Lógica Computacional 1

Flávio L. C. de Moura
Departamento de Ciência da Computação
Universidade de Brasília[1]

19 de agosto de 2025

[1]flaviomoura@unb.br

Capítulo 1

TODO Introdução

Este material foi desenvolvido para dar suporte à disciplina Lógica Computacional 1 do Departamento de Ciência da Computação da Universidade de Brasília.

A lógica consiste no estudo do raciocínio, ou seja, consiste no estudo dos caminhos que nos permitem concluir quando um determinado fato é verdadeiro. Na antiguidade, a lógica consistia em um ramo da Filosofia. Depois passou a ser também um ramo da Matemática, e mais recentemente, pode ser vista também como um ramo da Computação.

No contexto computacional, o estudo da lógica é particularmente importante porque é o fundamento matemático dos programas de computador. De fato, a lógica é utilizada tanto em projetos de *hardware* quanto de *software*.

A primeira parte deste material apresenta os conceitos básicos da Lógica Proposicional (LP), que será dividida em três etapas:

1. A Lógica Proposicional Minimal (LPM);

2. A Lógica Proposicional Intuicionista (LPI), e;

3. A Lógica Proposicional Clássica (LPC).

O estudo dessas etapas é feito de forma incremental, onde a compreensão de uma etapa nos permite avançar para a próxima. Utilizaremos o sistema de Dedução Natural para construirmos a demonstração dos sequentes, que são os objetos a serem provados, e portanto, verdadeiros.

A segunda parte, consiste no estudo da Lógica de Primeira Ordem (LPO), também conhecida como Lógica de Predicados. A LPO pode ser vista como a "lógica padrão"utilizada, ainda que informalmente, em Matemática e Computação.

Chamaremos os caminhos trilhados até uma conclusão de demonstração. Assim, podemos definir uma demonstração como sendo um objeto que nos permite concluir se uma determinada afirmação é verdadeira. As provas, ou demonstrações, tanto na LP quanto na LPO serão desenvolvidas em dois níveis: inicialmente serão 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. Para as provas formais, isto é, as provas desenvolvidas em computador utilizaremos o assistente de provas Rocq, que é um *software* de código aberto que pode ser facilmente instalado em qualquer sistema operacional.

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 [RV02, KSUV15, McC10]. 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[ORS92], o Isabelle/HOL[NPW02], o Lean[dMU21] e o Coq[Tea21]. 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[APJ17].

Existem materiais muito interessantes que servem como tutoriais do Coq, como por exemplo, [PCG$^+$14], ou [Chl17]. 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* [Pau14] 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[HAB+15, Gon08, ADGR07, AH14] e Computação[Ler09, Pau15, NNdMA10]. 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"[Smu09]. 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[HR04, vD13].

Capítulo 2

A Lógica Proposicional (LP)

Linguagens naturais, como o Português por exemplo, são ambíguas. De fato, considere a seguinte afirmação:

> Eu vi a moça com o binóculo.

Quem estava com o binóculo? Ou ainda,

> O professor da Maria acabou a aula fazendo apontamentos no seu caderno.

Em qual caderno o professor estava fazendo apontamentos, no da Maria ou no do dele mesmo? A primeira coisa que faremos para evitarmos ambiguidades consiste em restringir a linguagem com a qual trabalharemos.

A Lógica Proposicional é baseada na noção de **proposição**, que consiste em uma afirmação (ou 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. As proposições também são chamadas de sentenças lógicas.

Nos exemplos acima podemos observar dois tipos de proposições:

1. **Proposições atômicas**: são sentenças que não podem ser divididas em proposições menores como é o caso de:

   - 2+2 = 4.
   - 1+3 < 0.
   - 2 é um número primo.

2. **Proposições compostas**: são sentenças formadas pela composição de uma ou mais proposições atômicas como é o caso de:

   - João tem 20 anos e Maria tem 22 anos.
   - Se está chovendo então o chão está molhado.

Como dito anteriormente, a lógica é o estudo do raciocínio, mais especificamente, neste curso estamos interessados no raciocínio dedutivo. Ou seja, estamos interessados no processo de inferência onde a conclusão é garantida se as premissas forem verdadeiras. Vejamos dois exemplos:

Premissa 1: Se está chovendo então o chão está molhado.
Premissa 2: Está chovendo.
Conclusão: O chão está molhado.

Premissa 1: Se $x = 4$ então a terra é o centro do universo.
Premissa 2: $x = 4$
Conclusão: a terra é o centro do universo.

Estruturalmente, os dois exemplos são iguais, o que muda é o conteúdo das proposições. Se abstrairmos o conteúdo das proposições, e escrevermos a implicação "Se p então q"na forma $p \to q$, teremos algo da seguinte forma:

Premissa 1: $p \to q$
Premissa 2: p
Conclusão: q

Como veremos, o conteúdo das proposições será irrelevante porque estamos interessados nos passos da inferência. A linguagem da LP nos permite estudar as regras de inferência sem considerar o conteúdo das proposições. As estruturas abstratas construídas na linguagem da LP são chamadas de *fórmulas bem formadas*, ou simplesmente, fórmulas. Utilizaremos letras latinas minúsculas, aqui chamadas de *variáveis proposicionais*, para representar proposições atômicas. Assim, variáveis proposicionais são fórmulas. Adicionalmente, utilizaremos a implicação para construirmos fórmulas mais complexas, ou seja, para representarmos proposições compostas. Assumiremos um conjunto enumerável $\mathbb{P}$ de variáveis proposicionais.

O Fragmento Implicacional da Lógica Proposicional (FILP) contém exatamente as fórmulas que são construídas utilizando apenas variáveis proposicionais e a implicação como conectivo lógico. Podemos representar as fórmulas deste fragmento pela seguinte gramática:

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

onde $p \in \mathbb{P}$, e o construtor $\varphi \to \varphi$ diz que uma fórmula implicacional é construída a partir de duas fórmulas já construídas anteriormente (não necessariamente iguais). Por exemplo, se p e q denotam variáveis proposicionais então podemos concluir que $p \to q$ 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 $(p \to q) \to p$ ou $p \to (p \to q)$, e assim por diante. Como indicado pela gramática (2.1), utilizaremos letras gregas minúsculas para representar as fórmulas da LP.

A implicação lógica possui algumas diferenças em relação à implicação que costumamos utilizar no dia a dia em linguagem natural. Por exemplo, não precisa existir uma relação de causa e efeito entre o antecedente e o consequente de uma implicação lógica. No primeiro exemplo acima, a causalidade existe entre o antecedente (está chovendo) e o consequente (o chão está molhado) da implicação, ou seja, o fato de estar chovendo é causa do chão estar molhado. No segundo exemplo, não existe relação de causa e efeito entre o antecedente ($x = 4$) e o consequente (a terra é o centro do universo). A implicação lógica é definida apenas pela relação de dependência entre os valores de verdade do antecedente e do consequente, sintetizada na seguinte tabela:

φ	ψ	$\varphi \to \psi$
V	V	V
V	F	F
F	V	V
F	F	V

A tabela acima é conhecida como *tabela verdade* da implicação e fornece todas as possíveis relações de valores de verdade entre o antecedente e o consequente da implicação. Ou seja, nos fornece a semântica ou o significado da implicação. Existem outras formas de expressar a implicação $\varphi \to \psi$ que podem facilitar a compreensão destas relações:

1. φ é condição suficiente para ψ, ou seja, se *varphi* for verdadeira (V) então ψ também será verdadeira. Basta que φ seja verdadeira para que ψ também seja.

2. ψ é condição necessária para φ, ou seja, *psi* segue de φ.

Nosso objetivo agora é raciocinar sobre as fórmulas construídas a partir da gramática (2.1). 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, ou conhecidas, da nova informação (fórmula) que queremos derivar ou 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 $\varphi_1, \varphi_2, \ldots, \varphi_n$ são as premissas de um sequente, e se ψ é a sua conclusão, então escrevemos $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ para representar o sequente que tem ψ como conclusão, e o conjunto $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ de premissas. O conjunto $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$, isto é, a primeira componente do sequente $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ 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:

$$\frac{\Gamma_1 \vdash \gamma_1 \; \Gamma_2 \vdash \gamma_2 \ldots \Gamma_k \vdash \gamma_k}{\Gamma \vdash \psi}$$

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

$$\frac{}{\{\psi\} \vdash \psi} \text{ (Ax)}$$

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 sequentes. A raiz da árvore é anotada com o sequente que queremos provar, digamos, $\Gamma \vdash \psi$, e as folhas 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:

$$\frac{\Gamma_1 \vdash \varphi \to \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1 \cup \Gamma_2 \vdash \psi} \; (\to_e)$$

ou seja, para construirmos uma prova de um sequente com a forma $\Gamma \vdash \psi$ utilizando esta regra, precisamos construir duas outras provas: uma do sequente $\Gamma \vdash \varphi \to \psi$, e outra do sequente $\Gamma \vdash \psi$. 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 $\Gamma \vdash \varphi \to \psi$, e de uma prova do antecedente desta implicação, a saber $\Gamma \vdash \psi$ 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:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi} \; (\to_{\mathrm{i}})$$

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[Hin97]. O fundamento teórico destas linguagens é o cálculo λ[Bar84] desenvolvido por Alonzo Church em 1936 [Chu36, Chu40]. Para mais detalhes veja o Capítulo 1 de [AdM17].

Como primeiro exemplo, considere o sequente $\vdash (p \to q) \to (q \to r) \to p \to r$. A primeira observação a ser feita aqui é que a implicação é associativa à direita, ou seja, $\varphi \to \psi \to \gamma$ deve ser lido como $\varphi \to (\psi \to \gamma)$, e não como $(\varphi \to \psi) \to \gamma$. Portanto, o sequente que queremos provar deve ser lido como $\vdash (p \to q) \to ((q \to r) \to (p \to r))$. Vejamos como aplicar as regras $(\to_{\mathrm{i}}), (\to_{\mathrm{e}})$ e (Ax) apresentadas anteriormente para construirmos a árvore de derivação do sequente $\vdash (p \to q) \to (q \to r) \to p \to r$. Nosso objetivo é construir uma árvore com raiz $\vdash (p \to q) \to (q \to r) \to p \to r$, cuja folha sejam axiomas. Assim, iniciaremos a prova de baixo para cima, isto é, da raiz para as folhas da árvore. Como a fórmula a ser provada é uma implicação, a utilização da regra $(\to_{\mathrm{i}})$ parece ser uma boa ideia:

$$\dfrac{\dfrac{?}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{}\quad\Longleftrightarrow\quad\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to_{\mathrm{i}})$$

Aqui temos o passo chave para aplicarmos as regras de Dedução Natural: precisamos instanciar as variáveis Γ, φ e ψ da regra $(\to_{\mathrm{i}})$. Neste caso, Γ é instanciada como o conjunto vazio, φ é instanciada com a fórmula $p \to q$, e ψ é instanciada com a fórmula $(q \to r) \to p \to r$. Esta instanciação nos permite saber o que colocar no antecedente da regra $(\to_{\mathrm{i}})$:

$$\dfrac{\dfrac{\dfrac{?}{p \to q \vdash (q \to r) \to p \to r}}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{}\quad\Longleftrightarrow\quad\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to_{\mathrm{i}})$$

Agora podemos repetir o processo com o sequente $p \to q \vdash (q \to r) \to p \to r$. Como o consequente é uma implicação, vamos novamente aplicar a regra $(\to_{\mathrm{i}})$ instanciando Γ com o conjunto unitário contendo a fórmula $p \to q$, *varphi* com a fórmula $q \to r$ e ψ com a fórmula $p \to r$:

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{}\quad\Longleftrightarrow\quad\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to_{\mathrm{i}})$$

E assim, podemos avançar mais um passo na construção da árvore:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{?}{p \to q, q \to r \vdash p \to r}}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{}\quad\Longleftrightarrow\quad\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to_{\mathrm{i}})$$

Mais uma vez, a fórmula a ser provada é uma implicação, a saber $p \to r$. Então podemos aplicar a regra $(\to_{\mathrm{i}})$ mais uma vez:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{?}{p \to q, q \to r \vdash p \to r}}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{}\quad\Longleftrightarrow\quad\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to_{\mathrm{i}})$$

Instanciando Γ com o conjunto $p \to q, q \to r$, φ com p e ψ com r, podemos adicionar um novo nó à árvore de derivação:

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{p \to q, q \to r, p \vdash r}}{p \to q, q \to r \vdash p \to r}\,(\to_{\mathrm{i}})}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})$$

Agora precisamos construir uma prova de r tendo como premissas as fórmulas $p \to q, q \to r$ e p. Como a fórmula r não é uma implicação, não temos mais como aplicar a regra $(\to_{\mathrm{i}})$. Este sequente também não tem a forma do axioma, e portanto não podemos aplicar a regra (Ax). Assim, vamos tentar aplicar a regra $(\to_e)$ de eliminação da implicação. A instanciação a ser feita é relativamente simples: $\Gamma_1 \cup \Gamma_2$ será instanciado com o conjunto $p \to q, q \to r, p$ e ψ será instanciado com r:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{?}{p \to q, q \to r, p \vdash r}\,(\to_e)}{p \to q, q \to r \vdash p \to r}\,(\to_{\mathrm{i}})}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}}) \iff \dfrac{\Gamma_1 \vdash \varphi \to \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1 \cup \Gamma_2 \vdash \psi}\,(\to_e)$$

As premissas da regra $(\to_e)$ exigem um pouco mais de atenção. Observe que a árvore de prova é bifurcada em dois ramos: um para $\Gamma_1 \vdash \varphi \to \psi$ e outro para $\Gamma_2 \vdash \varphi$. Adicionalmente, a fórmula φ não aparece no consequente da regra. Como fazer então esta instanciação? A única fórmula que temos que nos permite concluir r é $q \to r$, então parece uma boa ideia instanciar φ com q. Adicionalmente, nosso objetivo é chegar em um axioma, e portanto se instanciarmos Γ_1 com $q \to r$ conseguiremos concluir o ramo esquerdo da árvore com um axioma. No ramo direito, Γ_2 será instanciado com $p \to q, p$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q \to r \vdash q \to r}\,(\mathrm{Ax}) \qquad \dfrac{?}{p \to q, p \vdash q}}{p \to q, q \to r, p \vdash r}\,(\to_e)}{p \to q, q \to r \vdash p \to r}\,(\to_{\mathrm{i}})}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}}) \iff \dfrac{\Gamma_1 \vdash \varphi \to \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1 \cup \Gamma_2 \vdash \psi}\,(\to_e)$$

Por fim, a prova do sequente $p \to q, p \vdash q$ consiste em mais uma aplicação da regra $(\to_e)$. Você consegue dizer como são as instanciações neste caso?

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q \to r \vdash q \to r}\,(\mathrm{Ax}) \qquad \dfrac{\dfrac{}{p \to q \vdash p \to q}\,(\mathrm{Ax}) \qquad \dfrac{}{p \vdash p}\,(\mathrm{Ax})}{p \to q, p \vdash q}\,(\to_e)}{p \to q, q \to r, p \vdash r}\,(\to_e)}{p \to q, q \to r \vdash p \to r}\,(\to_{\mathrm{i}})}{p \to q \vdash (q \to r) \to p \to r}\,(\to_{\mathrm{i}})}{\vdash (p \to q) \to (q \to r) \to p \to r}\,(\to_{\mathrm{i}})$$

O exemplo que acabamos de fazer contém as ideias mais importantes que utilizaremos ao longo de todo o curso: como instanciar e aplicar as regras de Dedução Natural. Agora veja se compreendeu o processo resolvendo os exercícios a seguir com as regras $(\to_{\mathrm{i}}), (\to_{\mathrm{e}})$ e (Ax):

Exercício 1. *Prove o sequente* $\vdash (p \to q \to r) \to (q \to p \to r)$.

Exercício 2. *Prove o sequente* $\vdash (p \to q \to r) \to (p \to q) \to p \to r$.

Exercício 3. *Prove o sequente* $\vdash (p \to q) \to (p \to r) \to (q \to r \to t) \to p \to t$.

Exercício 4. *Prove o sequente* $\vdash (q \to r \to t) \to (p \to q) \to p \to r \to t$.

Exercício 5. *Prove o sequente* $\vdash (p \to p \to q) \to p \to q$.

Referências Bibliográficas

[ADGR07] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic*, 9(1):2–es, December 2007.

[AdM17] M. Ayala-Rincón and F. L. C. de Moura. *Applied Logic for Computer Scientists - Computational Deduction and Formal Proofs*. UTCS. Springer, 2017.

[AH14] Jeremy Avigad and John Harrison. Formally verified mathematics. *Communications of the ACM*, 57(4):66–75, April 2014.

[APJ17] Emilio Jesús Gallego Arias, Benoît Pin, and Pierre Jouvelot. jsCoq: Towards Hybrid Theorem Proving Interfaces. *Electronic Proceedings in Theoretical Computer Science*, 239:15–27, January 2017.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Number v. 103 in Studies in Logic and the Foundations of Mathematics. North-Holland ; Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co, Amsterdam ; New York : New York, N.Y, rev. ed edition, 1984.

[Chl17] A. Chlipala. *Certified Programming with Dependent Types*. MIT Press, 2017.

[Chu36] A. Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[Chu40] A. Church. A Formulation of the Simple Theory of Types. *journal of Symbolic Logic*, 5:56–68, 1940.

[dMU21] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, Lecture Notes in Computer Science, pages 625–635, Cham, 2021. Springer International Publishing.

[Gon08] G. Gonthier. A computer-checked proof of the Four Colour Theorem. Technical report, Microsoft Research Cambridge, 2008.

[HAB+15] T. Hales, M. Adams, G. Bauer, D. Tat Dang, J. Harrison, T. Le Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. Tat Nguyen, T. Quang Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. Hoai Thi Ta, T. N. Tran, D. Thi Trieu, J. Urban, K. Khac Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *ArXiv e-prints*, January 2015.

[Hin97] J. Roger Hindley. *Basic Simple Type Theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[HR04] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004.

[KSUV15] Cezary Kaliszyk, Stephan Schulz, Josef Urban, and Jiří Vyskočil. System Description: E.T. 0.1. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195, pages 389–398. Springer International Publishing, Cham, 2015.

[Ler09] Xavier Leroy. Formal Verification of a Realistic Compiler. *Communications of the ACM*, 52(7):107, 2009.

[McC10] W. McCune. Prover9 and mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005/2010.

[NNdMA10] R. B. Nogueira, A. C. A. Nascimento, F. L. C. de Moura, and M. Ayala-Rincón. Formalization of Security Proofs Using PVS in the Dolev-Yao Model. In *Booklet Proc. Computability in Europe - CiE*, 2010.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lncs*. Springer, 2002.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *CADE*, volume 607 of *Lnai*, pages 748–752. sv, 1992.

[Pau14] C. Paulin-Mohring. Introduction to the Calculus of Inductive Constructions. 2014.

[Pau15] Lawrence C. Paulson. A Mechanised Proof of Gödel's Incompleteness Theorems Using Nominal Isabelle. *J Autom Reasoning*, 55(1):1–37, 2015.

[PCG+14] Benjamin C. Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Catvalin Hriatcu, Vilhelm Sjoberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2014.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[Smu09] Raymond Smullyan. *Logical Labyrinths*. AK Peters, 2009.

[Tea21] The Coq Development Team. The Coq Proof Assistant. Zenodo, October 2021.

[vD13] D. van Dalen. *Logic and Structure*. Universitext. Springer London, 2013.