem realizadas no assistente de provas têm um arquivo correspondente de apoio cujo *link* é fornecido junto com o texto da atividade.

No contexto de algoritmos e desenvolvimento de *software* é comum a utilização de testes como método de validação. Ou seja, o programa (ou *software*) é executado com diversas entradas distintas, e se nenhum problema é encontrado, o programa é considerado bom o suficiente para ser utilizado. De fato, a primeira coisa que fazemos após implementar um algoritmo é testá-lo para diversas entradas, e caso alguma resposta seja incorreta, uma revisão da implementação é feita para corrigir o erro, e então novos testes são realizados. Este processo é repetido até que o programador sinta confiança na implementação, mas depois de todos estes testes é possível dizer que o programa é correto? Certamente não! Pensando no caso particular da implementação de um algoritmo de ordenação listas de naturais ou inteiros (ou qualquer estrutura munida de uma ordem total), sabemos que existe uma infinidade de listas de inteiros que podem ser utilizadas nos testes, e portanto não é possível testar todas elas. Em se tratando de programas utilizados em sistemas críticos (aviação, medicina, sistemas bancários, etc), por menores que sejam as chances de erros, falhas não são toleradas. O que fazer então para garantir a correção de um programa? Uma abordagem possível consiste em utilizar a lógica para **provar** a correção do programa! Uma prova de uma propriedade de um programa fornece a garantia de que o programa satisfaz a propriedade provada **sempre!** Esta é a abordagem que utilizaremos aqui, e que tem se mostrado cada vez mais importante para o desenvolvimento da Matemática[15, 13, 2, 3] e Computação[20, 28, 25]. Para concluir esta seção e começarmos a colocar a mão na massa, listamos três exemplos famosos de erros em sistemas computacionais:

1. **Therac-25:** Uma máquina de radioterapia controlada por computador causou a morte de pelo menos 6 pacientes entre 1985 e 1987 por overdose de radiação.
2. **Pentium FDIV:** Um erro na construção da unidade de ponto flutuante do processador Pentium da Intel causou um prejuízo de aproximadamente 500 milhões de dólares para a empresa que se viu forçada a substituir os processadores que já estavam no mercado em 1994.

3. **Ariane 5:** Um foguete que custou aproximadamente 7 bilhões de dólares para ser construído explodiu no seu primeiro voo em 1996 devido ao reuso sem verificação apropriada de partes do código do seu predecessor.

Já deixamos claro que vamos **provar** muita coisa aqui. Mas o que é uma prova? Uma resposta possível "é um argumento feito para convencer alguém"[32]. O problema deste argumento é que pessoas diferentes podem ter compreensões distintas sobre o argumento, de forma que o argumento pode ser uma prova para uma pessoa, mas não para a outra... estranho, não? Uma definição geral e abstrata para a noção de prova não é uma tarefa fácil, mas forneceremos uma definição precisa em um contexto mais restrito, a saber, o da lógica simbólica[18, 34].

Capítulo 2

Lógica

O que é lógica? Esta é uma pergunta difícil de responder de forma direta e precisa, mas em um contexto bem amplo, podemos dizer que é um ramo da filosofia que estuda os princípios e métodos de inferência e demonstração, que nos permitem determinar as condições em que determinados fatos podem, ou não, serem concluídos a partir de outros [23]. Ao longo deste material trabalharemos para deixar esta ideia mais clara! Além disso, a lógica tem aplicações em muitas disciplinas além da filosofia, como matemática, ciência da computação, linguística e até mesmo em algumas ciências sociais, onde ela é usada para ajudar a estruturar e analisar argumentos e inferências. Na próxima seção, iniciaremos nosso estudo com a lógica proposicional, que nos fornecerá um ambiente relativamente simples, mas rigoroso, para as construções de nossas primeiras inferências.

2.1 A Lógica Proposicional

A lógica proposicional (LP) é baseada na noção **proposição**, que é uma sentença que pode ser qualificada como verdadeira ou falsa, mas nunca ambos. Por exemplo, são proposições:

- $2+2 = 4$.
- $1+3 < 0$.
- 2 é um número primo.
- João tem 20 anos e Maria tem 22 anos.

Mas nem toda sentença é uma proposição. De fato, a sentença "Feche a porta!", ou ainda a pergunta "Qual é o seu nome?" não podem ser qualificadas como verdadeira ou falsa, e portanto não são proposições. Estudaremos diversas lógicas distintas, e todas elas utilizam *fórmulas* como elementos fundamentais. As fórmulas são expressões ou *strings* (sequência de símbolos) utilizados para representar uma proposição. Por exemplo, utilizaremos letras latinas minúsculas para representar proposições que não podem ser subdivididas em proposições menores como " $2+2=4$ ", mas não "João tem 20 anos e Maria tem 22 anos". Chamamos estas letras latinas minúsculas de *variáveis proposicionais*. Assim, a variável proposicional é o elemento básico utilizado na construção das fórmulas.

Faremos um estudo incremental partindo de lógicas mais simples e pouco expressivas, e aos poucos vamos incrementando o poder de expressividade destas lógicas tendo o contexto computacional como referência. Trabalharemos com a lógica simbólica de forma que o significado de uma variável proposicional (ou a informação que ela codifica) será irrelevante daqui para frente. Outro aspecto importante

é que o foco do nosso estudo é a construção de provas. Para isto utilizaremos um sistema dedutivo conhecido como *dedução natural*. Este sistema foi criado pelo lógico alemão Gerhard Gentzen (1909-1945), e consiste em um sistema lógico composto por um conjunto de regras de inferência que tenta capturar o raciocínio matemático da forma mais *natural* possível. Iniciaremos a construção de provas em uma lógica denominada o *fragmento implicacional da lógica proposicional*.

2.1.1 O fragmento implicacional da lógica proposicional

As fórmulas do fragmento implicacional da lógica proposicional são formadas pelas variáveis proposicionais apresentadas anteriormente e pelo conectivo binário chamado *implicação*. Podemos representar as fórmulas deste fragmento pela seguinte gramática:

$$\varphi ::= p \mid (\varphi \rightarrow \varphi) \tag{2.1}$$

onde p denota um elemento de um conjunto enumerável de variáveis proposicionais, *i.e.* podemos utilizar qualquer número finito de variáveis proposicionais para construir uma fórmula. O construtor $\varphi \rightarrow \varphi$ diz que uma fórmula implicacional é construída a partir de duas fórmulas já construídas anteriormente. Por exemplo, se p e q denotam variáveis proposicionais então podemos concluir que $p \rightarrow q$ é uma fórmula, e neste caso, chamamos p de *antecedente*, e q de *sucedente* da implicação. Utilizando esta nova fórmula, podemos construir uma nova implicação a partir dela, e por exemplo, p , obtendo $(p \rightarrow q) \rightarrow p$ ou $p \rightarrow (p \rightarrow q)$, e assim por diante. Como indicado pela gramática (2.1), utilizaremos letras gregas minúsculas para representar fórmulas do fragmento implicacional da lógica proposicional.

Nosso objetivo agora é raciocinar sobre as estas fórmulas. Mais especificamente, queremos obter (ou derivar) novas informações a partir de informações conhecidas. Tudo isto em um contexto abstrato onde os símbolos proposicionais utilizados podem representar qualquer informação que corresponda a uma proposição. Utilizaremos a notação de *sequentes* para separar as informações (fórmulas) dadas da nova informação (fórmula) que queremos concluir. Chamaremos as fórmulas dadas de *premissas*, e a fórmula a ser derivada de *conclusão*, assim um sequente é formado por duas partes: um conjunto finito de fórmulas (que são as premissas), digamos Γ , e uma fórmula que é a conclusão, digamos φ , que escrevemos como $\Gamma \vdash \varphi$. Assim, se $\varphi_1, \varphi_2, \dots, \varphi_n$ são as premissas de um sequente, e se ψ é a sua conclusão, então escrevemos $\varphi_1, \varphi_2, \dots, \varphi_n \vdash \psi$ para representar o sequente que tem ψ como conclusão, e o conjunto $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$ de premissas. O conjunto $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$, isto é, a primeira componente do sequente $\varphi_1, \varphi_2, \dots, \varphi_n \vdash \psi$ também pode ser chamado de *contexto* ao longo do texto, e normalmente será escrito sem as chaves que usualmente são usadas para representar conjuntos. Este é um abuso de linguagem usado para deixar a notação mais leve. Assim, se Γ denota um conjunto finito de fórmulas, ao invés de $\Gamma \cup \{\varphi\} \vdash \psi$, escreveremos simplesmente $\Gamma, \varphi \vdash \psi$, onde Γ, φ deve então ser lido como a união Γ com o conjunto unitário $\{\varphi\}$.

O conceito de prova agora será definido de forma mais precisa. Concretamente, uma prova (ou uma derivação) de um sequente da forma $\Gamma \vdash \psi$ é uma sequência de passos dedutivos, e um passo dedutivo consiste na aplicação de uma *regra de inferência* que possui a seguinte forma:

$$\frac{\Gamma_1 \vdash \gamma_1 \quad \Gamma_2 \vdash \gamma_2 \quad \dots \quad \Gamma_k \vdash \gamma_k}{\Gamma \vdash \psi}$$

onde $k \geq 0$. Quando $k = 0$ e $\psi \in \Gamma$ a regra corresponde a um *axioma*:

$$\frac{}{\Gamma \vdash \psi} \text{ (Ax), se } \psi \in \Gamma$$

Uma prova (*i.e.* uma sequência de passos dedutivos) pode ser representada por meio de uma estrutura de árvore, onde os nós são anotados com seqüentes. A raiz da árvore é anotada com o seqüente que queremos provar, digamos, $\Gamma \vdash \psi$, e as são axiomas. Quais são as regras de inferência que podem ser utilizadas no fragmento implicacional da lógica proposicional? Além do axioma apresentado acima, temos duas regras para a implicação. Antes de apresentá-las devemos lembrar que o sistema dedutivo que utilizaremos se chama *dedução natural*, e as regras deste sistema são divididas em dois tipos: *introdução* e *eliminação*. A regra de eliminação da implicação é conhecida pelo nome *modus ponens* e tem a seguinte estrutura:

$$\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} (\rightarrow_e)$$

ou seja, para construirmos uma prova de um seqüente com a forma $\Gamma \vdash \psi$ utilizando esta regra, precisamos construir duas outras provas: uma do seqüente $\Gamma \vdash \varphi \rightarrow \psi$, e outra do seqüente $\Gamma \vdash \varphi$. Ou seja, na leitura desta regra de baixo para cima (*i.e.* da conclusão para as premissas) reduzimos o problema de provar $\Gamma \vdash \psi$ a dois outros problemas (potencialmente) mais simples. Esta regra também pode ser lida de cima para baixo (*i.e.* das premissas para a conclusão), e neste caso precisamos de uma prova de uma implicação, a saber $\Gamma \vdash \varphi \rightarrow \psi$, e de uma prova do antecedente desta implicação, a saber $\Gamma \vdash \varphi$ para construirmos uma prova da conclusão da implicação, ou seja, uma prova de $\Gamma \vdash \psi$.

A regra de introdução é bastante intuitiva e, em certo sentido, nos fornece uma definição da implicação:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} (\rightarrow_i)$$

ou seja, na leitura de baixo para cima, para construirmos a prova de uma implicação precisamos construir uma prova do sucedente assumindo que temos uma prova do antecedente. Na leitura de cima para baixo, precisamos transformar uma prova do antecedente em uma prova do sucedente.

O interesse computacional do fragmento implicacional está diretamente relacionado ao algoritmo de inferência de tipos em linguagens funcionais[17]. O fundamento teórico destas linguagens é o cálculo λ [5] desenvolvido por Alonzo Church em 1936 [7, 8]. Para mais detalhes veja o Capítulo 1 de [4]. Vejamos um exemplo.

Exemplo 1. Considere o sequente $\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow p \rightarrow r$. A primeira observação a ser feita aqui é que a implicação é associativa à direita, ou seja, $\varphi \rightarrow \psi \rightarrow \gamma$ deve ser lido como $\varphi \rightarrow (\psi \rightarrow \gamma)$, e não como $(\varphi \rightarrow \psi) \rightarrow \gamma$. Portanto, o sequente que queremos provar deve ser lido como $\vdash (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$. Utilizando inicialmente a regra (\rightarrow_i) , temos a seguinte situação:

$$\frac{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)}{\vdash (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))} (\rightarrow_i)$$

Agora podemos aplicar novamente a regra (\rightarrow_i) :

$$\frac{\frac{p \rightarrow q, q \rightarrow r \vdash p \rightarrow r}{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)$$

E mais uma vez, já que a conclusão do sequente é ainda uma implicação:

$$\frac{\frac{\frac{p \rightarrow q, q \rightarrow r, p \vdash r}{p \rightarrow q, q \rightarrow r \vdash p \rightarrow r} (\rightarrow_i)}{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)$$

Agora não é mais possível utilizar a regra (\rightarrow_i) porque a conclusão r não é uma implicação, mas podemos utilizar a hipótese $q \rightarrow r$ para obter r , desde que tenhamos uma prova de q para utilizarmos (\rightarrow_e) . Neste ponto, a árvore é bifurcada em dois ramos e precisamos dividir o contexto de forma adequada em cada um dos ramos.

$$\frac{\frac{\frac{p \rightarrow q, q \rightarrow r, p \vdash q}{p \rightarrow q, q \rightarrow r, p \vdash p \rightarrow q} (\text{Ax})}{p \rightarrow q, q \rightarrow r, p \vdash r} (\rightarrow_e)}{\frac{\frac{p \rightarrow q, q \rightarrow r \vdash p \rightarrow r}{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)}$$

Observe que o ramo da direita consiste em um axioma já que a fórmula $q \rightarrow r$ pertence ao conjunto de hipóteses. No ramo da esquerda podemos obter q por meio da regra (\rightarrow_e) com as hipóteses $p \rightarrow q$ e p . A prova completa é dada a seguir:

$$\frac{\frac{\frac{\frac{p \rightarrow q, q \rightarrow r, p \vdash p}{p \rightarrow q, q \rightarrow r, p \vdash q} (\text{Ax})}{p \rightarrow q, q \rightarrow r, p \vdash p \rightarrow q} (\rightarrow_e)}{\frac{p \rightarrow q, q \rightarrow r, p \vdash r}{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)} (\rightarrow_i)$$

Este primeiro exemplo possui uma característica importante: a escolha da regra a ser aplicada em cada passo é única, e portanto não temos outra opção para a construção desta prova! Se todo sequente tivesse esta característica, poderíamos construir um programa de computador que fizesse este trabalho para nós. Mesmo programas que não foram desenvolvidos especificamente para a construção destas provas podem resolver facilmente estas situações particulares. Por exemplo, solicitei ao ChatGPT¹ a construção de uma prova para o sequente acima e obtive a seguinte resposta:

ChatGPT (modelo gpt-4)

1. $p \rightarrow q$ (suposição)
2. $q \rightarrow r$ (suposição)
3. p (suposição)
4. q ($\rightarrow E, 1, 3$)
5. r ($\rightarrow E, 2, 4$)
6. $p \rightarrow r$ ($\rightarrow I, 3 - 5$)
7. $(q \rightarrow r) \rightarrow (p \rightarrow r)$ ($\rightarrow I, 2 - 6$)
8. $(p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)$ ($\rightarrow I, 1 - 7$)

Observe que, como nossa prova, a resposta do ChatGPT possui 3 aplicações da regra (\rightarrow_i) e 2 de (\rightarrow_e), o que já dá uma boa indicação de que as duas provas são iguais. De fato, as linhas 1, 2 e 3 são suposições, isto é, fórmulas que são assumidas, e considerando as folhas da nossa árvore, estas 3 fórmulas fazem parte do contexto. A linha 4 constrói uma prova da fórmula q a partir de uma aplicação da regra (\rightarrow_e) utilizando as linhas 1 e 3, ou seja, utilizando as fórmulas p e $(p \rightarrow q)$. Isto corresponde à aplicação de (\rightarrow_e) mais à esquerda na nossa árvore. Na linha 5, uma prova de r é feita utilizando (\rightarrow_e) a partir das linhas 2 e 4, ou seja, utilizando as fórmulas q e $q \rightarrow r$. Isto corresponde à aplicação de (\rightarrow_e) mais à direita na nossa árvore. A linha 6 constrói uma prova da fórmula $p \rightarrow r$ com uma aplicação da regra (\rightarrow_i) a partir da subprova que vai da linha 3 até a linha 5. Esta subprova assume p (linha 3), e conclui r (linha 5), e isto corresponde à primeira aplicação de (\rightarrow_i) (de cima para baixo) na nossa árvore. Na linha 7, uma prova de $((q \rightarrow r) \rightarrow (p \rightarrow r))$ é feita com uma aplicação da regra (\rightarrow_i) a partir da subprova que vai da linha 2 até a linha 6. Esta subprova assume $(q \rightarrow r)$ (linha 2), e conclui $(p \rightarrow r)$ (linha 6), e isto corresponde à segunda aplicação de (\rightarrow_i) na nossa árvore. Por fim, a linha 8 corresponde à terceira aplicação de (\rightarrow_i) na nossa árvore, e portanto a prova dada pelo ChatGPT é a mesma prova apresentada no exemplo escrita de outra forma. Será que o ChatGPT ou uma inteligência artificial análoga é capaz de, neste momento, resolver todas as questões que serão propostas neste curso? Responderemos esta questão ao longo deste material, mas note que uma resposta afirmativa pode ser vista como uma possibilidade de substituição (parcial ou total) por máquinas, ainda que não de forma imediata, dos profissionais responsáveis por estas tarefas. Vejamos um outro exemplo.

¹Foi utilizado o modelo GPT-4 em 04/06/2023.

Exemplo 2. Considere o sequente $\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$. Como no exemplo anterior, iniciaremos com uma aplicação da regra (\rightarrow_i) :

$$\frac{p \rightarrow q \vdash p \rightarrow p \rightarrow q}{\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)} (\rightarrow_i)$$

Podemos aplicar mais uma vez a regra (\rightarrow_i) :

$$\frac{\frac{\frac{}{p \rightarrow q, p \vdash p \rightarrow q} (\text{Ax})}{p \rightarrow q, p \vdash p \rightarrow q} (\rightarrow_i)}{p \rightarrow q \vdash p \rightarrow p \rightarrow q} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)} (\rightarrow_i)$$

E a prova está completa porque $(p \rightarrow q)$ é uma das fórmulas do contexto. No entanto, poderíamos ter continuado com outra aplicação de (\rightarrow_i) :

$$\frac{\frac{\frac{p \rightarrow q, p \vdash q}{p \rightarrow q, p \vdash p \rightarrow q} (\rightarrow_i)}{p \rightarrow q \vdash p \rightarrow p \rightarrow q} (\rightarrow_i)}{\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)} (\rightarrow_i)$$

Como o contexto é um conjunto de fórmulas, a nova introdução de p não adiciona nenhuma fórmula no contexto porque os conjuntos $\{p \rightarrow q, p\}$ e $\{p \rightarrow q, p, p\}$ são idênticos. E agora, podemos concluir esta prova com uma aplicação de (\rightarrow_e) :

$$\frac{\frac{\frac{\frac{}{p \rightarrow q, p \vdash p} (\text{Ax})}{p \rightarrow q, p \vdash p} (\text{Ax})}{p \rightarrow q, p \vdash q} (\rightarrow_e)}{p \rightarrow q, p \vdash p \rightarrow q} (\rightarrow_i)}{\frac{p \rightarrow q \vdash p \rightarrow p \rightarrow q}{\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)} (\rightarrow_i)} (\rightarrow_i)$$

No exemplo anterior, construímos duas provas para o mesmo sequente, ainda que bastante similares. Será que o ChatGPT consegue construir alguma destas provas? Vejamos:

ChatGPT (modelo gpt-4)

1. $p \rightarrow q$ (suposição)
2. p (suposição)
3. q ($\rightarrow E, 1, 2$)
4. $p \rightarrow q$ ($\rightarrow I, 2 - 3$)
5. $p \rightarrow p \rightarrow q$ ($\rightarrow I, 2 - 4$)
6. $(p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ ($\rightarrow I, 1 - 5$)

Sim! Ele construiu uma derivação correspondente à nossa segunda prova. Vejamos mais um exemplo antes de finalizarmos esta seção.

Exemplo 3. *Sejam φ e ψ fórmulas quaisquer da LP. Considere o seqüente $\psi \vdash \varphi \rightarrow \psi$. Queremos provar que a partir de uma prova de ψ podemos provar qualquer implicação que tenha ψ como conclusão:*

$$\frac{\frac{}{\psi, \varphi \vdash \psi} \text{ (Ax)}}{\psi \vdash \varphi \rightarrow \psi} \text{ (}\rightarrow_i\text{)}$$

O seqüente do exemplo anterior nos diz que podemos construir a prova de uma implicação a partir de uma prova do conseqüente desta implicação. Esta prova é utilizada com muita frequência em outras provas, e por esta razão promovemos este seqüente ao *status* de regra derivada:

$$\frac{\psi}{\varphi \rightarrow \psi} \text{ (}\rightarrow_i\text{)} \emptyset$$

Podemos agora refazer a prova do exemplo anterior utilizando esta regra derivada:

$$\frac{\frac{\frac{}{p \rightarrow q \vdash p \rightarrow q} \text{ (Ax)}}{p \rightarrow q \vdash p \rightarrow p \rightarrow q} \text{ (}\rightarrow_i\text{)} \emptyset}{\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)} \text{ (}\rightarrow_i\text{)}$$

Observe que esta prova corresponde à primeira prova do exemplo anterior: de fato, basta substituir a aplicação da regra derivada $(\rightarrow_i) \emptyset$ pela prova correspondente. Agora vejamos se o ChatGPT consegue fazer o mesmo:

ChatGPT (modelo gpt-4)

Pergunta: Prove o seqüente $\psi \vdash \varphi \rightarrow \psi$.

Resposta: Para provar o seqüente $\psi \vdash \varphi \rightarrow \psi$, podemos usar as regras de inferência do fragmento implicacional da lógica proposicional. Aqui está a prova, em um formato de derivação:

1. ψ (suposição)
2. φ (suposição)
3. ψ (repetição, 1)
4. $\varphi \rightarrow \psi$ ($\rightarrow I$, 2 – 3)

Nesta prova, utilizamos a regra de introdução da implicação ($\rightarrow I$) para fazer a suposição φ (linha 2), e repetimos a suposição original ψ (linha 3). Finalmente, usamos a regra de introdução da implicação ($\rightarrow I$) para inferir a proposição $\varphi \rightarrow \psi$ (linha 4).

Ok, a prova construída é análoga à que fizemos anteriormente, exceto pela linha 3 que repete a fór-

mula assumida na linha 1. Esta diferença ocorre porque o ChatGPT não está escrevendo as provas com a mesma notação de seqüentes, mas a relação entre elas é mais ou menos direta. Retomando a conversa com o ChatGPT:

ChatGPT (modelo gpt-4)

Pergunta: A prova anterior nos diz que podemos provar uma implicação qualquer desde que tenhamos uma prova do seu conseqüente, correto?

Resposta: Sim, a prova anterior ilustra que, dado um conseqüente ψ , podemos sempre provar a implicação $\varphi \rightarrow \psi$ para qualquer antecedente φ , no fragmento implicacional da lógica proposicional. Isso ocorre porque a regra de introdução da implicação (\rightarrow I) nos permite assumir qualquer proposição como uma suposição temporária. Portanto, mesmo que não tenhamos nenhuma informação sobre φ (o antecedente), ainda podemos provar a implicação $\varphi \rightarrow \psi$ se já temos uma prova de ψ .

Pergunta: Ótimo, então você pode construir uma nova prova do seqüente $\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ utilizando a ideia dada na resposta anterior?

Resposta: Sim, podemos construir uma prova do seqüente $\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ usando a ideia que discutimos anteriormente. Essencialmente, podemos provar uma implicação de qualquer antecedente, desde que tenhamos uma prova de seu conseqüente. Aqui está a prova, em um formato de derivação:

1. $p \rightarrow q$ (suposição)
2. p (suposição)
3. $p \rightarrow q$ (repetição, 1)
4. $p \rightarrow p \rightarrow q$ (\rightarrow I, 2 – 3)
5. $(p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ (\rightarrow I, 1 – 4)

Nesta prova, usamos a regra de introdução da implicação (\rightarrow I) para fazer as suposições $p \rightarrow q$ (linha 1) e p (linha 2), e repetimos a suposição original $p \rightarrow q$ (linha 3). Depois, usamos a regra de introdução da implicação (\rightarrow I) para inferir a proposição $p \rightarrow p \rightarrow q$ (linha 4). Finalmente, usamos novamente a regra de introdução da implicação (\rightarrow I) para inferir a proposição $(p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ (linha 5).

Opa! Aqui o ChatGPT se confundiu e não utilizou o raciocínio anterior de forma direta, ainda que a prova apresentada esteja correta. Parece que ele não consegue abstrair a ideia de que uma implicação pode ser provada diretamente a partir do seu conseqüente. Ou seja, ele não conseguiu reproduzir a prova que fizemos acima utilizando a regra (\rightarrow_i). Vamos dar mais uma ajuda?

Pergunta: Que tal chamar de $(\rightarrow i2)$ a nova regra que nos permite provar uma implicação a partir de uma prova do seu consequente? Com esta informação, pode tentar novamente construir a prova utilizando a nova regra $(\rightarrow i2)$?

Resposta: Entendi sua sugestão. A nova regra $(\rightarrow i2)$ permite a introdução de uma implicação a partir de uma prova do consequente. Dessa forma, podemos construir uma prova mais direta do sequente $\vdash (p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$. Aqui está a prova, em um formato de derivação:

1. $p \rightarrow q$ (suposição)
2. $p \rightarrow p \rightarrow q$ $(\rightarrow i2, 1)$
3. $(p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ $(\rightarrow I, 1 - 2)$

Nesta prova, usamos a regra de introdução da implicação $(\rightarrow I)$ para fazer a suposição $p \rightarrow q$ (linha 1). Depois, usamos a nova regra $(\rightarrow i2)$ para inferir a proposição $p \rightarrow p \rightarrow q$ (linha 2). Finalmente, usamos novamente a regra de introdução da implicação $(\rightarrow I)$ para inferir a proposição $(p \rightarrow q) \rightarrow (p \rightarrow p \rightarrow q)$ (linha 3).

Excelente! Com este empurrãozinho o ChatGPT conseguiu utilizar a regra derivada. Os testes acima nos mostram que, apesar de ser uma ferramenta com potencial para o ensino, ainda deve ser utilizada com muita cautela. Devemos ter em mente que esta ferramenta responde a qualquer pergunta, e portanto algumas (ou muitas) delas podem conter erros. Durante este curso, utilizaremos o ChatGPT para enfatizar a importância da criatividade humana no processo de construção de uma prova, especialmente de uma prova não trivial. Mesmo em exemplos simples como os anteriores, o ChatGPT não conseguiu abstrair a ideia da nova regra e utilizá-la diretamente em uma prova sem ajuda, mas inicialmente ele disse que tinha utilizado a nova ideia. Agora é a sua vez! Deixando o ChatGPT de lado, e utilizando as regras (Ax) , (\rightarrow_i) e (\rightarrow_e) , resolva os exercícios a seguir:

Exercício 1. Prove o sequente $\vdash (p \rightarrow p \rightarrow q) \rightarrow p \rightarrow q$.

Exercício 2. Prove o sequente $\vdash (q \rightarrow r \rightarrow t) \rightarrow (p \rightarrow q) \rightarrow p \rightarrow r \rightarrow t$.

Exercício 3. Prove o sequente $\vdash (p \rightarrow q \rightarrow r) \rightarrow (p \rightarrow q) \rightarrow p \rightarrow r$.

Exercício 4. Prove o sequente $\vdash (p \rightarrow q \rightarrow r) \rightarrow (q \rightarrow p \rightarrow r)$.

Exercício 5. Prove o sequente $\vdash (p \rightarrow r) \rightarrow p \rightarrow q \rightarrow r$.

Exercício 6. Prove o sequente $\vdash (p \rightarrow q) \rightarrow (p \rightarrow r) \rightarrow (q \rightarrow r \rightarrow t) \rightarrow p \rightarrow t$.

Na próxima seção veremos como resolver os sequentes do fragmento implicacional da lógica proposicional utilizando um programa de computador feito especificamente para a construção de provas! Utilizaremos o assistente de provas Coq (<https://coq.inria.fr/>) que é uma ferramenta de código aberto e que pode ser facilmente instalada em qualquer sistema operacional. A próxima seção é opcional, mas pode ajudar bastante no estudo da lógica.

2.1.2 O fragmento implicacional no Coq

O assistente de provas Coq (https://coq.inria.fr) é um sistema que implementa, entre outras coisas, as lógicas que estudaremos por aqui. Um assistente de provas, como o nome sugere, é uma ferramenta que, sob nossa orientação, nos permitirá mecanizar uma prova. Ou seja, nos permitirá passar uma prova manuscrita para o computador. Mas qual é a vantagem de refazer uma prova no computador? A grande vantagem é ter a garantia de que a prova está efetivamente correta. Provas manuscritas são mais susceptíveis a erros. No entanto, como comentado na Introdução, um assistente de provas não é adequado para encontrar uma nova prova. Ou seja, não deve ser utilizado de forma não planejada para eventualmente conseguir completar uma prova. A utilização eficiente consiste em primeiro ter uma prova manuscrita (ou pelo menos na cabeça) para, em seguida, reconstruí-la no assistente de prova. Provas feitas em assistentes de prova são bastante confiáveis, e amplamente aceitas na comunidade científica. Um caso interessante que levou o matemático americano Thomas Hales ao mundo dos assistentes de prova. Em 1998, Hales submeteu um artigo para publicação com aproximadamente 250 páginas contendo uma prova da Conjectura de Kepler. Esta conjectura foi formulada em 1611 pelo matemático Johannes Kepler e permaneceu sem solução até então, ou seja, por quase 400 anos! Uma comissão de especialistas foi montada para analisar o artigo e concluiu, após quase 2 anos depois, que tinha 99% de certeza de que a prova apresentada estava correta. Conta-se, que o então diretor do periódico onde se pleiteava a publicação, disse a Hales que se ele fornecesse algum ponto adicional de que a prova estava correta, o artigo seria publicado. Mas o que pode fornecer uma evidência maior da prova de um teorema do que o próprio artigo? Sim, uma prova formal! Em 2003, Hales começou a trabalhar no projeto Flyspeck que tinha como objetivo formalizar sua prova utilizando os assistentes de prova Isabelle[24] e HOL/Light², mas o artigo acabou sendo publicado em 2005[16]. O projeto tinha uma previsão inicial de durar 20 anos, mas em 2014, isto é 11 anos depois, o grupo do projeto Flyspeck anunciou que tinha completado a formalização da prova da Conjectura de Kepler[15].

Voltando ao Coq, é importante observarmos que ele é uma ferramenta de pesquisa, e não foi desenvolvida exclusivamente para o ensino. Grandes projetos foram e são desenvolvidos nele como a formalização do Teorema das 4 cores[13], do Teorema de Feit-Thompson (??) e a verificação de um compilador C[20]. Ainda, assim, veremos que ela nos será bastante útil. O Coq pode ser utilizado via uma *interface web* sem a necessidade de instalação³, no entanto a instalação local costuma ser bastante simples (independente do sistema operacional). Assumiremos que você tem o Coq⁴ instalado em sua máquina ou que tem acesso à internet para utilizar a *interface web*.

Utilizaremos algumas notações para facilitar a identificação de diferentes contextos, principalmente no que se refere ao assistente de provas Coq[33]. Os códigos do Coq são escritos em *verbatim*, e uma sessão típica do Coq possui três janelas:

²<https://www.cl.cam.ac.uk/~jrh13/hol-light/>

³Veja <https://coq.vercel.app/>

⁴Utilizamos a versão 8.16 durante a elaboração deste material.



A janela da esquerda é a janela de especificação, ou seja, onde escrevemos as definições, os lemas e as provas. As especificações serão escritas em uma caixa azul:

janela de especificação

A janela do canto superior direito nos mostra o *status* atual da prova; e a do canto inferior direito nos mostra mensagens do sistema. O texto correspondente a estas janelas aparecerá em uma caixa verde:

janela de prova ou mensagens do sistema

Existem diversas razões para a utilização do Coq, mas aqui apresentaremos duas: a primeira é que o Coq implementa o sistema de dedução natural e portanto, é o candidato natural para explorar computacionalmente este sistema dedutivo; segundo, que o Coq possui uma comunidade muito grande e sempre disponível para ajudar com as dúvidas⁵. Apresentaremos as regras do fragmento implicacional da LP e faremos uma analogia entre o sistema de dedução natural que apresentamos aqui e o Coq sempre que possível, mas como veremos, esta analogia não é feita via uma correspondência direta entre as regras em dedução natural e as *regras* do Coq que são chamadas de *táticas*. De fato, as táticas são desenvolvidas para realizarem vários passos de prova, incluindo simplificações, de só uma vez porque isto facilita o processo de construção de provas em sistemas mais complexos.

A regra de introdução da implicação (\rightarrow_i) é simulada por meio da tática `intro`. Por exemplo, para provarmos uma implicação, digamos:

```
=====
phi -> psi
```

⁵Existem diversos canais disponíveis para dúvidas sobre o Coq como, por exemplo, <https://coq.discourse.group/> e <https://proofassistants.stackexchange.com/>.

podemos utilizar a tática `intro` que vai mover o antecedente `phi` da implicação para as hipóteses, reduzindo assim, o problema de provar `phi -> psi` ao problema de provar `psi` assumindo `phi`:

```
H : phi
=====
psi
```

A regra de *eliminação da implicação* (\rightarrow_e), isto é, a regra *modus ponens* corresponde à tática `apply`. Considere uma aplicação da regra (\rightarrow_e) como a seguir:

$$\frac{\frac{\varphi \rightarrow \psi, \varphi \vdash \varphi \rightarrow \psi}{\varphi \rightarrow \psi, \varphi \vdash \varphi} (Ax) \quad \frac{\varphi \rightarrow \psi, \varphi \vdash \varphi}{\varphi \rightarrow \psi, \varphi \vdash \psi} (Ax)}{\varphi \rightarrow \psi, \varphi \vdash \psi} (\rightarrow_e)$$

A situação correspondente em Coq é dada a seguir:

```
H1 : phi -> psi
H2 : phi
=====
psi
```

Podemos aplicar a tática `apply H1` que vai reduzir a prova de `psi` a uma prova de `phi` já que o conseqüente da implicação `phi -> psi` coincide com a fórmula que queremos provar. Em seguida, concluímos a prova com a utilização da tática `assumption` porque `phi` é a hipótese `H2`.

Agora vamos refazer aqui os exemplos da seção anterior. Iniciaremos com o $\vdash (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$. Precisamos declarar as variáveis `p`, `q` e `r` e, em seguida enunciarmos o lema a ser provado:

```
Parameter p q r: Prop.

Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
Proof.
```

A janela de prova correspondente é:

```
1 goal (ID 1)

=====
(p -> q) -> (q -> r) -> p -> r
```

Repetindo os passos da prova construída na seção anterior, devemos aplicar 3 vezes a regra (\rightarrow_i). Na primeira aplicação, a fórmula `p -> q` é colocada no contexto. Em outras palavras, introduzimos a fórmula `p -> q`:

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intro H1.
```

```
1 goal (ID 2)
```

```
H1 : p -> q
```

```
=====
```

```
(q -> r) -> p -> r
```

Em seguida, introduzimos a fórmula $q \rightarrow r$:

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intro H1. intro H2.
```

```
1 goal (ID 3)
```

```
H1 : p -> q
```

```
H2 : q -> r
```

```
=====
```

```
p -> r
```

E então, introduzimos a fórmula p :

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intro H1. intro H2. intro H3.
```

```
1 goal (ID 4)
```

```
H1 : p -> q
```

```
H2 : q -> r
```

```
H3 : p
```

```
=====
```

```
r
```

Alternativamente, podemos compor os 3 passos de introdução de uma única vez usando a variante `intros`:

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intros H1 H2 H3.
```

```
1 goal (ID 4)
```

```
  H1 : p -> q
```

```
  H2 : q -> r
```

```
  H3 : p
```

```
  =====
```

```
  r
```

Agora podemos provar r via a regra (\rightarrow_e) utilizando a hipótese $H2$ que tem r como consequente. Isto corresponde a utilizar a tática `apply H2`, que resulta na obrigação de provar q :

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intros H1 H2 H3. apply H2.
```

```
1 goal (ID 5)
```

```
  H1 : p -> q
```

```
  H2 : q -> r
```

```
  H3 : p
```

```
  =====
```

```
  q
```

Agora utilizamos novamente a regra (\rightarrow_e) com a hipótese $H1$, ou seja, aplicamos a tática `apply H1`, e concluímos com `assumption` já que p corresponde à hipótese $H3$:

```
Parameter p q r: Prop.
```

```
Lemma ex1: (p -> q) -> ((q -> r) -> (p -> r)).
```

```
Proof.
```

```
  intros H1 H2 H3. apply H2. apply H1.
```

```
  assumption.
```

```
Qed.
```

Será que o ChaGPT também consegue fazer provas em Coq? Vejamos:

Pergunta: Construa uma prova para o sequente $\vdash (p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$ no assistente de provas Coq.

Resposta: Claro, eu posso ajudar com isso. Coq é uma linguagem de programação funcional que também serve como assistente de provas. Aqui está como você provaria o sequente em Coq:

```
Theorem example : forall (p q r : Prop), (p -> q) -> ((q -> r) -> (p -> r)).
Proof.
  intros p q r.
  intros Hpq Hqr Hp.
  apply Hqr.
  apply Hpq.
  apply Hp.
Qed.
```

Uau! A prova está correta! A diferença para o que fizemos se restringe à forma como as variáveis p , q e r foram declaradas: nos declaramos como parâmetro, e ele declarou utilizando quantificação universal. A tática `apply Hp` corresponde à nossa utilização de `assumption`. A utilização de ferramentas como o ChatGPT como um assistente para os estudos pode ser interessante, se utilizadas com senso crítico porque da mesma forma que recebemos respostas corretas também recebemos respostas que não fazem o menor sentido, mas como a ferramenta responde com uma linguagem persuasiva, o seu uso acrítico pode ser bastante prejudicial.

Os exercícios propostos na seção anterior também podem ser feitos em Coq. O arquivo com os enunciados encontra-se disponível em **removido**⁶.

2.1.3 A Lógica Proposicional Minimal

Agora vamos ampliar o poder de expressividade do fragmento implicacional da LP adicionando novos construtores gramaticais e novas regras de inferência. Assim, a nova gramática terá, além da implicação, a constante (\perp), a negação (\neg), a conjunção (\wedge) e a disjunção (\vee):

$$\varphi ::= p \mid \perp \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \quad (2.2)$$

A constante \perp é utilizada para representar a negação: $\neg\varphi$ é o mesmo que $\varphi \rightarrow \perp$. Ou seja, temos duas maneiras distintas de escrever a negação, e portanto a gramática acima possui redundâncias. De fato, veremos que existem outras redundâncias na gramática (2.2), mas elas são úteis porque simplificam a escrita das fórmulas.

A gramática (2.2) define as fórmulas da LP, e a partir dela consideraremos 3 sublógicas da LP: a minimal, a intuicionista e a clássica. Nesta seção estudaremos a lógica proposicional minimal (LPM), que assim como no fragmento implicacional visto anteriormente, possui uma regra de introdução e uma regra de eliminação para cada um dos conectivos lógicos. Ou seja, uma regra de introdução e uma de eliminação para cada um dos construtores recursivos da gramática (2.2).

Apesar da gramática apresentada acima não incluir a bi-implicação, este é um conectivo bastante utilizado, e pode ser escrito em função dos outros conectivos: $\varphi \leftrightarrow \psi$ é o mesmo que $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.

⁶Removido para não identificar a autoria, conforme o item 7.2(c) do edital conjunto DEG/EDU 04/2023.