Lógica Computacional e Algoritmos

Uma introdução assistida por computador

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

7 de junho de 2022

[1]flaviomoura@unb.br

Sumário

Capítulo 1

Introdução

Este material está sendo desenvolvido para dar suporte aos alunos de graduação da Universidade de Brasília nas disciplinas de Lógica Computacional 1 e Projeto e Análise de Algoritmos. 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 trata da análise de algoritmos. 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. Novas atividades serão incorporadas sempre que possível, de forma que este material está em constante atualização.

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, 23]. 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[12] 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, e até mesmo ser executado via browser[1].

Existem materiais muito interessantes que servem como tutoriais do Coq, como por exemplo, [29], ou [8]. Aqui não pretendemos apresentar um tutorial do Coq. Nosso foco é o estudo da lógica proposicional e de primeira ordem, assim como a utilização delas 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. 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. 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 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.

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 em sistemas críticos. 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[21, 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.

A *Lógica Computacional* (LC) tem por objetivo utilizar a lógica para raciocinar sobre Computação, ou seja, consiste na utilização da lógica para a resolução de problemas computacionais. Isto pode ser feito considerando a lógica como uma linguagem de programação [32], ou considerando uma mecanização

do raciocínio lógico de forma a permitir a resolução de exercícios no computador, ao invés da resolução usual em papel e lápis [18]. A abordagem que utilizaremos difere das anteriores, mas possui um pouco de cada uma delas como veremos a seguir.

Para explicar a nossa abordagem, suponha que você tenha um grande banco de dados com informações de uma determinada população, e que por alguma razão precise ordenar estas informações por idade em determinado momento; em outro momento, a ordenação que precisa ser feita é por nome ou outro critério qualquer. O que você faz? Uma alternativa é utilizar uma implementação já feita, mas precisamos então perguntar se a implementação utilizada é correta. A alternativa que utilizaremos consiste em construirmos uma implementação e provarmos que ela está correta. Em particular, estudaremos sistemas dedutivos que nos permitirão expressar e provar propriedades de programas[4].

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"[31]. 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[17, 34].

Este material está sendo construído a partir de notas de aulas utilizadas nas minhas turmas de Lógica Computacional 1 e Projeto e Análise de Algoritmos na Universidade de Brasília nos últimos anos, e portanto está em constante atualização.

Agradecimentos Este material foi escrito com o apoio do programa Aprendizagem para o Terceiro Milênio (A3M) coordenado pelo CEAD/UnB, que viabilizou o trabalho do estudante Rafael Monteiro Rodrigues na elaboração de diversas atividades. O estudante Gabriel Silva também fez contribuições importantes na elaboração das atividades, e em particular, na formalização do algoritmo *mergesort*. No entanto, todos os erros eventualmente existentes são de minha inteira responsabilidade. Críticas e/ou sugestões são muito bem-vindas e podem ser enviadas para `flaviomoura@unb.br`.

Parte I

Lógica

Capítulo 2

A Lógica Proposicional

Iniciaremos nosso estudo com a *lógica proposicional* (LP) que é uma lógica baseada na noção de **proposição**. Uma proposição, por sua vez, é uma sentença que pode ser qualificada como verdadeira ou falsa. São exemplos de proposição:

- $2+2 = 4$.
- $1+3 < 0$.
- 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. Algumas proposições podem ser divididas em proposições menores. Por exemplo, a proposição "João tem 20 anos e Maria tem 22 anos"é composta da proposição "João tem 20 anos"e da proposição "Maria tem 22 anos", que por sua vez não podem mais serem divididas porque os pedaços menores não são mais qualificáveis como verdadeiro ou falso. Uma proposição que não pode ser dividida é um elemento básico utilizado na construção de proposições mais complexas, que chamaremos de *fórmula atômica*. Utilizaremos letras latinas minúsculas para representar fórmulas atômicas. Por exemplo, podemos utilizar a letra q para representar a proposição "Maria tem 22 anos", e a letra p para "João tem 20 anos". A proposição do exemplo acima é construída com a utilização do conectivo "E"(conjunção), que será representado pelo símbolo $\wedge$. Com esta simbologia, podemos codificar a proposição "João tem 20 anos e Maria tem 22 anos"pela fórmula $p \wedge q$. Vejamos então a gramática utilizada na construção das fórmulas da LP, que serão representadas por letras gregas minúsculas:

$$\varphi ::= p \mid \bot \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \tag{2.1}$$

A gramática (2.1) define como as fórmulas da LP são construídas. Ela possui 6 construtores:

1. O primeiro denota uma variável proposicional, e caracteriza uma fórmula atômica, i.e. uma fórmula que não pode ser subdividida em fórmulas menores.

2. O segundo construtor é uma constante que denota o absurdo ($\bot$), que também é uma fórmula atômica. O absurdo será utilizado quando tivermos informações contraditórias em nossas provas. Isto ficará mais claro nos exemplos.

3. O terceiro construtor denota a negação.

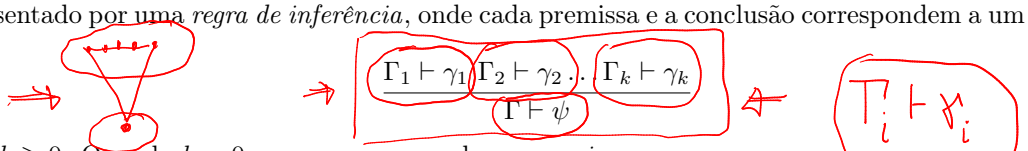
4. O quarto construtor denota a conjunção.

5. O quinto construtor denota a disjunção.

6. O sexto construtor é a implicação.

Uma gramática como (2.1) nos fornece as regras sintáticas para a construção das fórmulas da LP. São quatro construtores recursivos (negação, conjunção, disjunção e implicação) também chamados de conectivos lógicos, e dois não recursivos. 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 $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. Na verdade, a gramática apresentada possui redundâncias, isto é, conectivos que podem ser escritos em função de outros, mas veremos isto posteriormente.

O sistema conhecido como *dedução natural* será utilizado para a construção das provas. 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. Como veremos, estas regras nos permitem derivar novos fatos a partir das premissas. Os fatos a serem provados são representados por meio de fórmulas da LP. Neste contexto, o primeiro conceito importante que aparece é o de *sequente*. Formalmente, um sequente é um par cujo primeiro elemento é um conjunto finito de fórmulas (hipóteses), e o segundo elemento é uma fórmula (conclusão). Assim, se $\varphi_1, \varphi_2, \ldots, \varphi_n$ são as hipóteses de um dado problema, e se ψ é a sua conclusão, escrevemos $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ para representar o sequente que simboliza que ψ tem uma prova a partir das hipóteses $\varphi_1, \varphi_2, \ldots, \varphi_n$. 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 será 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 não deixar a notação sobrecarregada. Assim, ao invés de $\Gamma \cup \{\varphi\} \vdash \psi$, escreveremos simplesmente $\Gamma, \varphi \vdash \psi$, onde Γ, φ deve então ser lido como o conjunto que contém a fórmula φ e todas as fórmulas de Γ. O conceito de prova agora será definido de forma mais precisa. Concretamente, uma prova de um sequente da forma $\Gamma \vdash \psi$ consiste em uma sequência de passos dedutivos, onde cada um destes passos utiliza uma quantidade finita de premissas para avançar para o próximo passo da prova. Cada passo pode ser representado por uma *regra de inferência*, onde cada premissa e a conclusão correspondem a um sequente:



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

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

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

Ou seja, um axioma é uma regra que tem como conclusão um sequente cuja conclusão é uma fórmula que está no contexto.

Uma prova (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, isto é, $\Gamma \vdash \psi$, e as folhas da árvore são axiomas.

Como veremos, a construção desta árvore deve obedecer alguns critérios que detalharemos ao longo deste capítulo, mas em linhas gerais, o principal critério consiste em obedecer as regras que definem o sistema de dedução natural. As regras são divididas em dois tipos: *regras de introdução* e *regras de eliminação* dos conectivos. Iniciaremos com a regra de *eliminação da implicação* que é bastante conhecida. Denotaremos esta regra, que também é conhecida como *modus ponens*, por $(\rightarrow_e)$:

$$\frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to_e)$$

A regra $(\to_e)$ nos diz que a partir de uma prova de $\Gamma \vdash \varphi \to \psi$ e de outra prova de $\Gamma \vdash \varphi$ podemos concluir que $\Gamma \vdash \psi$, ou seja, uma prova de ψ a partir de Γ. As regras de introdução são bastante intuitivas e, em certo sentido, podem ser vistas como uma definição do conectivo que estão introduzindo. A regra de *introdução da implicação*, denotada por $(\to_i)$, possui alguns detalhes importantes. Para construirmos uma prova de uma implicação, digamos do sequente $\Gamma \vdash \varphi \to \psi$, precisamos conseguir construir uma prova de ψ tendo φ como hipótese adicional ao contexto Γ. Em outras palavras, na leitura de baixo para cima, reduzimos o problema de $\Gamma \vdash \varphi \to \psi$ ao novo problema (possivelmente mais simples) de provar o sequente $\Gamma, \varphi \vdash \psi$:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to_i)$$

Também podemos observar esta regra de cima para baixo. Neste caso, ela nos permite passar uma fórmula do conjunto de hipóteses para o consequente como antecedente de uma implicação. Assim, a fórmula φ que era uma das hipóteses necessárias para provar ψ, deixa de ser hipótese, e passa a ser antecedente de uma implicação no consequente. Considerando o subconjunto das fórmulas da lógica proposicional construídas apenas com a implicação (, variáveis e a constante $\bot$) e as regras $(\to_e)$ e $(\to_{\mathrm{i}})$ temos o chamado *fragmento implicacional da lógica proposicional*. O interesse computacional deste fragmento está diretamente relacionado ao algoritmo de inferência de tipos em linguagens funcionais[16]. O fundamento teórico destas linguagens é o cálculo-λ[6] desenvolvido por Alonzo Church em 1936[9, 10]. Para mais detalhes veja o Capítulo 1 de [4]. Como primeiro exemplo, vamos considerar 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 $\vdash (p \to q) \to ((q \to r) \to (p \to r))$. Utilizando inicialmente a regra $(\to_{\mathrm{i}})$, temos a seguinte situação:

$$\frac{p \to q \vdash (q \to r) \to (p \to r)}{\vdash (p \to q) \to ((q \to r) \to (p \to r))}\ (\to_{\mathrm{i}})$$

Agora podemos aplicar novamente a regra $(\to_{\mathrm{i}})$:

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

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

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

Agora não é mais possível utilizar a regra $(\to_{\mathrm{i}})$ porque a conclusão r não é uma implicação, mas podemos utilizar a hipótese $q \to r$ para obter r, desde que tenhamos q para utilizarmos $(\to_e)$. Mas

$$A \to B \to C \equiv A \to (B \to C)$$

$$(A \to B) \to C$$

podemos obter q por meio da regra $(\rightarrow_e)$ com as hipóteses $p \rightarrow q$ e p. A prova completa é dada a seguir:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \rightarrow q, q \rightarrow r, p \vdash p}\,(\text{Ax}) \quad \dfrac{}{p \rightarrow q, q \rightarrow r, p \vdash p \rightarrow q}\,(\text{Ax})}{p \rightarrow q, q \rightarrow r, p \vdash q}\,(\rightarrow_e) \quad \dfrac{}{p \rightarrow q, q \rightarrow r, p \vdash q \rightarrow r}\,(\text{Ax})}{p \rightarrow q, q \rightarrow r, p \vdash r}\,(\rightarrow_e)}{p \rightarrow q, q \rightarrow r \vdash p \rightarrow r}\,(\rightarrow_\text{i})}{p \rightarrow q \vdash (q \rightarrow r) \rightarrow (p \rightarrow r)}\,(\rightarrow_\text{i})}{\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow (p \rightarrow r)}\,(\rightarrow_\text{i})$$

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

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

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

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

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

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

Exercício 7. *Prove o sequente* $\vdash (p \rightarrow q) \rightarrow (p \rightarrow r) \rightarrow (q \rightarrow r \rightarrow t) \rightarrow p \rightarrow t$.

Exercício 8. *Prove o sequente* $\vdash ((((p \rightarrow q) \rightarrow p) \rightarrow p) \rightarrow q) \rightarrow q$.

A *regra de introdução da conjunção*, denotada por $(\wedge_i)$, nos diz o que precisamos fazer para construir uma prova de um sequente que possui uma conjunção na conclusão, isto é, um sequente da forma $\Gamma \vdash \varphi_1 \wedge \varphi_2$, onde Γ é um conjunto finito de fórmulas da LP, e φ_1 e φ_2 são fórmulas da LP. A regra $(\wedge_i)$ é dada pela seguinte regra de inferência:

$$\dfrac{\Gamma \vdash \varphi_1 \qquad \Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \wedge \varphi_2}\,(\wedge_i) \tag{2.2}$$

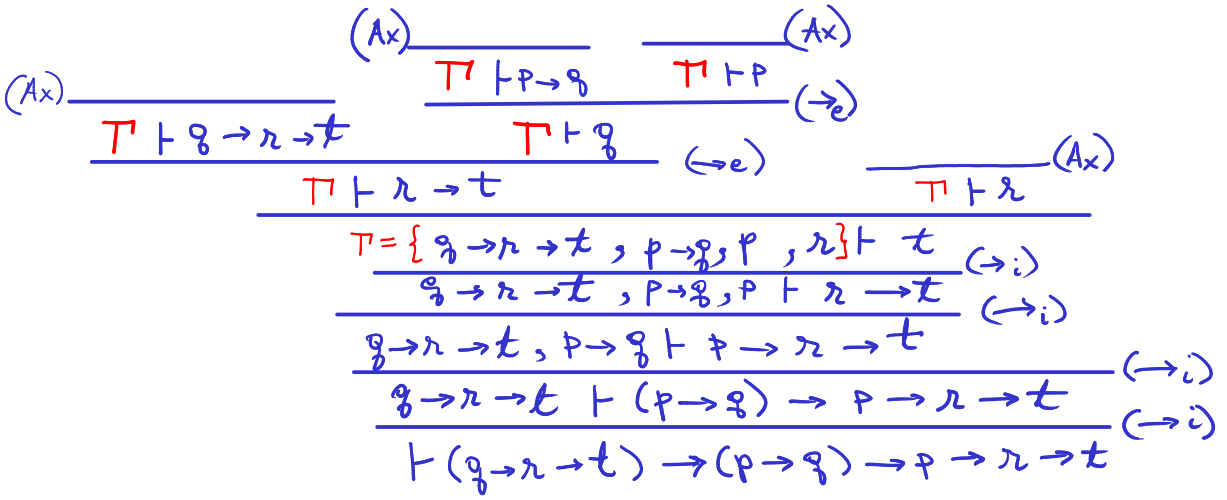
ou seja, uma prova de $\Gamma \vdash \varphi_1 \wedge \varphi_2$ é construída a partir de uma prova de $\Gamma \vdash \varphi_1$ e de uma prova de $\Gamma \vdash \varphi_2$.

Como exemplo de utilização desta regra, construiremos uma prova para o sequente $\varphi, \psi \vdash \varphi \wedge \psi$. Podemos aplicar a regra acima instanciando Γ, φ_1 e φ_2, respectivamente com o conjunto $\{\varphi, \psi\}$, e com as fórmulas φ e ψ. Como resultado temos a árvore abaixo onde os dois ramos gerados por $(\wedge_i)$ são axiomas:

$$\dfrac{(\text{Ax})\ \dfrac{}{\varphi, \psi \vdash \varphi} \qquad \dfrac{}{\varphi, \psi \vdash \psi}\,(\text{Ax})}{\varphi, \psi \vdash \varphi \wedge \psi}\,(\wedge_i)$$

Existem duas regras de eliminação para a conjunção já que podemos extrair qualquer uma das componentes de uma conjunção:

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



Exercício 8. *Prove o sequente* $\vdash ((((p \to q) \to p) \to p) \to q) \to q$.