Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Programa de Pós-Graduação em Matemática
Aplicada e Estatística

Bruno Francisco Xavier

Formalização da Lógica Linear em Coq

Natal/RN

Fevereiro, 2017

Bruno Francisco Xavier

Formalização da Lógica Linear em Coq

Trabalho apresentado ao Programa de Pós-Graduação em Matemática Aplicada e Estatística da Universidade Federal do Rio Grande do Norte, em cumprimento com as exigências legais para obtenção do título de Mestre. Área de Concentração: Modelagem Matemática. Linha de Pesquisa: Matemática Computacional

Universidade Federal do Rio Grande do Norte

Centro de Ciências Exatas e da Terra

Programa de Pós-Graduação em Matemática Aplicada e Estatística

Orientador: Carlos Alberto Olarte Vega

Natal/RN

Fevereiro, 2017

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial Especializada do Centro de Ciências Exatas e da Terra – CCET.

Xavier, Bruno Francisco.
Formalização da lógica linear em Coq / Bruno Francisco Xavier. – Natal, 2017.
63f. : il.

Orientador: Prof. Dr. Carlos Alberto Olarte Vega.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Matemática Aplicada e Estatística.

1. Lógica Matemática - Dissertação. 2. Lógica linear - Dissertação. 3. Coq - Dissertação. 4. Eliminação de corte - Dissertação. I. Vega, Carlos Alberto Olarte. II. Título.

RN/UF/BSE-CCET CDU: 510.6

Bruno Francisco Xavier

Formalização da Lógica Linear em Coq

Trabalho apresentado ao Programa de Pós-Graduação em Matemática Aplicada e Estatística da Universidade Federal do Rio Grande do Norte, em cumprimento com as exigências legais para obtenção do título de Mestre. Área de Concentração: Modelagem Matemática. Linha de Pesquisa: Matemática Computacional

---

Dr. Carlos Alberto Olarte Vega
Presidente - UFRN

Dr. Elaine Gouvêa Pimentel
Interno - UFRN

Dr. Mário Sérgio F. Alvim Junior
Externo à Instituição - UFMG

Natal/RN
Fevereiro, 2017

Agradecimentos

Agradeço a Deus pelo dom da vida e pela Matemática presente em toda sua criação. Desde o início desta jornada, os meus amigos acadêmicos demostraram surpresa, às vezes inveja, do quão atencioso, prestativo, compreensível e paciente é o professor Carlos Olarte em suas orientações, sempre educado e incentivador. Um ser humano fantástico que merece tudo de bom e muito mais. Por isso, eu lhe agradeço.

Não poderia deixar de grafar a ajuda da minha amada esposa Dorinha. Sempre paciente, me orientou de várias maneiras e nos momentos em que vinham os pensamentos de desistência ou do "eu não vou conseguir" suas palavras me revigoravam e me faziam seguir.

Agradeço também aos meus colegas de trabalho que sempre me apoiaram nos estudos, me incentivando, pois nesta dissertação há um pouco de cada um deles. Agradecimentos especiais são direcionados aos colegas do PPGMAE-UFRN que compartilharam o conhecimento e apoiaram uns aos outros nos momentos de dificuldade acadêmica.

Obrigado Laura pelo companheirismo acadêmico, pois sempre esteve de prontidão para ajudar das mais variadas formas.

Sou grato ao amigo Jardson Oliveira que nunca duvidou da minha capacidade, mesmo eu duvidando.

"Apesar dos nossos defeitos, precisamos enxergar que somos pérolas únicas no teatro da vida e entender que não existem pessoas de sucesso ou pessoas fracassadas. O que existe são pessoas que lutam pelos seus sonhos ou desistem deles. (Augusto Cury)

Resumo

Em teoria da prova, o teorema da eliminação do corte (ou *Hauptsatz*, que significa resultado principal) é de suma importância, uma vez que, em geral, implica na consistência e na propriedade subfórmula para um dado sistema. Ele assinala que qualquer prova em cálculo de sequentes que faz uso da regra do corte pode ser substituída por outra que não a utiliza. A prova procede por indução na ordem lexicográfica (peso da fórmula, altura do corte) e gera múltiplos casos quando a fórmula de corte é ou não principal. De forma geral, deve-se considerar a última regra aplicada nas duas premissas imediatamente depois de aplicar a regra do corte, o que gera um número considerável de situações. Por essa razão, a demonstração poderia ser propensa a erros na hipótese de recorremos a uma prova informal. A lógica linear (LL) é uma das lógicas subestruturais mais significativas e a regra do corte é admissível no seu cálculo de sequentes. Ela é um refinamento do modelo clássico e intuicionista. Sendo uma lógica sensível ao uso de recursos, LL tem sido amplamente utilizada na especificação e verificação de sistemas computacionais. À vista disso, se torna relevante sua abordagem neste trabalho. Nesta dissertação, formalizamos, em Coq, três cálculos de sequentes para a lógica linear e provamos que são equivalentes. Além disso, provamos metateoremas tais como admissibilidade da regra do corte, generalização das regras para axioma inicial, ! e *copy* e invertibilidade das regras para os conectivos ⅋, ⊥, & e ?. No tocante à invertibilidade, demonstramos uma versão por indução sobre a altura da derivação e outra com aplicação da regra do corte, o que nos possibilitou conferir que, em um sistema que satisfaz *Hauptsatz*, a regra do corte simplifica bastante as provas em seu cálculo de sequentes. Com a finalidade de atenuar o número dos diversos casos, desenvolvemos várias táticas em Coq que nos permite realizar operações semiautomáticas.

Palavras-chave: logica linear, Coq, eliminação do corte.

Abstract

In proof theory, the cut-elimination theorem (or *Hauptsatz*, which means main result) is of paramount importance since it implies the consistency and the subformula property for the given system. This theorem states that any proof in the sequent calculus that makes use of the cut rule can be replaced by other that does not make use of it. The proof of cut-elimination proceeds by induction on the lexicographical order (formula weight, cut height) and generates multiple cases, considering for instance, when the formula generated by the cut rule is, or is not, principal. In general, one must consider the last rule applied in the two premises immediately after applying the cut rule (seeing the proof bottom-up). This thus generates a considerable amount of cases. For this reason, the proof of cut-elimination includes several cases and it could be error prone if we use an informal proof. Linear Logic (LL) is one of the most significant substructural logics and the cut rule is admissible in its sequent calculus. LL is a refinement of the classical and the intuitionistic model. As a resource sensible logic, LL has been widely used in the specification and verification of computer systems. In view of this, it becomes relevant the study of this logic in this work. In this dissertation we formalize three sequent calculus for linear logic in Coq and prove all of them equivalent. Additionally, we formalize meta-theorems such as admissibility of cut, generalization of initial rule, bang and copy and invertibility of the rules for the connectives par, bot, with and quest. Regarding the invertibility, we demonstrate this theorem in two different ways: a version by induction on the height of the derivation and by using the cut rule. This allows us to show how the cut rule greatly simplifies the proofs in the sequent calculus. In order to mitigate the number of several cases in the proofs, we develop several tactics in Coq that allow us to perform semi-automatic reasoning.

Keywords: linear logic, coq, cut-elimination

Sumário

Introdução

Costa (1980) radica a íntima correlação entre a Lógica e a Matemática como sendo o uso básico que ambas fazem do método axiomático e da formalização na manipulação de seus objetivos. Grandes progressos nessas áreas contribuíram para o ingresso de novas tecnologias no nosso cotidiano que influenciaram substancialmente, dentre outros, o mercado econômico, a comunicação e a saúde a ponto de estarmos totalmente dependentes delas.

Entretanto, no decurso da história, observamos também falsos "progressos". O que se pensava estar correto em uma época fora mostrado incorreto em outra. Hoje temos a possibilidade de verificar formalmente uma demonstração matemática de maneira semiautomática.

A lógica linear difere-se da lógica usual, principalmente, pelo fato desta estar ligada à noção de verdade lógica, i.e. uma declaração verdadeira, e aquela ao de consumo de recursos. Por isso, podemos aplicar a lógica linear em situações de ordem biológica, química, robótica, computacional e muito mais, em que, normalmente, as transições entre estados consomem recursos.

Nesse contexto, o trabalho em pauta empenha-se, sobretudo, em prover uma prova formal da consistência da lógica linear.

Justificativa

> O sistema comportou-se de forma inesperada e por isso não foi possível realizar com sucesso a operação selecionada. [...]. Pedimos desculpas por eventuais transtornos.
>
> *(SIPAC/UFRN)*

A inovação tecnológica é uma fonte inesgotável. Frequentemente, surgem soluções tecnológicas que influenciam nosso modo de (con)viver. É quase impossível, para algumas pessoas, permanecer longe dos aparatos tecnológicos, seja por dependência no trabalho,

nos estudos ou até mesmo na saúde.

Em virtude da complexidade de sistemas de alta-integridade, precisamos atentar para a confiabilidade, porquanto tais sistemas são mais susceptíveis a falhas que podem causar grandes prejuízos financeiros ou até a morte.

> Uma maneira de alcançar esse objetivo é através da utilização de métodos formais, que são técnicas baseadas em formalismos matemáticos e ferramentas para especificar e verificar tais sistemas. *A priori*, o uso de métodos formais não garante a corretude. No entanto, eles podem aumentar significativamente a nossa compreensão de um sistema, revelando incoerências, ambiguidades e incompletude que poderiam passar despercebidos.(CLARKE; WING, 1996, p. 1, tradução nossa)[1]

Os economistas denotam o período atual como a era da informação, ou era digital, a qual se deve muito à conjunção das áreas de conhecimento. Na verdade, não podemos separar totalmente certas áreas do conhecimento, ou seja, se cada área dessas fosse um conjunto, eles não seriam disjuntos. Isso acontece com a filosofia e a matemática, por exemplo. De fato, ambas lançam mão da lógica para alcançar o convencimento. A matemática também possui laços fortíssimos com a computação: o cálculo lambda, equivalente à máquina de Turing, junto com a Teoria de Tipos inspiram as linguagens de programação funcionais.

Desde o seu surgimento, a lógica linear (GIRARD; TAYLOR; LAFONT, 1989) e suas modificações/aprimoramentos têm sido foco de muitos estudos e aplicações, inclusive sistemas computacionais. Este trabalho busca formalizar, em um assistente de provas, vários metateoremas de LL. Isso não só permite dar credibilidade aos resultados na literatura como também, prover ferramentas computacionais para verificar sistemas especificados em LL.

A prova em Coq contribui para uma compreensão muito mais profunda do problema, de suas implicações e das dependências entre os diferentes resultados. Muitos casos que não estão explícitos em uma prova informal surgem quando em uma prova mecanizada.

Motivação e Objetivos

Existe na *internet* uma lista[2], não exaustiva, de cerca de 60 provas matemáticas que foram aceitas como concluídas e publicadas antes de uma lacuna ou um erro ter sido encontrado. Dentre elas, podemos listar:

[1] One way of achieving this goal is by using formal methods, which are mathematically based languages, techniques, and tools for specifying and verifying such systems. Use of formal methods does not *a priori* guarantee correctness. However, they can greatly increase our understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected.

[2] Disponível em: https://en.wikipedia.org/wiki/List_of_incomplete_proofs

1806 André-Marie Ampère afirmou ter provado que uma função contínua é diferenciável em muitos pontos, mas em 1872 Weierstrass deu um exemplo de uma função contínua que não era diferenciável em nenhum ponto: a função de Weierstrass.

1932 A publicação original de Church na tentativa de definir um sistema formal que era inconsistente, que foi corrigido em 1933. A parte consistente de seu sistema transformou-se mais tarde o cálculo lambda.

Além disso, o uso de tecnologia sem garantia do cumprimento integral de seus requisitos ocasionou vários desastres. Dentre as falhas em processos de computação, destacamos:

Anos 90 Bug no módulo de divisão do processador Pentium II[3] da Intel. Prejuízo estimado na ordem de $475.000.000 (quatrocentos e setenta e cinco milhões de dólares)

1996 O foguete Ariane-5[4] se autodestruiu 36 segundos depois do lançamento devido a uma conversão de um número real de 64 bits em um inteiro de 16 bits.

1985–1987 Erro no software de controle da máquina de radiações Therac-25[5], que custou milhões de dólares. Quatro pacientes morreram por excesso de radiação.

Esses problemas nos instigam a questionamentos oportunos: (1) Como evitar que isso aconteça conosco? (2) Como se ter certeza que nossa prova está correta ou pelo menos confiável? (3) Haveria uma maneira eficaz de "provar a nossa prova"? (4) Um programa de computador está correto até que se prove o contrário?

Felizmente, já temos muitos avanços nessa área. Um deles é o assistente de provas: um *software* criado para um computador de uso geral que realmente consegue verificar uma suposta demonstração formal de um teorema ou sistema. Não cabe a este trabalho discutir sobre o grau de confiança de assistente de provas, até porque recairíamos na última pergunta acima. Frisamos que, dada uma prova informal de um teorema, a prova desse teorema em um assistente de provas a torna mais fiável.

Busca-se objetivamente especificar sintaticamente a lógica linear em um assistente de provas, no caso o Coq[6]. Além disso, manipular provas e metateoremas dessa lógica fazem parte dos objetivos a serem alcançados.

[3] Thomas R. Nicely. Original e-mail message announcing the discovery of the Pentium division flaw. Acesso: 23/05/16.

[4] Inquiry Board Traces Ariane 5 Failure to Overflow Error, SIAM News, Vol. 29, No. 8. out/1996. disponível em http://www.math.ufl.edu/ cws/3114/ariane-siam.html. Acesso: 23/05/16.

[5] Ricardo Campos. Acidentes originados por falhas de software. Disponível em: www.di.ubi.pt/ ppra-ta/sdtf/Avarias3.pps. Acesso: 23/05/16.

[6] Versão Coq 8.5pl3, disponível em https://coq.inria.fr/

Metodologia

Este trabalho segue um viés teórico-prático, pois, devido aos objetivos, é necessário transpor para o computador os resultados teóricos. Apesar disso, não atentamos às provas dos teoremas na dissertação, mas somente em Coq, visto que todos eles já estão provados na literatura.

Traçamos uma abordagem conceitual mediante as teorias envolvidas para compreensão deste trabalho e fizemos uma pesquisa acerca dos resultados atuais da lógica linear em Coq. Desenvolvemos um módulo polimórfico *MyMultiset* baseado no módulo *Multiset* da biblioteca CoLoR v1.2[7] e a partir dele formalizamos a lógica linear em três versões: clássica, *one-sided* monádico[8] e *one-sided* diádico[9].

Ao passo que são introduzidos lemas, teoremas e definições importantes procedemos para a formalização em Coq, a fim de que o leitor se sinta confortável com a sintaxe utilizada.

Principais Trabalhos

Existem algumas implementações em Coq da Lógica Linear, sendo a maioria apenas a codificação das regras. Destacamos a formalização da Lógica Linear em Coq de Pierre-Marie Pédrot[10], porém ela segue uma abordagem semântica, enquanto que neste trabalho tratamos do problema pela sintaxe. Destacamos também a implementação[11] em Abella[12], um outro assistente de provas, da Lógica Linear Multiplicativa e Aditiva (MALL) e da Lógica Linear Exponencial Multiplicativa (MELL). No entanto, esses são fragmentos da Lógica Linear.

Organização da Dissertação

Para facilitar a compreensão deste trabalho, o seu conteúdo ficou particionado da seguinte maneira:

Introdução Introduz o tema da dissertação, a lógica linear em Coq, e o que se deseja alcançar, demonstrar metateoremas em LL. Trazemos uma argumentação do motivo de usar um assistente de provas para formalizar a matemática ou sistemas e demonstrar suas propriedades. Em seguida, foi feita uma varredura sobre o tema na

[7] Disponível em: http://color.inria.fr/
[8] Composto por um multiconjunto
[9] Composto por dois multiconjuntos
[10] Encontrada em: https://github.com/ppedrot/ll-coq
[11] Disponível em: https://github.com/meta-logic/abella-reasoning
[12] Homepage: http://abella-prover.org/

literatura, mais especificamente a formalização de LL em Coq.

Fundamentação Teórica Basilar para compreensão do trabalho, este capítulo carrega a revisão da literatura cientifica acerca da lógica, do cálculo lambda, da teoria de tipos, da teoria de prova estrutural e do assistente Coq.

A Lógica Linear em Coq Apresentamos a lógica linear clássica e seu sistema de regras de inferência em três versões: *two-sided*, *one-sided* com um multiconjunto e *one-sided* com dois multiconjuntos. Também mostramos a codificação em Coq. Demonstramos a relação entre esses sistemas. Os teoremas, inerentes aos sistemas da LL, descritos foram provados em Coq. Em especial, apresentamos demonstrações formais para invertibilidade de conectivos e eliminação do corte. Além disso, dissertamos acerca das implicações da eliminação do corte em LL [13].

Aspectos sobre a Codificação A pretensão deste capítulo é trazer ao leitor um pouco do caminho das pedras para se chegar aos resultados almejados.
Desse modo, discutimos teoremas auxiliares e táticas para automatização do trabalho repetitivo.

Considerações Com respeito ao último capítulo, discorremos os resultados do trabalho abrindo caminho para a realização de trabalhos futuros.

[13] Disponível em: https://github.com/brunofx86/LL

Capítulo 1

Fundamentação Teórica

1.1 Um Breve Discurso sobre a Lógica

As origens da lógica formal moderna estão na cultura grega antiga com os silogismos, certas formas de argumentos simples que nunca parecem induzir a erro. Depois, em meados de 1870, Gottlob Frege desenvolveu o que reconhecemos hoje como cálculo de predicados para raciocinar sobre verdades aritméticas. Ele começou usando a teoria dos conjuntos recentemente desenvolvida pelo matemático russo Georg Cantor, hoje conhecida como a teoria ingênua de conjuntos. Ele queria usar a matemática de conjuntos para apoiar a matemática da aritmética e usar a lógica para sustentar a matemática dos conjuntos (BORNAT, 2005).

Todavia, no início dos anos 1900, Bertrand Russell notou algo de errado na teoria de conjuntos de Frege. Ele apresentou um paradoxo em termos de conjuntos que não são membros de si mesmos.

> **Exemplo 1.1.1.** *Há uma vila na qual todos os homens devem ter a barba feita e só há um barbeiro, sendo este homem. O barbeiro faz a barba somente de todos aqueles que não fazem a barba de si mesmos. Quem faz a barba do barbeiro?*
>
> *A vila é um conjunto de pessoas. Dentro desse conjunto, há o conjunto de pessoas que são barbeadas pelo barbeiro, digamos* $\boldsymbol{B}$*. O barbeiro está em* $\boldsymbol{B}$ *ou não? A verdade é que não chegaríamos a nenhuma conclusão, pois o problema é infinitamente recursivo. Portanto, tal vila não pode existir.*

Russell solucionou seu próprio paradoxo através da teoria de tipos criada por ele. A solução consistiu em distinguir conjuntos de ordem sucessivamente superior, tal que os de ordem inferior são elementos daqueles. Assim, o conjunto de todos os conjuntos nunca poderá ser um conjunto incluído num conjunto de conjuntos do mesmo nível e simultaneamente de nível inferior, deixando de ser possível a pergunta acerca de se o conjunto de todos os conjuntos é, ou não, membro de si próprio.

Em 1920, o matemático alemão David Hilbert propôs reformular as bases da matemática de forma rigorosa, partindo da aritmética. Essa proposta ficou conhecida como o Programa de Hilbert. Segundo ele, toda a matemática poderia ser reduzida a um número finito de axiomas consistentes e que esse sistema seria completo, ou seja, qualquer

proposição da matemática poderia ser provada dentro desse sistema.

Na década de 1930, Kurt Gödel mostrou que era realmente impossível estabelecer um conjunto completo de axiomas que possibilitassem deduzir toda a Matemática. Conhecidos como teoremas da incompletude de Gödel, eles afirmam que qualquer sistema axiomático suficientemente expressivo para incluir a aritmética dos números inteiros não pode ser simultaneamente completo e consistente.

Entretanto, o trabalho de Frege, Russell e seus seguidores teve grande utilidade, pois podemos definir uma lógica como um sistema formal: um conjunto de axiomas e regras de inferência. Isso permitiu o surgimento das linguagens de programação funcional sabendo que toda linguagem desse tipo é um sistema formal, uma lógica particular, e todo programa nessa linguagem é um argumento para essa lógica.

O que podemos afirmar, devido o teorema da incompletude de Gödel, é que a lógica usada em Ciência da Computação é necessariamente incompleta. Turing provou que é impossível escrever um algoritmo que possa ser aplicado a qualquer programa de computador, com uma entrada, para decidir se o programa para ou não com esta entrada, i.e. se produz algum resultado.

1.1.1 Lógica clássica *versus* lógica intuicionista

Em lógica, o princípio do terceiro excluído diz que para qualquer proposição p, ou p é verdadeira, ou sua negação é verdadeira: $p \vee \sim p$. Tal princípio pode ser visto, também, em provas por redução ao absurdo, muito usado na matemática. Ele é aceito na lógica clássica (LK), mas rejeitado na intuicionista (LJ) (NEGRI; PLATO; RANTA, 2008).

Atente à seguinte prova clássica usando esse princípio:

Teorema 1.1.2. *Existem números irracionais a e b, tais que a^b é racional.*

Demonstração. Sabe-se que $\sqrt{2}$ é irracional. Considere o número $c = \sqrt{2}^{\sqrt{2}}$. Tal número é racional ou irracional. Se for racional, finalizamos a prova e $a = b = \sqrt{2}$. Mas, se c for irracional, tome $a = \sqrt{2}^{\sqrt{2}}$ e $b = \sqrt{2}$. Logo, $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$, que é claramente racional. □

As provas desse tipo são ditas não-construtivas. Em particular, a prova mostrada não oferece nenhum valor específico para a e b que satisfaça o teorema. Além disso, não sabemos se $\sqrt{2}^{\sqrt{2}}$ é racional ou não.

> A lógica intuicionista é dita construtiva por causa da correspondência entre provas e algoritmos. Portanto, por exemplo, se provamos que $\exists n \in$

> $\mathbb{N}$, $P(n)$, então podemos exibir um natural n que satisfaz a propriedade P. (GIRARD; TAYLOR; LAFONT, 1989, p. 149, tradução nossa)[1].

A ideia de um procedimento mecanizável, i.e. um algoritmo, está no coração da ciência da computação. A lógica intuicionista foca em proposições que possuem uma prova direta, construtiva, i.e. um algoritmo. Assim, pode-se dizer que a lógica clássica é tão importante para a Matemática quanto a intuicionista para a Ciência da Computação.

1.1.2 Cálculo de sequentes

O Cálculo de Sequentes foi criado por Gerard Gentzen em 1935 com o objetivo de estabelecer propriedades de um sistema de dedução natural. Em particular, a propriedade de consistência, a qual significa que nem toda proposição é demonstrável. Durante as últimas treze décadas, o cálculo de sequentes tem sido o interesse central na teoria da prova. Isso deu origem a uma ampla literatura e numerosos resultados.

Definição 1.1.3. *Um* ***sequente*** *possui a forma* $\Gamma \vdash \Delta$*, na qual* Γ *e* Δ *são multiconjuntos finitos de fórmulas. Chamamos* Γ *e* Δ *de* ***antecedente*** *e* ***sucedente****, respectivamente. Uma* ***demonstração*** *para o sequente* $\Gamma \vdash \Delta$ *é uma árvore finita, construída utilizando as regras de inferência do sistema tal que a raiz é* $\Gamma \vdash \Delta$*. Podemos nos referir de maneira mais geral à* Γ *e* Δ *como* ***cedentes****.*

Uma regra tem a forma:

$$\frac{P_1, P_2, \ldots, P_n}{C}\,[regra],$$

a qual nos diz que a conclusão C é verdade sempre que as premissas $P_1, P_2, \ldots, P_n$ são verdadeiras. Neste trabalho, as provas seguem uma análise *bottom-up*, ou seja, as regras são aplicadas de baixo para cima. Um *sequente para a lógica intuicionista* é caracterizado pela presença de somente uma expressão no seu segundo membro, ou seja, Δ é unitário.

Neste trabalho, concentramos nossos esforços no modelo clássico de lógica, pois é intuitivo perceber que se um sequente $\Gamma \vdash \Delta$ é válido em LJ, então também vale para LK, porém o inverso nem sempre é verdade. As regras para sequentes em um modelo clássico, geralmente, são simétricas, portanto essa escolha acarretou a vantagem de diminuirmos a quantidade de regras, transformando um sequente de dois membros (*two-sided*) em um de um membro (*one-sided*).

[1] Intuitionistic logic is called constructive because of the correspondence between proofs and algorithms (the Curry-Howard isomorphism). So, for example, if we prove a formula $\exists n \in \mathbb{N},\ P(n)$, we can exhibit an integer n which satisfies the property P.

1.2 Cálculo-λ

> Um tanto quanto o chassi de um ônibus, que suporta o veículo, mas não é visto pelos seus usuários, versões de cálculo-λ apoiam vários sistemas lógicos importantes e linguagens de programação.
>
> *(Cardone e Hindley)*

Na matemática, é muito comum representar funções por meio de expressões, e.g. $x^2 + 1$, que exprimem como, dado o valor de entrada x, podemos calcular um valor de saída. Comumente chamamos x de *variável* e seu valor é arbitrário (ou abstrato) dentro do domínio da função.

O cálculo-λ foi originalmente proposto por Alonzo Church, na década de 1930, como uma nova notação para funções. Com a finalidade de enfatizar o papel abstrato de uma variável em funções, usamos o símbolo λ junto com as variáveis abstratas, seguidos de um ponto, em frente à expressão. Por exemplo, em vez de $x^2 + 1$, escrevemos $\lambda x.x^2 + 1$. Isso nos permite analisar funções de uma perspectiva abstrata. De fato, não estamos interessados em computar valores concretos para funções.

> Por volta de 1928, ele [Church] começou a construir um sistema formal com o objectivo de proporcionar uma fundação para a lógica que seria mais natural do que a Teoria de Tipos de Russell ou a Teoria de Conjuntos de Zermelo. (CARDONE; HINDLEY, 2006, p. 6, tradução nossa)[2].

Definição 1.2.1 (λ**-termos**). *Dado um conjunto enumerável V de variáveis, o conjunto Λ de λ-termos é assim definido:*

1. *(Variável): Se $x \in V$, então $x \in \Lambda$;*
2. *(Aplicação) Se $M, N \in \Lambda$, então $(MN) \in \Lambda$;*
3. *(Abstração) Se $x \in V$ e $M \in \Lambda$, então $(\lambda x.M) \in \Lambda$.*

[2] Around 1928 he began to build a formal system with the aim of providing a foundation for logic which would be more natural than Russell's type theory or Zermelo's set theory.

Exemplo 1.2.2. *Exemplos de $\lambda - termos$ para $V = \{x, y\}$*

x	$\lambda x.y$	$(\lambda x.x)\ (\lambda x.x)$
$x\ y$	$(\lambda x.x)\ y$	$\lambda x.\lambda y.x$

Considerando o termo $\lambda x.M$, dizemos que M é o *escopo* de λx. Um ocorrência de x no *escopo* de λx é dita ligada. Caso contrário, x ocorre *livre*.

O cálculo-λ possui um operação de substituição: $M[x := N]$, que substitui todas as ocorrências livres de x em M por N.

Exemplo 1.2.3.

$$(\lambda x.xy)[x := w] = (\lambda x.xy)$$
$$(\lambda x.xy)[y := w] = (\lambda x.xw)$$

Obervação 1.2.4. *Na realidade tudo em cálculo-λ é definido usando abstrações e aplicações. Por exemplo, o algarismo '1', ou a soma '+', da expressão $\lambda x.x + 1$ devem ser definidos desse modo. Veja:*

$$1 = \lambda f.\lambda x.fx$$
$$plus = \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx)$$

O cálculo-λ compreende o universo de todas as funções computáveis. Todavia, como sistema lógico, λ[3] é inconsistente (KLEENE, 1935). Posteriormente, na década de 1940, Church desenvolve a Teoria de Tipos Simples a fim de resolver esse problema.

[3] Por simplicidade, a partir daqui por vezes iremos nos referir ao cálculo-λ somente por λ

1.3 Teoria de Tipos

> Tipos são tipos e proposições são proposições; tipos pertencem às linguagens de programação, e proposições à lógica, e aparentemente não há relação entre eles. [...] se fizermos certas suposições sobre ambos lógica e programação, então podemos definir um sistema que é simultaneamente uma lógica e uma linguagem de programação, e na qual proposições e tipos são idênticos.
>
> *(Simon Thompson)*

Nem só de sucessos vive a matemática. Sua história é marcada tanto por acertos quanto por erros e lacunas. Em resposta às três grandes crises da matemática, causadas principalmente pelos paradoxos e informalidade da construção do cálculo, houve grandes avanços sobre os fundamentos da matemática, em especial a Teoria de Conjuntos. Dentre outros, os fundamentos da matemática também incluem a Teoria de Tipos e a Teoria da Prova, discutidas neste trabalho.

Neste capítulo, não é nossa intenção abordar todos os detalhes da Teoria de Tipos. O leitor pode consultar a obra (NEDERPELT; GEUVERS, 2014) para maior compreensão, pois presume-se que já estejamos familiarizados com o assunto, principalmente o cálculo-λ. Por isso, nos debruçaremos sobre os aspectos essenciais dessa teoria. Resgataremos, resumidamente, alguns fatos históricos e, em seguida, entraremos no campo da formalização.

1.3.1 Cálculo-λ simplesmente tipado

Conhecido como cálculo-λ simplesmente tipado, o embrião dessa teoria foi introduzido por Alonzo Church, em 1940 como solução para o fracasso do λ como sistema lógico.

Definição 1.3.1 (O conjunto $\mathbb{T}$ de todos os tipos simples). *Seja $\mathbb{V}$ um conjunto infinito de variáveis de tipo, $\mathbb{V} = \{\alpha, \beta, \gamma, \ldots\}$, o conjunto de tipos simples $\mathbb{T}$ é defino por:*

1. *Tipo variável (tipos básicos): se $\alpha \in \mathbb{V}$, então $\alpha \in \mathbb{T}$;*
2. *Tipo seta (tipos função): se $\tau,\ \sigma \in \mathbb{T}$, então $(\tau \to \sigma) \in \mathbb{T}$.*

Definição 1.3.2 (**λ-termo pré-tipado**). *O conjunto de termos pré-tipados é definido por:*

$$\Lambda_{\mathbb{T}} := \mathbb{V} \mid (\Lambda_{\mathbb{T}}\ \Lambda_{\mathbb{T}}) \mid \lambda V : \mathbb{T}.\Lambda_{\mathbb{T}}$$

Estamos interessados em estabelecer se um termo $M \in \Lambda_{\mathbb{T}}$ é *tipável* e qual é o tipo de M. Para isso, usamos um conjunto de regras de derivação para indicar se uma sentença $\Gamma \vdash M : \sigma$ é *derivável*, ou seja, se M tem tipo σ, assumindo Γ como hipótese.

A notação $M : \alpha$ significa que o termo M é do tipo α ou, equivalentemente, α é um tipo que habita o termo M.

Figura 1 – Regras de derivação para cálculo lambda simplesmente tipado

$$\frac{\Gamma, x : \tau \vdash M : \gamma}{\Gamma \vdash \lambda x : \tau.M : \tau \to \gamma}\, abst \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}\, appl$$

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\, var$$

Exemplo 1.3.3. *Sejam y uma função do tipo $\alpha \to \beta$ e z um argumento do tipo α, a aplicação de y a z (notação yz) deve retornar um objeto do tipo β*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : \alpha \to \beta \vdash y : \alpha \to \beta}\, var \quad \dfrac{}{z : \alpha \vdash z : \alpha}\, var}{y : \alpha \to \beta, z : \alpha \vdash yz : \beta}\, appl}{y : \alpha \to \beta \vdash \lambda z : \alpha.yz : \alpha \to \beta}\, abst}{\vdash \lambda y : \alpha \to \beta.\lambda z : \alpha.yz : (\alpha \to \beta) \to \alpha \to \beta}\, abst$$

Teorema 1.3.4 (**Unicidade**). *Se $\Gamma \vdash M : \alpha$ e $\Gamma \vdash M : \beta$, então $\alpha = \beta$.*

O cálculo-λ simplesmente tipado ($\lambda_{\to}$) é um sistema menos expressivo do que o cálculo-λ. Além disso, não conseguimos tipar o termo $\lambda x.xx$. Porém, as autoaplicações são evitadas e os tipos são únicos.

1.3.2 Cálculo-λ tipado de segunda ordem

No $\lambda_{\to}$, a abstração atua sobre os termos. Dizemos, então, abstração de primeira ordem. A aplicação também é de primeira ordem. No cálculo-λ tipado de segunda ordem ($\lambda 2$) consideramos o tipo de todos os tipos, abstrações e aplicações de segunda ordem. Isso nos garante o tipo para funções polimórficas.

Em $\lambda_{\to}$ o termo $\lambda x : \sigma.M$ depende do termo x. Dizemos que os termos são dependentes dos termos. Já em $\lambda 2$ os termos são dependentes dos tipos: em $\lambda\alpha : *.\lambda x : \alpha.x$, o tipo de que ele dependente é α. A função $\lambda\alpha : *.\lambda x : \alpha.x$ denota que, dado um tipo α,

retorna a função identidade em α (polimorfismo). O simbolo * denota o tipo de todos os tipos.

Um aspecto poderoso dos tipos de segunda ordem é que se pode definir vários tipos de dados, como números naturais, listas e árvores.

Podemos considerar outra extensão do $\lambda_{\rightarrow}$ chamada λ_{ω}, na qual os tipos são dependentes do tipos. Esse sistema permite definir construtores de tipos. Por exemplo, $\lambda\alpha : *.\alpha \rightarrow \alpha$ é uma função que constrói o tipo $\alpha \rightarrow \alpha$ ao ser aplicada ao tipo α.

O tipos podem depender de termos também no sistema λ_P permitindo, e.g., definir famílias de tipos. Tais construções são importantes para definir predicados. Por exemplo, um predicado P acerca dos números naturais possui o tipo $nat \rightarrow *$. Intuitivamente, P é um construtor que dado um número natural retorna uma proposição.

Todos esses sistemas juntos contribuem para a formação do Cálculo de Construções (CC) (COQUAND; HUET, 1988). O CC e suas variantes servem de base para Coq e outros assistentes de prova.

A linguagem implementada por Coq, *Gallina*, se baseia no Cálculo de Construções Indutivas que combina uma lógica de ordem superior e uma linguagem de programação funcional ricamente tipada.

1.4 O Isomorfismo de Curry-Howard

O isomorfismo de Curry-Howard é uma relação direta entre programas de computadores e provas matemáticas, portanto há uma correspondência bijetiva entre sistemas de lógica formal (*proof theory*) e cálculo computacional (*type theory*). Ele também é conhecido como correspondência provas-como-programas ou correspondência fórmulas-como-tipos. O isomorfismo permite duas leituras para a assertiva ($M : \alpha$): M é um termo (programa, expressão) do tipo α ou M é uma prova (derivação) da fórmula α.

Fórmulas $\leftrightarrow$ Tipos

Provas $\leftrightarrow$ Termos

Provabilidade $\leftrightarrow$ *Inhabitation*

Normalização de provas $\leftrightarrow$ Redução de termos

Problemas relacionados à Teoria de Tipos (NEDERPELT; GEUVERS, 2014)

Well-typedness :

Typability $? \vdash M :?$

Type assigment $\Delta \vdash M :?$

Type Checking :

$\Delta \vdash {}^{?}M : A$

Term Finding/Inhabitation :

Term Construction $\Delta \vdash ? : A$

Portanto, problemas de *type checking* tentam verificar se um tipo corresponde a um termo e é decidível. E problemas de *inhabitation* tentam encontrar uma prova de um teorema que, em geral, são indecidíveis.

1.5 Teoria da Prova Estrutural

A *Teoria da Prova* é a área da matemática que estuda os conceitos de prova e demonstrabilidade matemáticas (BUSS, 1998) . Segundo Bus (1998, p. 2), uma prova pode ser vista sobre duas óticas: as **provas sociais** que são discutidas em conversas ou publicadas em artigos e as **provas formais** que consistem em uma *string* de símbolos que satisfazem um conjunto de regras e que provam um teorema, o qual deve ser também expressado como uma *string* de símbolos.

As provas formais podem ser construídas com o auxílio do computador por meio de um assistente de provas. Neste trabalho, optamos pelo assistente de provas Coq pelo motivo de estar construído sobre uma base fundamentada na Teoria de Tipos. Ademais, existem muitos casos com êxito usando Coq, como a formalização completa do Teorema de Feit-Thompson[4] e o Teorema das Quatro Cores[5], por exemplo. O assistente verifica uma prova de um teorema automaticamente. Esta tarefa é geralmente simples, mas encontrar uma prova é realmente incerto, como discutido na seção anterior.

A *Teoria da Prova Estrutural* é a subárea da Teoria da Prova que estuda a estrutura geral e as propriedades das provas matemáticas (NEGRI; PLATO; RANTA, 2008, p. xi). Uma questão central nesse âmbito é a consistência de um sistema lógico.

O teorema da eliminação do corte (ou *Hauptsatz*, que significa resultado principal) é de suma importância, uma vez que, em geral, implica a consistência e a *propriedade subfórmula*[6] para um dado sistema. Ele assinala que qualquer prova em cálculo de sequentes que faz uso da regra do corte pode ser substituída por outra que não a utiliza. A prova procede por indução na ordem lexicográfica (peso da fórmula, altura do corte) que gera muitos casos. Por essa razão, a demonstração poderia ser propensa a erros na hipótese de recorremos a uma prova informal.

[4] http://www.msr-inria.fr/news/feit-thomson-proved-in-coq/
[5] https://math-comp.github.io/math-comp/
[6] Dada a derivação de $\Gamma \vdash \Delta$, todos os sequentes são compostos apenas por subfórmulas das fórmulas de Γ e Δ.

1.6 O Assistente de Provas: Coq

O assistente de prova Coq (BERTOT; CASTÉRAN, 2013) é uma ferramenta gratuita para computador delineada sobre cálculo de construções indutivas, que é baseado em $\lambda_{\rightarrow}$. O Coq fornece uma linguagem formal para escrever definições matemáticas e teoremas em um ambiente de desenvolvimento semi-interativo de provas verificadas por máquina. Ademais, oferece uma linguagem de táticas que permite ao usuário construir uma prova, além de dar suporte à construção de novas táticas pelo usuário.

A formalização da matemática, a especificação e verificação das propriedades de sistemas, a certificação das propriedades de linguagens de programação e ensino de matemática e ciência da computação são exemplos do uso atual desse assistente.

Caso o leitor não se sinta à vontade com ambiente de Coq, sugerimos, também, a leitura do livro *Software Foundations* do professor Benjamin C. Pierce et al, disponível gratuitamente[7]. Essa obra nos traz uma linguagem bastante cognoscível abordando os fundamentos e conceitos mais avançados. Além disso, há uma verdadeira imersão na ferramenta mediante exercícios e desafios propostos, pois, em se tratando de linguagens de programação e afins, a prática é imprescindível.

Considere, por exemplo, o código a seguir.

```
1 Definition soma (x y : nat) := x + y. (* soma is defined *)
2 Check soma. (* soma : nat → nat → nat *)
3 Check 2. (* 2 : nat *)
4 Eval compute in soma 3 4. (* = 7 : nat *)
```

Na primeira linha estamos atribuindo um nome a uma expressão. O comando **Check** é usado para verificar se uma expressão é bem formada. Ele retorna o tipo da expressão. O tipo nos diz em qual contexto a expressão pode ser usada. Por exemplo, na linha 2, a expressão *soma* é uma função que espera receber dois números naturais como entrada, x e y, e retorna outro número natural resultante da soma $x + y$.

```
1  Fixpoint fibonacci (n : nat) : nat :=
2    match n with
3    O ⇒ 1
4      | (S n') ⇒ match n' with
5          O ⇒ 1
6          | (S n'') ⇒ (fibonacci n') + (fibonacci n'')
7      end
8    end.
9  (* fibonacci is defined *)
10 (* fibonacci is recursively defined (decreasing on 1st argument) *)
```

[7] Acesse: https://www.cis.upenn.edu/ bcpierce/sf/current/index.html

```
11 Check S. (* S :  nat →  nat *)
12 Eval compute in (fibonacci 7). (* = 21 : nat *)
```

Código 1.1 – Sequência de Fibonacci em Coq

Podemos também programar funções recursivas, e.g. a sequência de Fibonacci composta por números inteiros, começando normalmente por 0 e 1, na qual, cada termo subsequente corresponde a soma dos dois anteriores: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$

$$F_0 = 1 \tag{1.1}$$

$$F_1 = 1 \tag{1.2}$$

$$F_n = F_{n-1} + F_{n-2}, para\ n \geq 2 \tag{1.3}$$

Sua versão em Coq está grafada no Código 1.1, na qual S é uma função que retorna o sucessor de um número natural e **Fixpoint** é a palavra-chave para definir funções recursivas. O Coq nos informa que a função *fibonacci* foi definida recursivamente e que ela decresce. Isso garante que não entraremos em um *loop* infinito. Todavia, podemos trabalhar com estruturas infinitas como um *stream*, nesse caso usamos **CoFixpoint**.

Uma característica muito importante em Coq é a capacidade de realizar definições indutivas, ou tipos indutivos. Por meio deles, podemos realizar provas por indução. O Coq ainda suporta polimorfismo, construção de estruturas de dados, *pattern-matching* e acompanha bibliotecas para aritmética em $\mathbb{N}$, $\mathbb{Z}$ e $\mathbb{Q}$, bibliotecas sobre listas, relações, lógica etc.

A seguinte definição indutiva declara um tipo chamado dia, cujos membros são segunda, terça etc.

```
Inductive dia : Type :=
  | segunda | terça | quarta | quinta | sexta | sábado | domingo.
```

Perceba que essa definição indutiva é semelhante ao *enum type* de Java ou *enumeration* de C++. Definições assim nos permite raciocinar com provas por indução estrutural, inclusive o Coq gera automaticamente esse princípio. Examine o próximo exemplo.

Exemplo 1.6.1. *Podemos definir os números naturais da seguinte maneira: o zero é um número natural, se n é um número natural, então o sucessor de n é um número natural. Quando definimos nat em Coq, isto é, o tipo número natural, é gerado automaticamente o princípio de indução nat_ind.*

```
Inductive nat :=
    O : nat | S : nat → nat.  (* nat_ind is defined *)
Check nat_ind.
(* nat_ind =
```

```
: forall P : nat →  Prop,
      P 0 →
(forall n : nat, P n →  P (S n)) →
      forall n : nat, P n *)
```

Podemos conferir aferindo o tipo de nat_ind que o princípo de indução para os números naturais está correto: dada um proposição P sobre os números naturais e se (P 0) é válido e se para todos n natural (P n) é válido podemos provar que (P S n), ou seja, também vale para seu sucessor, então (P n) vale para todo n natural.

Neste trabalho, utilizamos o Coq para formalizar a lógica linear por meio de definições indutivas e provar metateoremas acerca dessa lógica, geralmente por meio de indução. As definições, funções recursivas e tipos indutivos são suficientes para a codificação dessa lógica. Abordaremos um pouco o ambiente para provas em Coq.

Como ambiente semi-interativo de provas, existe outra tela que nos mostra o andamento delas.

```
Theorem le_n: forall n : nat, n ≤ n.
```

A cada tática que aplicamos, obviamente, esta tela se modificará, por isso é inviável exibirmos uma demonstração passo-a-passo nesta dissertação.

```
1 subgoal
______________________________________(1/1)
forall n : nat, n ≤ n
```

Porém, podemos resumir algumas táticas utilizadas em Coq.

Tabela 1 – Táticas que podem ser aplicadas na conclusão.

Quando o objetivo é ...	**... use a tática**
muito simples	*auto*, *tauto* ou *firstorder*
p ∧ q	*split*
p ∨ q	*left* ou *right*
p → q	*intro*
∼ p	*intro*
p ↔ q	*split*
uma hipótese	*assumption*
forall x, P	*intro*
exists x, P	*exists* t

Além dessas táticas Coq tem suporte à programação de novas táticas pelo usuário por meia da linguagem Ltac.

Como um sistema de desenvolvimento de prova, Coq fornece métodos de prova interativos e uma linguagem tática para permitir que o usuário defina seus próprios métodos de prova. Como uma plataforma para a formalização da matemática ou o desenvolvimento

Tabela 2 – Táticas que podem ser aplicadas na hipótese.

Para usar a hipótese H ...	**... use a tática**
p ∧ q	*destruct* H *as* [H_1 H_2]
p ∨ q	*destruct* H *as* [H_1 | H_2]
p → q	*apply* H
∼ p	*apply* H ou *elim*
p ↔ q	*apply* H
False	*contradction*
forall x, P	*apply* H
exists x, P	*destruct* H *as* [x G]
a = b	*rewrite* H ou *rewrite* ← H

Tabela 3 – Conjunto de táticas auxiliares.

Se você deseja ...	**... então use**
provar por contradição p ∧ ∼ p	*absurd* p
simplificar expressões	*simpl*
provar via lema intermediário p	*cut* p
provar por indução sobre t	*induction* t
simular que você finalizou	*admit* ou *give_up*
computar t	*Eval compute in* t
imprimir a definição de p	*Print* p
checar o tipo de t	*Check* t
procurar teoremas sobre p	*SearchAbout* p
repetir uma tática n vezes	*do* n tática
repetir uma tática em um loop	*repeat* tática

Tabela 4 – Exemplos de táticas preexistentes em Ltac.

$tac_1; tac_2$	Aplica tac_1 e tac_2 em todos os *subgoals*
$tac; [tac_1 \| \ldots \| tac_i \| \ldots \| tac_n]$	Aplica tac e tac_i para o i-ésimo *subgoal*
do n tac	Aplica tac n vezes
repeat tac	Repete tac até a falhar
try tac	Tenta aplica tac
$first[tac_1 \| \ldots \| tac_i \| \ldots \| tac_n]$	Aplica a primeira tac_i que não falha
$solve[tac_1 \| \ldots \| tac_i \| \ldots \| tac_n]$	Aplica a primeira tac_i que resolve

de programas, Coq fornece suporte para notações de alto nível, conteúdo implícito e vários outros tipos de macros úteis.

Capítulo 2

A Lógica Linear em Coq

2.1 A Lógica Linear

> Segunda vez foi Jesus a Caná da Galiléia, onde da água fizera vinho.
>
> *(João 4:46)*

As lógicas que possuem restrições sobre regras estruturais usuais são conhecidas como lógicas subestruturais. Nesse sentido, a lógica linear (LL) é uma das lógicas subestruturais mais significativas e encontra estudos em muitas áreas, a saber: na Computação (ALEXIEV, 1994) e (MATSKIN, 2003), Biologia Molecular (CHAUDHURI; DESPEYROUX, 2013), Linguística (MOOT; PIAZZA, 2001) e Linguagens de Programação (MACKIE, 1994).

Sabe-se que a lógica usual trata da verdade, isto é, do verdadeiro ou falso. De fato, tendo provado um teorema na lógica habitual, clássica ou intuicionista, podemos usá-lo quantas vezes forem necessárias. Essa circunstância é caracterizada, em cálculo de sequentes, pelo uso das regras *weakening* e *contraction*. Porém, observe os exemplos abaixo:

Exemplo 2.1.1. *Sejam X e Y conjuntos, A a proposição $X \equiv Y$, B a proposição $X \subseteq Y$ e C a proposição $Y \subseteq X$. Então, na representação simbólica temos que $(A \to B) \wedge (A \to C)$. Podemos concluir que $A \to (B \wedge C)$, ou seja, se $X \equiv Y$, então $X \subseteq Y$ e $Y \subseteq X$.*

Exemplo 2.1.2. *Agora, suponha que, em uma lanchonete, com R$1,00 (A) podemos comprar uma xícara de café (B) e uma fatia de bolo (C) que também custa R$1,00. Em símbolos, temos novamente:*

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

Todavia, não deveríamos poder concluir que $A \to (B \wedge C)$, ou seja, com apenas R$1,00,

> *não podemos saborear uma fatia de bolo com café, pois uma vez gasto (consumido) o dinheiro, não podemos usá-lo novamente.*

Para o filósofo francês Jean-Paul Sartre, viver é isso: ficar se equilibrando o tempo todo, entre escolhas e consequências. As nossas vidas são cingidas por restrições, e.g. espaço e tempo limitados. Uma vez que nossos recursos são limitados, o mau gerenciamento deles pode acarretar em circunstâncias indesejáveis, um desequilíbrio. Por isso, se pretendemos fazer escolhas sábias, a administração desses recursos é imprescindível. Tais escolhas podem parecer dilemas exclusivamente humanos, mas eles não são: computadores, também, enfrentam as mesmas restrições.

A lógica linear (LL) (GIRARD, 1995), desenvolvida pelo lógico francês Jean-Yves Girard, em 1987, deve ser vista como uma extensão da lógica usual, segundo o próprio. Uma característica diferencial é que a LL é sensível ao uso dos recursos. O controle da utilização dos recursos é demonstrado pelo fato de que as regras *weakening* e *contraction* não são admissíveis em geral, mas controlada pelas modalidades ! (*bang*) ou ? (*quest*), chamadas exponenciais.

Observe que a conjunção possui significados diferentes entre os exemplos 3.1.1 e 3.1.2. O mesmo acontece com a disjunção. Assim, LL refina o modelo clássico dividindo seus conectivos $(\wedge, \vee)$ em multiplicativos e aditivos Tabela 5.

Tabela 5 – Conectivos da Lógica Linear

	Aditivos	**Multiplicativos**
Conjunção	& (with)	$\otimes$ (tensor)
Disjunção	$\oplus$ (plus)	$\parr$ (par)

A implicação é representada por $\multimap$[1] e carrega o significado de consumo e produção ou a conversão do recurso, como em uma reação química, por exemplo.

- $A \multimap B$: o recurso A é consumido e B é produzido.

  E.g. **água** $\multimap$ **vinho**, esse exemplo está em consonância com a epigrafe atual e significa que, uma vez transformada a água em vinho, não temos mais a água, mas somente o vinho.

- $A \otimes B$: ambos os recursos A e B estão presentes.

  E.g. **real** $\otimes$ **real** $\multimap$ **bolo**, expressa que com dois reais podemos comprar uma fatia de bolo.

[1] Também conhecido como *lollipop* devido ao seu formato de pirulito

- $A \oplus B$: ou o recurso A ou o B está presente, mas não ambos.
  E.g. **bilhete_loteria** $\multimap$ **ganha** $\oplus$ **perde**, significa que um bilhete de loteria pode ser usado para ganhar ou perder, mas não pode-se escolher o resultado.

- A & B: pode-se escolher, exclusivamente, o recurso A ou B.
  E.g. **real** $\multimap$ **chá & café**, significa que com um real podemos escolher entre uma xícara de chá e uma de café.

- $!A$: pode-se obter um arbitrário número de cópias do recurso A.
  E.g. **!A**, significa que temos cópias do recurso A *ad libitum*
  E.g. **!(água** $\multimap$ **real)**, expressa a faculdade de vender água por um real tantas vezes quanto necessário

Observe que, nos exemplos anteriores, com a conjunção multiplicativa, a fórmula $(A \multimap B) \otimes (A \multimap C) \multimap (A \multimap (B \otimes C))$ não é válida. Por outro lado, com a conjunção aditiva, a fórmula $(A \multimap B) \,\&\, (A \multimap C) \multimap (A \multimap (B \,\&\, C))$ é válida.

Segue a sintaxe da lógica linear, na qual denotamos F de fórmula ou expressão linear. As regras de derivação da LL estão descritas na Figura 4.

Definição 2.1.3 (Sintaxe)**.** *Um fórmula em LL pode ser definida sintaticamente por meio da seguinte gramática:*

$$F ::= \underbrace{A \mid 0 \mid 1 \mid \top \mid \bot}_{\textit{Unidades}} \mid \underbrace{F_1 \,\&\, F_2 \mid F_1 \otimes F_2}_{\textit{Conjunções}} \mid \underbrace{F_1 \oplus F_2 \mid F_1 \parr F_2}_{\textit{Disjunções}} \mid \underbrace{?F \quad \mid \quad !F}_{\textit{Exponenciais}}$$

Esta definição indutiva significa que uma expressão em LL poderá ser concebida somente por meio dos seus construtores, e.g. para $A \otimes B$ teremos Tensor A B.

```
Inductive lexp : Type :=
  | Atom   : var → lexp (* Proposição atômica *)
  | Perp   : var → lexp (* Negação de uma proposição atômica *)
  | Top    : lexp
  | Bot    : lexp
  | Zero   : lexp
  | One    : lexp
  | Tensor : lexp → lexp → lexp
  | Par    : lexp → lexp → lexp
  | Plus   : lexp → lexp → lexp
  | With   : lexp → lexp → lexp
  | Bang   : lexp → lexp
  | Quest  : lexp → lexp.
```

Devido aos problemas com *character encodings*, foi estabelecido o uso de caracteres no Código Padrão Americano para o Intercâmbio de Informação (ASCII, do inglês *American Standard Code for Information Interchange*). Portanto, alguns símbolos apresentados neste trabalho foram especificados em Coq conforme a Tabela 6, sendo F e G expressões e A um átomo.

Tabela 6 – Convenção de símbolos de LL em Coq

Notação em LL	**Notação em Coq**
$F \parr G$	$F \ \$ \ G$
$F \otimes G$	$F \ ** \ G$
$F^{\perp}$	F°
A	A^{+}
$A^{\perp}$	A^{-}

2.2 Cálculo de Sequentes para LL

Nesta seção, introduzimos as regras do sistema para LL e, ao mesmo tempo, formalizamos tais regras em Coq. Optamos pela versão *one-sided* de sequentes da lógica linear.

O cálculo de sequentes é codificado da seguinte maneira (ver Figura 2):

Figura 2 – Cálculo de sequentes em Coq

$$\frac{P_1, P_2, \ldots, P_n}{C} \quad \rightsquigarrow \quad (P_1 \ \wedge \ P_2 \ \wedge \ \ldots \wedge P_n) \ \rightarrow \ C \quad \rightsquigarrow \ P_1 \ \rightarrow \ P_2 \ \rightarrow \ \ldots \rightarrow P_n \rightarrow C$$

Isso nos mostra que com $P_1, P_2, \ldots, P_n$ podemos concluir C. A linha horizontal representa uma implicação lógica e as conjunções podem ser simplificadas com implicações.

Figura 3 – Regras de derivação para lógica linear

Axioma inicial e regra cut

$$\frac{}{A \vdash A}\,[id] \qquad \frac{\Gamma \vdash A, \Delta \quad \Theta, A \vdash \Lambda}{\Gamma, \Theta \vdash \Delta, \Lambda}\,[cut]$$

Regras à direita

$$\frac{}{\Gamma \vdash \top, \Delta}\,[\top_R] \qquad \frac{\Gamma \vdash F, \Delta \quad \Gamma \vdash G, \Delta}{\Gamma \vdash F \,\&\, G, \Delta}\,[\&_R]$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\,[\bot_R] \qquad \frac{\Gamma \vdash , F, G, \Delta}{\Gamma \vdash F \parr G, \Delta}\,[\parr_R]$$

$$\frac{}{\vdash 1}\,[1_R]$$

$$\frac{\Gamma \vdash F, \Delta \quad \Theta \vdash G, \Lambda}{\Gamma, \Theta \vdash F \otimes G, \Delta, \Lambda}\,[\otimes_R] \qquad \frac{\Gamma \vdash F_i, \Delta}{\Gamma \vdash F_1 \oplus F_2, \Delta}\,[\oplus_{Ri}]$$

$$\frac{\Gamma, F \vdash \Delta}{\Gamma \vdash F^{\perp}, \Delta}\,[.^{\perp_R}] \qquad \frac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \multimap G, \Delta}\,[\multimap_R]$$

Regras à esquerda

$$\frac{\Gamma \vdash F, \Delta}{\Gamma, F^{\perp} \vdash \Delta}\,[.^{\perp_L}] \qquad \frac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \oplus G \vdash \Delta}\,[\oplus_L]$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}\,[1_L] \qquad \frac{\Gamma, F_i \vdash \Delta}{\Gamma, F_1 \,\&\, F_2 \vdash \Delta}\,[\&_{Li}]$$

$$\frac{}{\bot \vdash}\,[\bot_L]$$

$$\frac{}{\Gamma, 0 \vdash \Delta}\,[0_L] \qquad \frac{\Gamma, F \vdash \Delta \quad \Theta, G \vdash \Omega}{\Gamma, \Theta, F \parr G \vdash \Delta, \Omega}\,[\parr_L]$$

$$\frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \otimes G \vdash \Delta}\,[\otimes_L] \qquad \frac{\Gamma \vdash F, \Delta \quad \Theta, G \vdash \Omega}{\Gamma, \Theta, F \multimap G \vdash \Delta, \Omega}\,[\multimap_L]$$

Regras estruturais

$$\frac{!\Gamma \vdash F, ?\Delta}{!\Gamma \vdash !F, ?\Delta}\,[!] \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma, !F \vdash \Delta}\,[!_d] \qquad \frac{\Gamma, !F, !F \vdash \Delta}{\Gamma, !F \vdash \Delta}\,[!_c] \qquad \frac{\Gamma \vdash \Delta}{\Gamma, !F \vdash \Delta}\,[!_w]$$

$$\frac{!\Gamma, F \vdash ?\Delta}{!\Gamma, ?F \vdash ?\Delta}\,[?] \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma \vdash ?F, \Delta}\,[?_d] \qquad \frac{\Gamma \vdash ?F, ?F, \Delta}{\Gamma \vdash ?F, \Delta}\,[?_c] \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?F, \Delta}\,[?_w]$$

Exemplo 2.2.1. *Para as regras id e $\otimes_R$, na Figura 3, teremos:*

$$\frac{}{A \vdash A}\,[id] \qquad\qquad \frac{\Gamma \vdash F, \Delta \quad \Theta \vdash G, \Lambda}{\Gamma, \Theta \vdash F \otimes G, \Delta, \Lambda}\,[\otimes_R]$$

```
Inductive derivation: Multiset → Multiset → Prop :=
    | der_id : forall A, {{A}} ⊢ {{A}}
    | der_tensorR : forall F G Δ Γ Λ Θ,
        Γ ⊢ {{F}} ∪ Δ →
            Θ ⊢ {{G}} ∪ Λ →
                Γ ∪ Θ ⊢ {{F ⊗ G}} ∪ Δ ∪ Λ
```

A negação em LL é definida por equações de De Morgan:

Definição 2.2.2. *Sejam A um átomo[2] e F uma fórmula em LL, definimos a negação de F indutivamente:*

$$\begin{aligned} (A^\perp)^\perp &= A \\ (A)^\perp &= A^\perp \\ \perp^\perp &= 1 \\ \top^\perp &= 0 \end{aligned} \qquad \begin{aligned} 1^\perp &= \perp \\ 0^\perp &= \top \\ (?F)^\perp &= !F^\perp \\ (!F)^\perp &= ?F^\perp \end{aligned} \qquad \begin{aligned} (F \otimes G)^\perp &= F^\perp \parr G^\perp \\ (F \parr G)^\perp &= F^\perp \otimes G^\perp \\ (F \,\&\, G)^\perp &= F^\perp \oplus G^\perp \\ (F \oplus G)^\perp &= F^\perp \,\&\, G^\perp \end{aligned}$$

Lema 2.2.3. *[**A negação é involutiva**] Seja F uma fórmula em LL, então $(F^\perp)^\perp = F$.*

```
Theorem ng_involutive: forall F: lexp, (F^⊥)^⊥ = F.
Proof.(* A prova segue por indução em F *) Qed.
```

As provas descritas neste trabalho serão omitidas, mas estão completas nos arquivos de Coq disponíveis em <https://github.com/brunofx86/LL>.

A negação $(.)^\perp$ permite simplificar o cálculo de sequentes em uma versão *one-sided* devido a dualidade dos conectivos. O cálculo de sequentes *one-sided* para a lógica linear recebe nesta dissertação a divisão feita por Jean-Marc Andreoli (ANDREOLI, 1992).

- Sistema com sequente monádico (Σ_1): o sequente consiste de um simples multiconjunto de fórmulas, e.g $\vdash \Gamma, \Delta$. (ver seção **2.2.1**)
- Sistema com sequente diádico (Σ_2): o sequente consiste de um par de multiconjuntos de fórmulas, e.g $\vdash \Theta : \Gamma$. Nomeamos Θ e Γ de contextos clássico e linear, respectivamente. (ver seção **2.2.2**)

[2] *Chamamos um átomo A e sua versão negada $A^\perp$ de literais.*

2.2.1 O sistema monádico

Visto que os conectivos ($\otimes$ e $\parr$, & e $\oplus$, ? e !) e as unidades (1 e $\bot$, 0 e $\top$) são duais, podemos reduzir o número de regras em pelo menos a metade em LL. Pois, no sistema *two-sided* (Figura 3) há duas regras para cada conectivo (*left* e *right*), enquanto que no modelo *one-sided* só há uma regra para cada conectivo. Dessa forma, compõe-se um novo sistema de regras derivado do sistema clássico, o qual denominamos simplesmente de *sistema monádico* (ver Figura 4).

Figura 4 – Regras de derivação para lógica linear *one-sided* monádico

Sejam A um literal, F e G fórmulas e Γ um multiconjunto de fórmulas:

$$\frac{}{\vdash A, A^{\bot}}\,[init] \qquad \frac{\vdash \Gamma, F, G}{\vdash \Gamma, F \parr G}\,[\parr] \qquad \frac{}{\vdash \Gamma, \top}\,[\top]$$

$$\frac{\vdash \Gamma, F_i}{\vdash \Gamma, F_1 \oplus F_2}\,[\oplus_i] \qquad \frac{}{\vdash 1}\,[1] \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\,[\bot]$$

$$\frac{\vdash \Gamma, F \qquad \vdash \Delta, G}{\vdash \Gamma, F \otimes G, \Delta}\,[\otimes] \qquad \frac{\vdash \Gamma, F \qquad \vdash \Gamma, G}{\vdash \Gamma, F \,\&\, G}\,[\&]$$

$$\frac{\vdash \Gamma, ?F, ?F}{\vdash \Gamma, ?F}\,[?_c] \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?F}\,[?_w] \qquad \frac{\vdash ?\Gamma, F}{\vdash ?\Gamma, !F}\,[!] \qquad \frac{\vdash \Gamma, F}{\vdash \Gamma, ?F}\,[?_d]$$

Dizemos que A e B são equivalentes se, para qualquer Γ, o sequente $\vdash \Gamma, A$ é derivável se, e somente se, $\vdash \Gamma, B$ é derivável. Denotamos esse fato por $A \equiv B$.

Figura 5 – Algumas equivalências tipicas

$$A \otimes 1 \equiv A \qquad A \parr \bot \equiv A$$
$$A \otimes B \equiv B \otimes A \qquad A \parr B \equiv B \parr A$$
$$A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C \qquad A \parr (B \parr C) \equiv (A \parr B) \parr C$$

$$A \oplus 0 \equiv A \qquad A \,\&\, \top \equiv A$$
$$A \oplus B \equiv B \oplus A \qquad A \,\&\, B \equiv B \parr A$$
$$A \oplus (B \oplus C) \equiv (A \oplus B) \oplus C \qquad A \,\&\, (B \,\&\, C) \equiv (A \,\&\, B) \parr C$$

A Figura 5 indica que os conectivos aditivos e multiplicativos possuem as propriedades de associatividade, comutatividade e elemento neutro. Outras equivalências interessantes são: $!(A \,\&\, B) \equiv (!A \otimes !B)$ e $?(A \oplus B) \equiv (?A \parr ?B)$. Compare-as com a identidade $e^{(a+b)} = e^a \times e^b$. É daí que inferimos a nomenclatura: *exponenciais*.

Não denotamos a implicação linear porque podemos defini-la por $F \multimap G \equiv F^{\bot} \parr G$. Assim, a negação de A é representada por $A^{\bot} \equiv A \multimap \bot$. Com efeito, $A^{\bot} \equiv A \multimap \bot \equiv$

$A^{\perp} \parr \perp \equiv A^{\perp}$.

2.2.2 O sistema diádico

O sistema diádico, como dito anteriormente, separa os contextos linear e clássico (ver Figura 6). A formalização da LL em Coq, nesta dissertação, está focada no sistema Σ_2, porquanto ele contém um número menor de regras e é equivalente ao sistema Σ_1 e à lógica linear puramente dita.

Figura 6 – Regras de derivação para lógica linear *one-sided* diádico

Sejam A um literal, F e G fórmulas e Γ, Θ e Δ multiconjuntos de fórmulas:

$$\frac{}{\vdash \Theta \ : \ A, A^{\perp}}\,[init] \qquad \frac{}{\vdash \Theta \ : \ \Gamma, \top}\,[\top] \qquad \frac{}{\vdash \Theta \ : 1}\,[1]$$

$$\frac{\vdash \Theta \ : \ \Gamma, F, G}{\vdash \Theta \ : \ \Gamma, F \parr G}\,[\parr] \qquad \frac{\vdash \Theta \ : \ \Gamma}{\vdash \Theta \ : \ \Gamma, \perp}\,[\perp]$$

$$\frac{\vdash \Theta \ : \ \Gamma, F \qquad \vdash \Theta \ : \ \Delta, G}{\vdash \Theta \ : \ \Gamma, F \otimes G, \Delta}\,[\otimes] \qquad \frac{\vdash \Theta, F \ : \ \Gamma}{\vdash \Theta \ : \ \Gamma, ?F}\,[?]$$

$$\frac{\vdash \Theta \ : \ \Gamma, F \qquad \vdash \Theta \ : \ \Gamma, G}{\vdash \Theta \ : \ \Gamma, F \,\&\, G}\,[\&] \qquad \frac{\vdash \Theta, F \ : \ \Gamma, F}{\vdash \Theta, F \ : \ \Gamma}\,[Ab]$$

$$\frac{\vdash \Theta \ : \ \Gamma, F_i}{\vdash \Theta \ : \ \Gamma, F_1 \oplus F_2}\,[\oplus_i] \qquad \frac{\vdash \Theta \ : \ F}{\vdash \Theta \ : \ !F}\,[!]$$

Para trabalhar com multiconjuntos, desenvolvemos um módulo *multiset* baseado na biblioteca CoLoR[3](*a Coq Library on Rewriting and termination*). Usufruímos das definições e teoremas da estrutura de dados *multiset* de CoLoR e acrescentamos propriedades, com suas respectivas demostrações, úteis para as provas exibidas neste trabalho (ver Tabela 7).

Tabela 7 – Notação para multiconjuntos em Coq

Notação em Coq	**Significado**
{ }	multiconjunto vazio
{{a}}	multiconjunto unitário
a # Γ	multiplicidade de a em Γ
Γ / a	remoção de a em Γ
Γ =mul= Δ	igualdade entre multiconjuntos
$\Gamma \cup \Delta$	união entre multiconjuntos

Dado $\Gamma = \{A_1, A_2, ..., A_n\}$ um multiconjunto de fórmulas, assim podemos definir as seguintes operações sobre Γ:

[3] disponível em http://color.inria.fr/

$$\Gamma^{\perp} = \{A_1^{\perp}, A_2^{\perp}, \ldots, A_n^{\perp}\} \qquad ?\Gamma = \{?A_1, ?A_2, \ldots, ?A_n\} \qquad !\Gamma = \{!A_1, !A_2, \ldots, !A_n\}$$

Para realizarmos esse raciocínio em Coq, utilizamos a função *List.map* que, dados um termo f do tipo $\alpha \to \beta$ e uma lista L de termos do tipo α, mapeia todos os elementos de L em elementos do tipo β por meio de f.

```
Check map. (* map : forall  A B : Type, (A →  B) →  list A →  list B *)
```

A próxima definição nos permitirá demonstrar metateoremas da LL usando indução. Assim, faremos um pequena modificação nas regras para o sistema Σ_2 adicionando a altura relacionada a cada regra (ver Figura 7).

Definição 2.2.4 (Altura de uma regra). *Seja P_n a n-ésima premissa de uma regra em cálculo de sequentes, a altura h de uma instância dessa regra em uma derivação é definida como*

$$h = \begin{cases} 0 & , \textit{ caso não haja premissas;} \\ max(h(P_1), h(P_2), \ldots, h(P_n)) + 1 & , \textit{ caso contrário.} \end{cases}$$

Exemplo 2.2.5. *Para a regra & teremos:*

$$\frac{\begin{array}{cc} \pi_1 & \pi_2 \\ \vdots & \vdots \\ \vdash \Theta \ : \ \Gamma, F & \vdash \Theta \ : \ \Gamma, G \end{array}}{\vdash \Theta \ : \ \Gamma, F \,\&\, G}\ [\&]$$

$h = max(\pi_1, \pi_2) + 1.$

Se fomos atentos até este ponto, constatamos que o sistema Σ_2 não possui as regras $?_c$ e $?_w$ requeridas para manter a qualidade do controle de recursos em LL. Isso ocorre porque tais regras não são explícitas, mas inferidas pelo sistema. Se os sistemas Σ_1 e Σ_2 forem equivalentes, teremos como corolário que Σ_2 admite *weakening* e *contraction*. De fato, eles são equipolentes.

Por causa da noção interna de permutação em multiconjuntos, e.g. {a, a, b, c} =mul= {b, a, c, a}, não necessitamos de uma regra *exchange*. De fato, provamos em Coq estes teoremas.

```
Theorem der_exchange: forall n Θ Γ Δ, Δ =mul= Γ →  n ⊢ Θ : Δ →   n ⊢ Θ : Γ.
Theorem der_exchange_classic: forall n Θ Λ Γ, Θ =mul= Λ →  n ⊢ Θ : Γ →   n ⊢ Λ: Γ.
```

A prova desses teoremas é facilitada pela demonstração do lema a seguir.

Figura 7 – Regras de derivação para LL com altura

Sejam A um literal, F e G fórmulas e Γ, Θ e Δ multiconjuntos de fórmulas:

$$\frac{}{{}_0 \vdash \Theta \ : \ A, A^{\perp}}\,[init] \qquad \frac{}{{}_0 \vdash \Theta \ : \ \Gamma, \top}\,[\top] \qquad \frac{}{{}_0 \vdash \Theta \ : 1}\,[1]$$

$$\frac{{}_n \vdash \Theta \ : \ \Gamma, F, G}{{}_{n+1} \vdash \Theta \ : \ \Gamma, F \parr G}\,[\parr] \qquad \frac{{}_n \vdash \Theta \ : \ \Gamma}{{}_{n+1} \vdash \Theta \ : \ \Gamma, \perp}\,[\perp]$$

$$\frac{{}_{n_1} \vdash \Theta \ : \ \Gamma, F \qquad {}_{n_2} \vdash \Theta \ : \ \Delta, G}{{}_{max(n_1,\ n_2)+1} \vdash \Theta \ : \ \Gamma, F \otimes G, \Delta}\,[\otimes] \qquad \frac{{}_n \vdash \Theta, F \ : \ \Gamma}{{}_{n+1} \vdash \Theta \ : \ \Gamma, ?F}\,[?]$$

$$\frac{{}_{n_1} \vdash \Theta \ : \ \Gamma, F \qquad {}_{n_2} \vdash \Theta \ : \ \Gamma, G}{{}_{max(n_1,\ n_2)+1} \vdash \Theta \ : \ \Gamma, F \,\&\, G}\,[\&] \qquad \frac{{}_n \vdash \Theta, F \ : \ \Gamma, F}{{}_{n+1} \vdash \Theta, F \ : \ \Gamma}\,[copy]$$

$$\frac{{}_n \vdash \Theta \ : \ \Gamma, F_i}{{}_{n+1} \vdash \Theta \ : \ \Gamma, F_1 \oplus F_2}\,[\oplus_i] \qquad \frac{{}_n \vdash \Theta \ : \ F}{{}_{n+1} \vdash \Theta \ : \ !F}\,[!]$$

Lema 2.2.6. *Dados Θ_1, Θ_2, Γ_1 e Γ_2 multiconjuntos de fórmulas e n natural, se Θ_1 =mul= Θ_2, Γ_1 =mul= Γ_2 e o sequente ${}_n \vdash \Theta_1 : \Gamma_1$ é demonstrável, então podemos deduzir ${}_n \vdash \Theta_2 : \Gamma_2$.*

```
Lemma sig2h_der_compat : forall Θ Λ Γ Δ,
    Θ =mul= Λ → Γ =mul= Δ → n ⊢ Θ : Γ → n ⊢ Λ : Δ.
Proof. (* A prova prossegue por indução na hipótese n ⊢ Θ : Γ *) Qed.
```

Note que a igualdade entre multiconjuntos é uma relação de equivalência:

- (Reflexiva) M =mul= M;
- (Simétrica) Se M =mul= N, então N =mul= M;
- (Transitiva) Se M =mul= N e N =mul= P, então M =mul= P.

```
Notation "X =mul= Y" := (meq X Y) (at level 70).
Lemma meq_refl  : forall M, M =mul= M.
Lemma meq_sym   : forall M N, M =mul= N → N =mul= M.
Lemma meq_trans : forall M N P, M =mul= N → N =mul= P → M =mul= P.

Instance meq_Equivalence : Equivalence meq.
```

Desse modo, podemos tirar proveito da biblioteca de morfismos de Coq[4]. Com a prova de *sig2h_morphism*, poderemos substituir um multiconjunto em sequente por outro que seja equivalente, ou seja, que seja uma permutação (Teorema 3.2.6).

[4] Para saber mais acesse: https://coq.inria.fr/library/Coq.Classes.Morphisms.html

```
Instance sig2h_morphism : Proper (meq ⇒ meq ⇒ iff) (sig2h n).

(* forall  (n :  nat) (Θ Λ : Multiset),
   Θ =mul= Λ →
      forall  Γ Δ : Multiset,
         Γ =mul= Δ →
            n ⊢ Θ : Γ ↔  n ⊢ Λ : Δ *)
(** A prova é justamente a aplicação do Lema 3.2.1 (sig2h_der_compat) *)
```

Em Coq, provamos morfismos semelhantes para todos os sistema de LL descritos neste trabalho. Alguns fatos notáveis quanto à altura é que ela preserva *weakening* e *contraction* no contexto clássico.

Lema 2.2.7 (**Altura preserva enfraquecimento**). *Dados* Θ, Δ *e* Γ *multiconjuntos de fórmulas, se o sequente* $_n \vdash \Theta : \Gamma$ *é demonstrável, então* $_n \vdash \Theta, \Delta : \Gamma$ *é demonstrável.*

```
Lemma height_preserving_weakning_sig2h: forall n Θ Δ Γ,
    n ⊢ Θ : Γ →  n ⊢ Θ ∪ Δ : Γ.
```

Lema 2.2.8 (**Altura preserva contração**). *Dados* Θ *e* Γ *multiconjuntos de fórmulas, o sequente* $_n \vdash \Theta, F, F : \Gamma$ *é demonstrável se, e somente se, o sequente* $_n \vdash \Theta, F : \Gamma$ *é demonstrável.*

```
Lemma height_preserving_contraction_sig2h : forall n Θ Γ F,
    n ⊢ {{F}} ∪ {{F}} ∪ Θ : Γ ↔  n ⊢ {{F}} ∪ Θ : Γ.
```

Com o teorema de preservação da contração, concluímos que o contexto clássico atua como um conjunto de fórmulas e o linear como um multiconjunto delas. Os lemas **2.2.7** e **2.2.8** foram provados em Coq para todos os sistemas de LL tratados neste trabalho.

Em Coq, provamos que os sistemas com e sem a definição de altura são equivalentes.

```
Theorem sig2_iff_sig2h : forall Θ Γ, ⊢ Θ : Γ ↔  ∃ m, m  ⊢ Θ : Γ.
Proof.
    – (* sig2 →  sig2h *)
    (** A prova segue por indução sobre a hipótese ⊢ Θ : Γ *)
    (** Todos os casos gerados são triviais, pois as regras são idênticas *)
    – (* sig2n →  sig2 *)
    (** A prova segue por indução sobre a altura m *)
    (** Todos os casos gerados são triviais, pois as regras são idênticas *)
Qed.
```

No sentido comum, dizemos que uma conclusão é evidente toda vez que as premissas são. No entanto, algumas regras possuem um comportamento especial, chamamo-las de regras invertíveis. Dizemos que uma regra é invertível quando as premissas são deriváveis

sempre que a conclusão é derivável. Tais regras podem ser aplicadas com segurança sempre que possível sem perder provabilidade.

Definição 2.2.9. *Dizemos que uma regra é invertível se as premissas podem ser provadas sse a conclusão pode ser provada.*

$$\frac{P_1, P_2, \ldots}{C}\ \textit{[regra invertível]} \qquad\qquad \frac{C}{P_1, P_2, \ldots}\ \textit{[regra invertível]}$$

Teorema 2.2.10. *[**Invertibilidade**] As regras $\parr$, $\bot$, ? e & são invertíveis em Σ_{2h}.*

```
Theorem invertibility_par_sig2h : forall n Θ Γ Δ F G,
    Γ =mul= {{F ⅋ G}} ∪ Δ → n ⊢ Θ : Γ → ∃ m, m ⊢ Θ : {{F}} ∪ {{G}} ∪ Δ.
Proof. (** A prova segue por indução na altura da derivação "n" *) Qed.

Theorem invertibility_bot_sig2h : forall n Θ Γ Δ,
    Γ =mul= {{⊥}} ∪ Δ → n ⊢ Θ : Γ → ∃ m, m ⊢ Θ : Δ.
Proof. (** A prova segue por indução na altura da derivação "n" *) Qed.

Theorem invertibility_quest_sig2h : forall n Θ Γ Δ F,
    Γ =mul= {{? F}} ∪ Δ → n ⊢ Θ : Γ → ∃ m, m ⊢ {{F}} ∪ Θ : Δ.
Proof. (** A prova segue por indução na altura da derivação "n" *) Qed.

Theorem invertibility_with_sig2h : forall n Θ Γ Δ F G,
    Γ =mul= {{F & G}} ∪ Δ →
        n ⊢ Θ : Γ → ∃ m1 m2, m1 ⊢ Θ : {{F}} ∪ Δ ∧ m2 ⊢ Θ : {{G}} ∪ Δ.
Proof. (** A prova segue por indução na altura da derivação "n" *) Qed.
```

O teorema seguinte permite simplificar o término de uma prova em cálculo de sequentes. A regra *init* (axioma inicial) só permite concluir a prova se contamos somente com um átomo e sua versão negada no contexto linear. Porém, para qualquer fórmula e sua versão negada nas mesmas condições também concluiríamos a prova.

Teorema 2.2.11. *[**Generalização do axioma inicial**] Seja F uma fórmula, então o sequente $\vdash \Theta : F, F^{\perp}$ é demonstrável em Σ_{2h}.*

```
Theorem generalize_init_sig2h : forall F Γ,
    Γ =mul= {{F}} ∪ {{F^⊥}} → ∃ m, m ⊢ Θ : Γ.
Proof.
    (** A prova segue por indução sobre a fórmula F *)
    (** Demandamos o uso do lema 3.2.7 para concluir a prova *)
Qed.
```

As regras admissíveis de uma lógica são as regras sob as quais o conjunto de teoremas da lógica é fechado (IEMHOFF; METCALFE, 2009). Ou seja, o conjunto de

teoremas do sistema não se modifica quando uma regra admissível é incorporada. Se um sequente é derivável usando essa regra, então há uma derivação desse sequente sem ela.

As regras *copy*, ? e ! são regras de um único passo. Por exemplo, em *copy* copiamos uma única expressão em LL do contexto clássico para o linear. Portanto, se desejarmos copiar todo o contexto clássico, ou um subconjunto dele, para o linear teríamos que fazer um a um. Assim sendo, podemos generalizar tais regras.

Teorema 2.2.12. *[**Generalização da regra copy**] Sejam* Θ, Λ *e* Γ *multiconjuntos de fórmulas, então o sistema* Σ_2 *admite a seguinte regra:*

$$\frac{\vdash \Theta, \Lambda \ : \ \Gamma, \Lambda}{\vdash \Theta, \Lambda \ : \ \Gamma}\ [copy_G]$$

```
Theorem generalize_copy_sig2h: forall Θ Λ Γ n,
    n ⊢ Θ ∪ Λ ; Γ ∪ Λ → ∃ m, m ⊢ Θ ∪ Λ ; Γ.
Proof.
    (** A prova segue por indução sobre o multiconjunto de fórmulas Λ *)
    – (* caso base *)
    (** Trivial *)
    – (* caso indutivo *)
    (** Demandamos da hipótese de indução, das regra ? e copy e do uso do teorema 3.2.9,
    com relação a "?", para conclusão *)
Qed.
```

Teorema 2.2.13. *[**Generalização da regra quest**] Sejam* Θ, Γ *e* Δ *multiconjuntos de fórmulas, então o sistema* Σ_2 *admite a seguinte regra:*

$$\frac{\vdash \Theta, \Delta \ : \ \Gamma}{\vdash \Theta \ : \ \Gamma, ?\Delta}\ [?_G]$$

```
Theorem generalize_quest_sig2h : forall Θ Γ Δ n,
    n ⊢ Δ ∪ Θ ; Γ → ∃ m, m ⊢ Θ ; Γ ∪ (map ? Δ).
Proof.
    (** A prova segue por indução sobre o multiconjunto de fórmulas Λ *)
    – (* caso base *)
    (** Trivial *)
    – (* caso indutivo *)
    (** Demandamos da hipótese de indução e da regra ? para conclusão *)
Qed.
```

Teorema 2.2.14. *[**Generalização da regra bang**] Sejam* Θ *e* Δ *multiconjuntos de fórmulas, então o sistema* Σ_2 *admite a seguinte regra:*

$$\frac{\vdash \Theta \ : \ F, ?\Delta}{\vdash \Theta \ : \ !F, ?\Delta}\,[!_G]$$

```
Theorem generalize_bang_sig2h : forall Θ Δ F n,
    n ⊢ Θ : {{F}} ∪ (? Δ) → ∃ m, m ⊢ Θ : {{! F}} ∪ (map ? Δ).
Proof.
    (** A prova segue por indução sobre o multiconjunto de fórmulas Δ *)
    – (* caso base *)
    (** Trivial *)
    – (* caso indutivo *)
    (** Demandamos da hipótese de indução, ds regra ? e do uso do teorema 3.2.9,
    com relação a "?", para conclusão *)
Qed.
```

Observe a seguinte regra,

$$\frac{\begin{array}{cc}\pi_1 & \pi_2 \\ \vdots & \vdots \\ \vdash \Theta \ : \ \Gamma, \ F & \vdash \Theta \ : \ \Delta, \ F^{\perp}\end{array}}{\vdash \Theta \ : \ \Gamma, \ \Delta}\,[cut]$$

chamamo-la de regra do corte e, neste trabalho, denominamos a soma $\pi_1 + \pi_2$ de altura do corte e F de fórmula do corte. Muitas derivações e provas de teoremas podem ser simplificadas com o uso dessa regra.

Exemplo 2.2.15. *Considere a invertibilidade da regra* $\perp$.

$$\frac{\begin{array}{c}\pi \\ \vdots \\ \vdash \Theta \ : \ \Delta, \perp\end{array}}{\vdash \Theta \ : \ \Delta}\,[\perp\text{-}invert]$$

Se temos a prova do sequente $\vdash \Theta \ : \ \Delta, \perp$, *então conseguimos concluir* $\vdash \Theta \ : \ \Delta$. *A prova pode seguir apenas com a aplicação da regra do corte com* $F = 1$ *e* $\Gamma = \{\}$.

$$\frac{\dfrac{}{\vdash \Theta \ : \ 1}\,[1] \qquad \begin{array}{c}\pi \\ \vdots \\ \vdash \Theta \ : \ \Delta, \ 1^{\perp} \ (1^{\perp} = \perp)\end{array}}{\vdash \Theta \ : \ \Delta}\,[cut]$$

Semelhantemente, a verificação das outras invertibilidades prossegue facilmente em vez de indução na altura da derivação. Assim, podemos incorporar a regra do corte em Σ_{2h} para obtermos um sistema de regras mais poderoso denominado $\Sigma_{2hc} \equiv \Sigma_{2h} + cut$, no

qual um sequente, de altura n com contextos clássico Θ e linear Γ e com a aplicação de c vezes a regra do corte, detém a notação $n \vdash_c \Theta : \Gamma$.

```
Theorem invertibility_bot_sig2hc : forall n Θ Γ Δ,
    Γ =mul= {{⊥}} ∪ Δ → n ⊢c Θ : Γ → ∃ m, m ⊢c Θ : Δ.
Proof.
    (** Aplicamos a regra do corte com F = 1 e Δ = Δ *)
    – (* caso 1 *)
    (** Aplicamos a regra 1 *)
    – (* caso ⊥ *)
    (** Usamos a hipótese n ⊢c Θ : Γ *)
Qed.

Theorem invertibility_par_sig2hc : forall n F G Θ Γ Δ,
    Γ =mul= {{F ⅋ G}} ∪ Δ → n ⊢c Θ : Γ → ∃ m, m ⊢c Θ : {{F}} ∪ {{G}} ∪ Δ.
Proof.
    (** Aplicamos a regra do corte com F = F ⅋ G e Δ = {{F}} ∪ {{G}} *)
    – (* caso F ⅋ G *)
    (** Usamos a hipótese n ⊢c Θ : Γ *)
    – (* caso F^⊥ ⊗ G^⊥ *)
    (** Usamos a regra ⊗ e o teorema 3.2.10 adaptado para sig2hc *)
Qed.

Theorem invertibility_quest_sig2hc : forall n Θ Γ Δ F,
    Γ =mul= {{? F}} ∪ Δ → n ⊢c Θ : Γ → ∃ m, m ⊢c {{F}} ∪ Θ : Δ.
Proof.
    (** Aplicamos a regra do corte com F = ! F^⊥ e Δ = Δ *)
    – (* caso ! F^⊥ *)
    (** Aplicamos a regra !, em seguida a regra copy e o teorema 3.2.10 adaptado para sig2hc *)
    – (* caso ? F *)
    (** Usamos o lema 3.2.7 adaptado para sig2hc e a hipótese n ⊢c Θ : Γ *)
Qed.

Theorem invertibility_with_sig2hc : forall n Θ Γ Δ F G,
    Γ =mul= {{F & G}} ∪ Δ →
        n ⊢c Θ : Γ → ∃ m1 m2, m1 ⊢c Θ : {{F}} ∪ Δ ∧ m2 ⊢c Θ : {{G}} ∪ Δ.
Proof.
    – (* caso m1 ⊢c Θ : {{F}} *)
    (** Aplicamos a regra do corte com F = F with G e Δ = {{F}} *)
        + (* caso F with G *)
        (** Usamos a hipótese n ⊢c Θ : Γ *)
        + (* caso F^⊥ ⊕ G^⊥ *)
```

```
        (** Usamos a regra ⊕1 e o teorema 3.2.10 adaptado para sig2hc *)
    - (* caso m2 ⊢c Θ : {{G}} *)
    (** Aplicamos a regra do corte com F = F with G e Δ = {{G}} *)
        + (* caso F with G *)
        (** Usamos a hipótese n ⊢c Θ : Γ *)
        + (* caso F^⊥ ⊕ G^⊥ *)
        (** Usamos a regra ⊕2 e o teorema 3.2.10 adaptado para sig2hc *)
Qed.
```

A regra do corte introduz uma nova e arbitrária proposição F, que não necessariamente está relacionado com as fórmulas da conclusão. Portanto, a regra do corte é muito poderosa, mas, do ponto de vista de *proof search*, implica que devemos encontrar F tal que a prova termina. Claramente, isso incorpora uma grande quantidade de busca de prova não-determinística. Se para qualquer árvore finita de derivação de um sequente que usufrui da regra o corte podemos obter outra árvore, também finita, sem instâncias dessa regra, então podemos contornar tal problema, em outras palavras, podemos eliminar todos os cortes.

2.2.3 Eliminação do Corte

A prova deste teorema em Coq é o principal resultado deste trabalho. O teorema da eliminação do corte (ou *cut-elimination*) é de suma importância, visto que carrega implicações como a consistência, em geral, e a propriedade subfórmula.

Teorema 2.2.16. *[**Eliminação do corte**] O sequente* $\vdash \Theta : \Gamma$ *é demonstrável se, e somente se, é demonstrável sem a aplicação da regra cut.*

O teorema de eliminação do corte agora nos diz que nunca precisamos usar essa regra. Todas as restantes regras têm a propriedade de que as premissas contêm apenas instâncias de proposições na conclusão, ou partes dela. Esta propriedade é chamada de propriedade de subfórmula.

Definição 2.2.17 (**Peso de uma fórmula**)**.** *O peso w de uma expressão em LL é definido como*

$$w(F) = \begin{cases} 0 & , \text{ caso } F \text{ seja uma unidade ou um literal} \\ w(A) + 1 & , \text{ caso } F \text{ seja da forma } \circ A \\ w(A) + w(B) + 1 & , \text{ caso } F \text{ seja da forma } A \circ B. \end{cases}$$

A prova do Teorema 2.2.16 prossegue tradicionalmente por indução na ordem lexicográfica do par ordenado (peso da fórmula do corte, altura do corte). Evidentemente, essa indução gera múltiplos casos. De forma geral, na prova de eliminação do corte, deve-se

considerar a última regra aplicada nas duas premissas imediatamente depois de aplicar a regra do corte, o que gera um número considerável de situações que por razões de similaridade não são consideradas em uma prova informal.

Previamente, sabe-se que na prova do Teorema 2.2.16 deve-se considerar mais uma regra, denominada neste trabalho de *ccut*.

$$\dfrac{\begin{array}{cc}\pi_1 & \pi_2 \\ \vdots & \vdots \\ \vdash \Theta \ : \ \Gamma, \ !F & \vdash \Theta, \ F^{\perp} \ : \ \Delta\end{array}}{\vdash \Theta \ : \ \Gamma, \ \Delta}\,[ccut]$$

Desse modo, obtemos um novo sistema de regras $\Sigma_{2hcc} \equiv \Sigma_{2hc} + ccut \equiv \Sigma_{2h} + cut + ccut$. É fácil ver que Σ_{2hcc} é equivalente a Σ_{2hc}, pois, com exceção da regra *ccut*, as regras são idênticas.

```
Theorem sig2hcc_iff_sig2hc: forall Γ Δ Θ n, n ⊢cc Θ : Γ ↔ ∃ m, m ⊢c Θ : Γ.
Proof.
  – (* sig2hcc → sig2hc *)
    (** A prova segue por indução na estrutura de ⊢cc Θ : Γ *)
    (** Seguimos aplicando a regra correspondente entre os sistemas *)
    (** No caso em que é ultilizada a regra ccut: *)
    (** Aplicamos um cut com F = ! F e, em seguida,
            aplicamos a regra ? no sequente apropriado *)
  – (* sig2hc → sig2hcc *)
    (** A prova segue por indução na estrutura de ⊢c Θ : Γ *)
    (** Seguimos aplicando a regra correspondente entre os sistemas *)
Qed.
```

Com efeito, a regra *ccut* pode ser verificada com as aplicações de *cut* e ?.

$$\dfrac{\vdash \Theta \ : \ \Gamma, \ !F \qquad \dfrac{\vdash \Theta, \ F^{\perp} \ : \ \Delta}{\vdash \Theta \ : \ \Delta, \ (!F)^{\perp}}\,[?]}{\vdash \Theta \ : \ \Gamma, \ \Delta}\,[cut]$$

Para lidarmos com a eliminação do corte em Coq, precisamos incorporar, nas regras, o número c de vezes que empregamos a regra do corte em uma derivação. Por simplicidade, denominamos c como número de cortes. As folhas da árvore em uma derivação possuem $c = 0$, ou seja, o axioma inicial e as regras 1 e $\top$. O corte é a única regra que incrementa c, $c = c + 1$. Dizemos que um sequente é *cut-free* se $c = 0$.

Com o propósito de evitar imprecisão, construímos um novo sistema de regras intitulado Σ_3 (ver Figura 8), no qual um sequente, de altura n, com um número c de cortes (*cut* ou *ccut*) e com contextos clássico Θ e linear Γ, detém a notação ${}_n \Rightarrow_c \Theta : \ \Gamma$. Provamos, em Coq, que os sistemas Σ_{2hcc} e Σ_3 são equivalentes.

```
Theorem sig2hcc_iff_sig3 : forall B L n, n ⊢cc Θ : Γ ↔ ∃ c, n ⇒ c : Θ : Γ.
Proof.
  – (* sig2hcc → sig3 *)
    (** A prova segue por indução na altura da derivação *)
  – (* sig3 → sig2hcc *)
    (** A prova segue por indução na altura da derivação *)
Qed.
```

Em Coq, conforme a Figura 8, codificamos o sistema Σ_3 desta maneira:

```
Reserved Notation " n '⇒' m ';' B ';' L" (at level 80).
Inductive sig3 : nat → nat → Multiset → Multiset → Prop :=

  | sig3_init : forall B L A, L =mul= ({{A}} ∪ {{A^⊥}}) → 0 ⇒ 0 ; B ; L
  | sig3_one : forall B L, L =mul= {{1}} → 0 ⇒ 0 ; B ; L
  | sig3_top : forall B L M, L =mul= {{⊤}} ∪ M → 0 ⇒ 0 ; B ; L

  | sig3_bot : forall B L M n c, L =mul= {{⊥}} ∪ M → n ⇒ c ; B ; M → S n ⇒ c ; B ; L
  | sig3_par : forall B L M F G n c,
    L =mul= {{F ⅋ G}} U M → n ⇒ c ; B ; {{F}} ∪ {{G}} ∪ M → S n ⇒ c ;B ; L
  | sig3_tensor : forall B L M N F G n m c1 c2,
    L =mul= {{F ⊗ G}} ∪ (M ∪ N) →
      m ⇒ c1 ; B ; {{F}} ∪ M → n ⇒ c2 ; B ; {{G}} ∪ N → S (max n m) ⇒ c1+c2 ;B ; L
  | sig3_plus1 : forall B L M F G n c,
    L =mul= {{F ⊕ G}} ∪ M → n ⇒ c ; B ; {{F}} ∪ M → S n ⇒ c ; B ; L
  | sig3_plus2 : forall B L M F G n c,
    L =mul= {{F ⊕ G}} ∪ M → n ⇒ c ; B ; {{G}} ∪ M → S n ⇒ c ; B ; L
  | sig3_with : forall B L M F G n m c1 c2,
    L =mul= {{F with G}} ∪ M →
      m ⇒ c1 ; B ; {{F}} ∪ M → n ⇒ c2 ; B ; {{G}} ∪ M → S (max n m) ⇒ c1 + c2; B ; L
  | sig3_copy : forall B L F n c, F ∈ B → n ⇒ c; B ; {{F}} ∪ L → S n ⇒ c ; B ; L
  | sig3_quest : forall B L M F n c,
    L =mul= {{? F}} ∪ M → n ⇒ c; {{F}} ∪ B ; M → S n ⇒ c; B ; L
  | sig3_bang : forall B F L n c,
    L =mul= {{! F}} → n ⇒ c; B ; {{F}} → S n ⇒ c ; B ; L
  | sig3_CUT : forall B L n c w h, sig3_cut_general w h n c B L → S n ⇒ S c ; B ; L
  with
sig3_cut_general : nat → nat → nat → nat → Multiset → Multiset → Prop :=
    | sig3_cut : forall B L M N F m n c1 c2 w h,
        w = lexp_weight F →
        h = m + n →
        L =mul= (M ∪ N) →
        m ⇒ c1 ; B ; {{F}} ∪ M →
```

Figura 8 – Regras de derivação para LL com altura e número de cortes.

Sejam A um literal, F e G fórmulas e Γ, Θ e Δ multiconjuntos de fórmulas:

$$\frac{}{{}_0 \Rightarrow_0 \Theta \ : \ A, A^{\perp}}\,[init] \qquad \frac{}{{}_0 \Rightarrow_0 \Theta \ : \ \Gamma, \top}\,[\top] \qquad \frac{}{{}_0 \Rightarrow_0 \Theta \ : 1}\,[1]$$

$$\frac{{}_n \Rightarrow_c \Theta \ : \ \Gamma, F, G}{{}_{n+1} \Rightarrow_c \Theta \ : \ \Gamma, F \parr G}\,[\parr] \qquad \frac{{}_n \Rightarrow_c \Theta \ : \ \Gamma, F_i}{{}_{n+1} \Rightarrow_c \Theta \ : \ \Gamma, F_1 \oplus F_2}\,[\oplus_i]$$

$$\frac{{}_{n_1} \Rightarrow_{c_1} \Theta \ : \ \Gamma, F \qquad {}_{n_2} \Rightarrow_{c_2} \Theta \ : \ \Delta, G}{{}_{max(n_1,\ n_2)+1} \Rightarrow_{c_1+c_2} \Theta \ : \ \Gamma, F \otimes G, \Delta}\,[\otimes] \qquad \frac{{}_n \Rightarrow_c \Theta \ : \ \Gamma}{{}_{n+1} \Rightarrow_c \Theta \ : \ \Gamma, \bot}\,[\bot]$$

$$\frac{{}_{n_1} \Rightarrow_{c_1} \Theta \ : \ \Gamma, F \qquad {}_{n_2} \Rightarrow_{c_2} \Theta \ : \ \Gamma, G}{{}_{max(n_1,\ n_2)+1} \Rightarrow_{c_1+c_2} \Theta \ : \ \Gamma, F \,\&\, G}\,[\&] \qquad \frac{{}_n \Rightarrow_c \Theta, F \ : \ \Gamma}{{}_{n+1} \Rightarrow_c \Theta \ : \ \Gamma, ?F}\,[?]$$

$$\frac{{}_{n_1} \Rightarrow_{c_1} \Theta \ : \ \Gamma, F \qquad {}_{n_2} \Rightarrow_{c_2} \Theta \ : \ \Delta, F^{\perp}}{{}_{max(n_1,\ n_2)+1} \Rightarrow_{(c_1+c_2)+1} \Theta \ : \ \Gamma, \Delta}\,[cut] \qquad \frac{{}_n \Rightarrow_c \Theta, F \ : \ \Gamma, F}{{}_{n+1} \Rightarrow_c \Theta, F \ : \ \Gamma}\,[copy]$$

$$\frac{{}_{n_1} \Rightarrow_{c_1} \Theta \ : \ \Gamma, !F \qquad {}_{n_2} \Rightarrow_{c_2} \Theta, F^{\perp} \ : \ \Delta}{{}_{max(n_1,\ n_2)+1} \Rightarrow_{(c_1+c_2)+1} \Theta \ : \ \Gamma, \Delta}\,[ccut] \qquad \frac{{}_n \Rightarrow_c \Theta \ : \ F}{{}_{n+1} \Rightarrow_c \Theta \ : \ !F}\,[!]$$

```
        n ⇒  c2 ; B ; {{F⊥}} ∪ N →
        sig3_cut_general w h (max n m) (c1 + c2) B L
    | sig3_ccut : forall B L M N F m n c1 c2 w h,
        w = lexp_weight (! F) →
        h = m + n →
        L =mul= (M ∪ N) →
        m ⇒  c1 ; B ; {{! F}} ∪ M →
        n ⇒  c2 ; {{F⊥}} ∪ B ; N →
        sig3_cut_general w h (max n m) (c1 + c2) B L
where "n ⇒  m ; B ; L" := (sig3 n m B L).

Notation " n '⇝' m ';' w ';' h ';' B ';' L"
    := (sig3_cut_general w h n m B L) (at level 80).
```

Quando fala-se em eliminar o corte, tem-se em mente que precisamos excluir todas as instâncias das regras *cut* e *ccut* em uma derivação. Por essa razão, nessa definição do sistema Σ_3 a regra *sig3_CUT* corresponde às regras *cut* e *ccut*. Isso nos permite realizar indução mutual, pois, como veremos adiante, para eliminar todas as instâncias de *cut* necessitamos da regra *ccut* e vice-versa. Observe que já incluímos a definição de altura (h) e fórmula (w) do corte nas regras *cut* e *ccut* para facilitar a indução no par (h, w).

Diz-se que a fórmula do corte F é principal se, após a aplicação do corte, qualquer regra aplicada na premissa que contém F modifique o peso de F, de outro modo, diminui-se a sua complexidade.

Considere o seguinte lema em Coq,

Lema 2.2.18. *Se o sequente* $_n \Rightarrow_1 \Theta : \Gamma$ *é demonstrável, então existe* m *natural tal que* $_m \Rightarrow_0 \Theta : \Gamma$ *é demonstrável. Significa dizer que se temos uma derivação com exatamente um corte, então existe outra derivação do mesmo sequente sem o uso da regra CUT.*

```
Lemma aux_cut_elimination : forall n Θ Γ, n ⇒ 1 ; Θ ; Γ → ∃ m, m ⇒ 0 ; Θ ; Γ.
Proof.
(* A prova prossegue por indução na altura "n" *)
    - (* caso base *)
    (**
    Trivial, pois não existe sequente com altura zero que possua algum corte.
    Para n = 0, temos 0 ⇒ 1 ; Θ ; Γ → ∃ m, m ⇒ 0 ; Θ ; Γ.
    A hipótese 0 ⇒ 1 ; Θ ; Γ é claramente falsa,
    pois não existe regra que seja aplicada em um sequente de altura zero que possua cortes.
    Logo, a afirmação é verdadeira. *)
    - (* caso indutivo *)
    (** Temos que provar que (n + 1) ⇒ 1 ; Θ ; Γ → ∃ m, m ⇒ 0 ; Θ ; Γ
    A indução nos gerou a seguinte hipótese (HI):

        forall m, m ≤ n → forall Γ Θ, m ⇒ 1 ; Θ ; Γ → ∃ x, x ⇒ 0 ; Θ ; Γ.

    Desse modo, devido ao número de cortes, teremos dez casos
    conforme a última regra aplicada (⊥, ⅋, ⊗, ⊕1, ⊕2, &, copy, ?, ! e cut).
    A prova dos nove primeiros casos segue o mesmo padrão utilizando a hipótese indutiva.
    Por exemplo, suponha que a última regra aplicada tenha sido ⊥. *)
        + (* caso ⊥ *)
        (** Hyp1: Γ =mul= {{⊥}} ∪ Δ.
        Hyp2: n ⇒ 1 ; Θ ; Δ.

        Por HI e Hyp2, temos que x ⇒ 0 ; Θ ; Δ para algum x.
        Lembrando que precisamos demonstrar que ∃ m, m ⇒ 0 ; Θ ; Γ.
        O m que estamos procurando é x + 1.
        Logo, devido a Hyp1 e aplicando a regra ⊥ concluímos a prova. *)
(*  O caso interessante é quando a derradeira regra aplicada é o corte,
    pois não terminamos a prova com HI *)
        + (* caso CUT *)
(* Temos que provar o seguinte:
   forall n Θ Γ w h, n ⇝ 0 : w : h : Θ : Γ → ∃ m, m ⇒ 0 : Θ : Γ.
   O procedimento, como já discutido, é realizar indução no par (h, w),
   gerando quatro casos, com dois subcasos cada (cut ou ccut).
*)
    -- (* caso (0, 0) *)
```

```
        -- (* subcaso cut *)
        (** Como o peso da fórmula do corte é 0 (zero),
        teremos 6 (seis) subcasos (contando com os simétricos).
        Como a altura de todos é 0 (zero), minimizamos para 2 (dois),
        os quais são facilmenete resolvíveis. *)
        -- (* subcaso ccut *)
        (** Trivial, pois a fórmula do corte em ccut possui peso diferente de 0 (zero) *)
    -- (* caso (0, S w) *)
        -- (* subcaso cut *)
        (** Como o peso da fórmula do corte é diferente de 0 (zero),
        teremos 6 (seis) subcasos (contando com os simétricos).
        Todos são triviais, pois possuem altura 0 (zero). *)
        -- (* subcaso ccut *)
        (** Como o peso da fórmula do corte é diferente de 0 (zero),
        teremos 6 (seis) subcasos (contando com os simétricos).
        Todos são triviais, pois possuem altura 0 (zero). *)
    -- (* caso (S h, 0) *)
        -- (* subcaso cut *)
        (** Como o peso da fórmula do corte é diferente de 0 (zero),
        teremos 6 (seis) subcasos que são reduzidos para 3 (três) devido à simetria.
        Para esses, agrupamos cerca de 240 (duzenos e quarenta) outros subcasos.
        As táticas implementadas resolvem a grande maioria deles
        e o que resta é simples de se resolver *)
        -- (* subcaso ccut *)
        (** Trivial, pois a fórmula do corte em ccut possui peso diferente de 0 (zero) *)
    -- (* caso (S h, S w) *)
        -- (* subcaso cut *)
        (** Como o peso da fórmula do corte é diferente de 0 (zero),
        teremos 6 (seis) subcasos que são reduzidos para 3 (três) devido à simetria.
        Para esses, agrupamos cerca de 240 (duzenos e quarenta) outros subcasos.
        As táticas implementadas resolvem a grande maioria deles.
        A exceção fica a cargo dos casos em que a fórmula do corte é principal.
        Eles são resolvidos como na literatura. *)
        -- (* subcaso ccut *)
        (** Como a fórmula do corte em ccut possui peso diferente de 0 (zero),
        teremos 12 (doze) subcasos. Para esses, Coq nos gera cerca de 1400
        (mil e quatroscentos) outros subcasos. As táticas implementadas resolvem a grande
        maioria deles. A exceção fica a cargo do caso
        em que a fórmula do corte é principal.*)
Qed.
```

A ideia intuitiva da prova de eliminação do corte é, sempre que possível, substituir-

mos uma derivação por outra com o corte de altura menor. Quando não for possível fazer tal substituição, podemos diminuir a complexidade da fórmula do corte. Esse procedimento acarreta a eliminação das instâncias da regra do corte.

Observe como conseguimos diminuir a altura do corte no caso a seguir (Figura 9):

Figura 9 – Caso comutativo da regra $\parr$.

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F, A, B, \Gamma\end{array}}{\vdash \Theta \ : \ F, A \parr B, \Gamma}\,[\parr] \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Theta \ : \ F^{\perp}, \Delta\end{array}}{\vdash \Theta \ : \ A \parr B, \Gamma.\Delta}\,[cut : hc = (\pi_1 + 1) + \pi_2]$$

$$\Downarrow\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F, A, B, \Gamma\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Theta \ : \ F^{\perp}, \Delta\end{array}}{\vdash \Theta \ : \ A, B, \Gamma, \Delta}\,[cut : hc = \pi_1 + \pi_2]}{\vdash \Theta \ : \ A \parr B, \Gamma.\Delta}\,[\parr]$$

Isso sempre acontece quando a fórmula do corte não é principal. Denominamos esses casos de *comutativos*, pois a aplicação da regra do corte pode comutar com a aplicação da regra do conectivo presente na raiz da árvore de derivação. Inclusive, na seção **3.1**, mostramos como geramos táticas para essas situações. Portanto, os casos mais embaraçosos, conhecidos como *casos-chave*, são quando a fórmula do corte é principal.

As figuras 10 e 11 mostram as soluções dos dois casos-chave considerados mais importantes neste trabalho: quando a fórmula do corte $!F$ é principal em *cut* e *ccut*, respectivamente. Esses casos nos mostram a importância da indução mutual uma vez que há uma dependência recíproca entre *ccut* e *cut* na eliminação do corte.

Conseguimos diminuir altura do corte com a aplicação da regra *ccut* na Figura 10.

Em *cut*, estamos reduzindo o peso da fórmula do corte ($w(F) < w(!F)$) e em *ccut* estamos diminuindo a altura do corte (hc).

A partir daqui, estamos prontos para analisar o Teorema 2.2.16 (eliminação do corte) em Coq.

```
Theorem cut_elimination : forall Θ Γ n c, n ⇒ c : Θ : Γ →  ∃ m, m ⇒ 0 : Θ : Γ.
Proof.
(* A prova prossegue por indução em "n" seguida de indução em "c". *)
    + (* caso (0, 0) *)
```

Figura 10 – Quando a fórmula do corte $!F$ é principal em *cut*.

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F\end{array}}{\vdash \Theta \ : \ !F}\,[!] \qquad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \vdash F^{\perp}, \Theta \ : \ \Gamma\end{array}}{\vdash \Theta \ : \ ?F^{\perp}, \Gamma}\,[?]}{\vdash \Theta \ : \ \Gamma}\,[cut : hc = (\pi_1 + 1) + (\pi_2 + 1)]$$

$$\Downarrow\!\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F\end{array}}{\vdash \Theta \ : \ !F}\,[!] \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash F^{\perp}, \Theta \ : \ \Gamma\end{array}}{\vdash \Theta \ : \ \Gamma}\,[ccut : hc = (\pi_1 + 1) + \pi_2]$$

Figura 11 – Quando a fórmula do corte $!F$ é principal em *ccut*.

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F\end{array}}{\vdash \Theta \ : \ !F}\,[!] \qquad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \vdash F^{\perp}, \Theta \ : \ F^{\perp}, \Gamma\end{array}}{\vdash F^{\perp}, \Theta \ : \ \Gamma}\,[copy]}{\vdash \Theta \ : \ \Gamma}\,[ccut : hc = (\pi_1 + 1) + (\pi_2 + 1)]$$

$$\Downarrow\!\Downarrow$$

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F\end{array}}{\vdash \Theta \ : \ !F}\,[!] \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash F^{\perp}, \Theta \ : \ F^{\perp}, \Gamma\end{array}}{\vdash \Theta \ : \ F^{\perp}, \Gamma}\,[ccut : hc = (\pi_1 + 1) + \pi_2]}{\vdash \Theta \ : \ \Gamma}\,[cut]$$

```
    (** Trivial, pois a conclusão é idêntica à hipótese. *)
  + (* caso (0, S c) *)
    (** Trivial, pois, como a altura é 0 (zero), não há caso com número de cortes
    diferente de 0 (zero). *)
  + (* caso (S n, 0) *)
    (** Como a altura é diferente de 0 (zero), teremos 9 (nove) subcasos.
    Os quais podem ser considerados trivais, pois o número de cortes é nulo. *)
  + (* caso (S n, S c) *)
    (** Como a altura é diferente de 0 (zero), como também a quantidade de cortes,
```

```
        teremos 10 (dez) subcasos. *)
            - (* caso de uma regra que não é CUT *)
            (** É demonstrado com a hipótese indutiva e
            a aplicação da regra correspondente. *)
            - (* caso CUT *)
                -- (* caso cut *)
            (** Por hipótese de indução, temos as provas das premissas de cut sem cortes.
            Logo, pela regra cut, temos a prova da conclusão de cut com exatamente
            1 (um) corte. Pelo lema 2.2.18 (aux_cut_elimination) temos essa mesma prova
            sem cortes. *)
                -- (* caso ccut *)
            (** Por hipótese de indução, temos as provas das premissas de ccut sem cortes.
            Logo, pela regra ccut, temos a prova da conclusão de ccut com exatamente
            1 (um) corte. Pelo lema 2.2.18 (aux_cut_elimination) temos essa mesma prova
            sem cortes. *)
Qed.
```

2.2.4 Consequências da Eliminação do Corte

O teorema de eliminação do corte implica a consistência para a lógica linear. Com consistência, queremos dizer que não existe prova da unidade linear 0 (zero) com o contexto clássico vazio, ou seja, não podemos provar $\vdash \emptyset : 0$.

Teorema 2.2.19. *[Consistência da Lógica Linear] A lógica linear, admitindo-se a regra do corte, é consistente: para quaisquer n e c, o sequente ${}_n \Rightarrow_c \cdot : 0$ não pode ser provado.*

```
Theorem consistency : forall n c, ~ n ⇒ c : { } : {{0}}.
Proof.
(* Precisamos provar que (n ⇒ c : { } : {{0}}) →  False. *)
    (** Por cut_elimination, sabe-se que ∃ m, m ⇒  0; { }; {{0}}, chamemos de H.
    Entretanto, não há regra em nosso sistema que possa ser aplicada em H. *)
Qed.
```

Outro aspecto importante que a eliminação do corte nos proporciona é a propriedade subfórmula: em uma derivação as formulas das premissas sempre são subfórmulas das conclusões. A regra do corte é a única que introduz fórmulas que não são subfórmulas da conclusão, portanto, eliminando-se os cortes a propriedade é trivialmente válida.

Finalmente, sendo o corte admissível, podemos provar que todos os sistemas são equivalentes.

Capítulo 3

Aspectos sobre a Codificação

Como discutido neste trabalho, procedemos a formalização da lógica linear no assistente de provas semi-interativo Coq versão 8.5pl3. Todos os códigos estão disponíveis publicamente no repositório <https://github.com/brunofx86/LL>.

Em síntese, formalizamos o cálculo de sequentes da LL em várias versões (*two-sided*, Σ_1, Σ_2, Σ_{2h}, Σ_{2hc}, Σ_{2hcc} e Σ_3). Demostramos o quanto eles estão relacionados e provamos teoremas, principalmente, sobre regras invertíveis e eliminação do corte. A seguir, mostramos resumidamente o conteúdo dos arquivos em Coq:

O trabalho foi dividido em duas partes: 1) A formalização das provas das generalizações e invertibilidades e 2) A formalização da prova de eliminação do corte, ambas possuem um núcleo básico de definições e demonstrações de teoremas.

a_base.v : Definições básicas e a prova do principio de indução forte para os números naturais.

b_linear.v : Definição da sintaxe da lógica linear, da negação e do peso de uma expressão e a demonstração de que a negação é involutiva (Teorema 2.2.3).

c_multiset.v : Implementação de um módulo para multiconjuntos com várias demostrações de suas propriedades.

d_derivations.v : Formalização dos sistemas de regras para derivações em lógica linear e demostrações acerca do quanto estão relacionados.

e_LLExchange.v : Demonstração da admissibilidade da regra *exchange* em todos os sistemas.

f_height_preserving.v : Demonstração de teoremas que preservam a altura, isto é, a admissibilidade de regras que quando aplicadas não modificam a altura do sequente.

Todos esses arquivos compõem o grupo basilar para a formalização da lógica linear em Coq.

h_invertibilities_sig2 : Prova das invertibilidades (Teorema 2.2.10) para o sistema Σ_{2h}.

i_generalizations_sig2 : Prova das generalizações (Teorema 2.2.11, Teorema 2.2.12, Teorema 2.2.13 e Teorema 2.2.14) para o sistema Σ_{2h}.

j_generalizations_sig2c : Prova das generalizações, axioma inicial e regra !, para o sistema Σ_{2hc}, Σ_{2hcc} e Σ_3.

k_invertibilities_sig2c : Prova das invertibilidades para o sistema Σ_{2hc}, Σ_{2hcc} e Σ_3 e generalização das regras ? e *copy*.

g_aux_tactics : Programação de táticas para automatização de provas.

h_aux_cut : Demonstração de vários lemas para eliminação do corte quando a regra aplicada é *cut*.

i_aux_ccut : Demonstração de vários lemas para eliminação do corte quando a regra aplicada é *ccut*.

j_aux_elimination : Demonstração de casos básicos para o Teorema 2.2.18 com lemas adicionais.

k_cut_elimination_base : Demonstração do Teorema 2.2.18.

l_cut_elimination : Demonstração do Teorema 2.2.16 e do Teorema 2.2.19.

O leitor pode ter ficado surpreso com a quantidade de casos que provamos para a eliminação do corte. Muitos desses casos foram gerados por Coq porque formalização faz explícita a regra *exchange* que, nos sistemas apresentados na Figura 8, é implícita. Vejamos o exemplo abaixo:

Exemplo 3.0.1. *Vamos codificar em Coq a regra* $\oplus_1$ *para o sistema* Σ_2*:*

$$\frac{\vdash \Theta \ : \ \Gamma, F}{\vdash \Theta \ : \ \Gamma, F \oplus G}\,[\oplus_1]$$

```
Inductive sig2: Multiset → Multiset → Prop :=
  | sig2_plus_1 : forall F G Γ Δ,
      Γ =mul= {{F ⊕ G}} ∪ Δ → ⊢ Θ : {{F}} ∪ Δ → ⊢ Θ : Γ
```

```
Inductive sig2: Multiset → Multiset → Prop :=
  | sig2_plus_1 : forall F G Γ, ⊢ Θ : {{F}} ∪ Γ → ⊢ Θ : {{F ⊕ G}} ∪ Γ
```

É imediato que as duas definições correspondem à regra $\oplus_1$*, porém a segunda é uma cópia fiel dessa regra. Uma codificação desse tipo para todas as regras geraria uma quantidade de casos bem menor para provar. Todavia, precisaríamos incorporar a regra exchange para os contextos clássico e linear a fim de podermos usar o sistema.*

O sistema resultante seria mais simples e objetivo se considerarmos o segundo código e adicionarmos o axioma:

```
Axiom der_exchange: forall n Θ Γ Δ Λ,
    Δ =mul= Γ → Λ =mul= Θ →
        n ⊢ Λ : Δ → n ⊢ Θ : Γ.
```

Esse axioma parece óbvio, mas duas listas l_1 e l_2 são iguais se contêm a mesma quantidade de elementos e os seus elementos são iguais nas respectivas posições e isso não ocorre com multiconjuntos. Portanto, uma solução seria ordenar o elementos de l_1 e l_2, assim teríamos um sistema de regras mais simples. A dificuldade estaria em qual ordem atribuir.

E como Coq faz para decidir se dois conjuntos são iguais? Vejamos no módulo Coq.Sets.Ensembles.

```
Definition Included (B C: Ensemble) : Prop := forall x: U, In B x → In C x.
Definition Same_set (B C: Ensemble) : Prop := Included B C ∧ Included C B.
Axiom Extensionality_Ensembles : forall A B: Ensemble, Same_set A B → A = B.
```

Podemos verificar que se dois conjuntos A e B são o mesmo conjunto, isto é, todo elemento que pertence a A também pertence a B e vice-versa, então eles são considerados iguais. Portanto, estaríamos inclinados a adicionar o axioma:

```
Axiom Extensionality_Multiset : forall A B:  Multiset, A =mul= B → A = B.
```

O axioma parece inofensivo, porém axiomas inofensivos puseram em xeque sistemas de lógica matemática (e.g., o axioma 5 de Frege[a]). Portanto, neste trabalho decidimos provar nossos teoremas sem adicionar esse axioma.

[a] *https://plato.stanford.edu/entries/frege-theorem/*

Uma estratégia usada contra a exaustão de provas formais é o uso de táticas para simplificar as provas. No arquivo **g_aux_tactics.v**, se encontram as táticas mais importantes deste trabalho.

3.1 Táticas

As táticas implementadas foram criadas à medida que os lemas/teoremas eram provados em Coq.

As táticas para os casos comutativos de eliminação do corte foram implementadas em dois passos: *aux_tacX* e *cases_tacX*. Sucintamente, elas verificam se a regra tacX foi ultilizada e se ela pode comutar.

Vamos rever o caso comutativo da regra $\Re$ da Figura 9.

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F, A, B, \Gamma\end{array}}{\vdash \Theta \ : \ F, A \parr B, \Gamma}\,[\parr] \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Theta \ : \ F^{\perp}, \Delta\end{array}}{\vdash \Theta \ : \ A \parr B, \Gamma.\Delta}\,[cut : hc = (\pi_1 + 1) + \pi_2]$$

$$\Downarrow\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Theta \ : \ F, A, B, \Gamma\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Theta \ : \ F^{\perp}, \Delta\end{array}}{\vdash \Theta \ : \ A, B, \Gamma, \Delta}\,[cut : hc = \pi_1 + \pi_2]}{\vdash \Theta \ : \ A \parr B, \Gamma.\Delta}\,[\parr]$$

Em Coq teremos de provar o seguinte:

```
1 subgoal
h : nat
Θ, Δ, L, M1, M2 : Multiset
H : forall m : nat,
    m ≤ h →
        forall (n : nat) (L Θ : Multiset),
            n ⇝ 0 : 0 : m : Θ : L → ∃ m0 : nat, m0 ⇒ 0 : Θ : L
P : L =mul= M1 ∪ M2
M : Multiset
H0 : M1 =mul= Δ
M0 : Multiset
A, B : lexp
n : nat
H5 : n ⇒ 0 : Θ : ({{A}} ∪ {{B}}) ∪ M0
Hn2 : S n ⇒ 0 : Θ  : {{0}} ∪ M2
Hn1 : 0 ⇒ 0 : Θ  : {{⊤}} ∪ Δ
H3 : h = n
H1 : M0 =mul= {{0}} ∪ Γ
H2 : M2 =mul= {{A ⅋ B}} ∪ Γ
————————————————————————————————————(1/1)
exists m : nat, m ⇒  0 : Θ : L

(* PASSO 1 *)
(* Por meio da hipótese de indução H, temos de provar que
   exists  m : nat,  m ⇒ 0 : Θ : {{A}} ∪ {{B}} ∪ Γ ∪ Δ *)

(** Usando a hipótese H1, a comutatividade e a associatividade de multiconjuntos,
```

reescrevemos H5 como
H5 : n ⇒ 0 : Θ : {{0}} ∪ (Γ ∪ ({{A}} ∪ {{B}})) (PASSO 2)
Em seguida, aplicamos a hipótese de indução H com m = n e obtemos para provar:
− m ≤ m
Trivial
− n ⇝ 0 : 0 : n : Θ : {{0}} ∪ (Γ ∪ ({{A}} ∪ {{B}}))

Como
A regra cut diz:

| *sig3_cut* : forall Θ L M N F m n c1 c2 w h,
w = lexp_weight F →
h = m + n →
L =mul= (M ∪ N) →
m ⇒ c1 ; Θ ; {{F}} ∪ M →
n ⇒ c2 ; Θ ; {{$F^{\perp}$}} ∪ N →
(max n m) ⇝ (c1 + c2) : w : h : Θ : L

Precisamos modificar o primeiro 0 (zero)
para 0 + 0, pois a regra está escrita como (c1 + c2). (PASSO 3)
Aplicamos sig3_cut e obtemos:
+ 0 = lexp_weight 0, trivial
+ h = 0 + n, exatamente H3
+ {{0}} ∪ (Γ ∪ ({{A}} ∪ {{B}})) =mul= Δ ∪ {{0}} ∪ (Γ ∪ ({{A}} ∪ {{B}}),
a tática de permutação resolve
+ 0 ⇒ 0 ; Θ ; {{⊤}} ∪ Δ, exatamente Hn1
+ n ⇒ 0 ; Θ ; {{0}} ∪ (Γ ∪ ({{A}} ∪ {{B}}), exatamente H5
∗)

(∗ Agora temos uma nova hipótese:
Hyp: exists m : nat, m ⇒ 0 : Θ : {{A}} ∪ {{B}} ∪ Γ ∪ Δ ∗)

(∗∗ destruct Hyp as [t Ht] e obtemos (PASSO 4)
t : nat
Ht: t ⇒ 0 : Θ : { {A}} ∪ {{B}} ∪ Γ ∪ Δ
(isso permite substituir a variável quantificada por uma *fresh variable*)

A regra ⅋ diz:
| *sig3_par* : forall Θ L M F G n c,
L =mul= {{F ⅋ G}} U M → n ⇒ c : Θ : {{F}} ∪ {{G}} ∪ M → S n ⇒ c : Θ : L

Logo, o "m" que estamos procurando em ∃ m : nat, m ⇒ 0 : Θ : L é S t. (PASSO 5)

```
   Assim temos de provar S t ⇒ 0 : Θ : L
   Aplicamos a regra ⅋ com M = Γ ∪ Δ e, por fim, temos de provar que:
      + L =mul= {{A ⅋ B}} U Γ ∪ Δ,
      pelas hipóteses P e H2, a tática de permutação resolve ( PASSO 7 )
       + n ⇒ c : Θ : {{A}} ∪ {{B} ∪ Γ ∪ Δ, exatamente Ht ( PASSO 6 )
 *)

(* Finalizamos a prova deste caso ! *)
```

Esse foi um caso no qual a fórmula do corte é $\top$, porém ela deve funcionar para qualquer fórmula do corte. Por isso, casos assim merecem táticas que podem automatizar o processo. Vamos compor uma tática para casos desse tipo: quando a regra ⅋ comuta. Comece analisando a tática *cases_par* e siga os passos, pois os passos nessas tácticas correspondem aos passos descritos na prova do exemplo anterior.

```
Ltac aux_par H:=
match goal with
| [
   P : ?L =mul= ?M1 ∪ ?M2,
   H1 : ?n ⇒ 0 ; ?B; ({{?F}} ∪ {{?G}}) ∪ ?M,
   Hn2 : ?n0' ⇒ 0 ; ?B; {{?b}} ∪ ?M2,
   Hn1 : ?n' ⇒ 0 ; ?B; {{?a}} ∪ ?M1,
   H0 : ?M =mul= {{?b}} ∪ ?x0,
   H4 : ?M2 =mul= {{?F ⅋ ?G}} ∪ ?x0 ⊢ _ ] ⇒
(* O símbolo ? funciona como variáveis lógicas que Coq pode instanciar
   segundo as hipóteses/objetivo da prova *)

rewrite union_comm in H1; rewrite H0 in H1; rewrite union_assoc in H1;  (** PASSO 2 *)
refine (H _ _ _ _ _ _);       (** PASSO 3 *)
[ | change (0) with (0+0); refine (sig3_cut _ _ _ Hn1 H1); auto; try resolve_rewrite];
resolve_max
end.

Ltac cases_par H :=
match goal with
| [
   P : ?L =mul= ?M1 ∪ ?M2,
   H1 : ?n ⇒ 0 ; ?B; ({{?F}} ∪ {{?G}}) ∪ ?M,
   Hn2 : ?n0' ⇒ 0 ; ?B; {{?b}} ∪ ?M2,
   Hn1 : ?n' ⇒ 0 ; ?B; {{?a}} ∪ ?M1,
   H0 : ?M =mul= {{?b}} ∪ ?x0,
   H4 : ?M2 =mul= {{?F ⅋ ?G}} ∪ ?x0 ⊢ _ ] ⇒
   (* Coq procurará os casos em que as hipóteses coincidem com essas.
```

```
      Se encontrar ele  aplicar á  o  que segue ∗)
assert (∃ m, m ⇒  0 : B : {{F}} ∪ {{G}} ∪ M1 ∪U x0) as Hyp by aux_par H;  (** PASSO 1 ∗)
destruct Hyp as [t Ht]; (** PASSO 4 ∗)
eexists;                (** PASSO 5 ∗)
  refine (sig3_par _ Ht); auto; (** PASSO 6 ∗)
  resolve_rewrite; try rewrite H4; try perm_simplify; auto (** PASSO 7 ∗)
end.
```

Casos assim, com a fórmula do corte sendo $\top$, Coq gera 80 (oitenta) subcasos. Usando essa tática, diminuímos para 64 (sessenta e quatro). Táticas similares para todos os conectivos foram criadas e resolvem a maioria dos casos.

Obervação 3.1.1. *Considere novamente a Figura 9.*

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ \vdash \Theta \ : \ F, A, B, \Gamma & \vdash \Theta \ : \ F^{\perp}, \Delta\end{array}}{\vdash \Theta \ : \ A, B, \Gamma, \Delta}\,[cut]}{\vdash \Theta \ : \ A \parr B, \Gamma, \Delta}\,[\parr]$$

Mas poderíamos ter:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ \vdash \Theta \ : \ F, A, B, \Delta & \vdash \Theta \ : \ F^{\perp}, \Gamma\end{array}}{\vdash \Theta \ : \ A, B, \Gamma, \Delta}\,[cut]}{\vdash \Theta \ : \ A \parr B, \Gamma, \Delta}\,[\parr]$$

Como a regra cut divide o contexto Γ, Δ, um deles acompanha F e o outro $F^{\perp}$. Então temos dois casos: F, Γ e $F^{\perp}, \Delta$ ou F, Δ e $F^{\perp}, \Gamma$. Coq (e qualquer prova formal) deve considerar todas as possibilidades que, no papel, a intuição diz que são similares (simétricas). As tácticas desenvolvidas ajudam a automatizar esse processo e verificar que, nossa intuição no papel, de fato é verdadeira.

Considerações

Este trabalho se propôs a formalizar a lógica linear em um assistente de provas e demonstrar metateoremas acerca dessa lógica. Trabalhamos com os conceitos de peso de uma fórmula e de altura de uma derivação acoplados na definição de sequentes.

Por ser baseado no Cálculo de Construções Indutivas, além de ser um *software* livre, o assistente de provas Coq é uma ferramenta bastante confiável, em se tratando de *theorem provers*. Ele é o resultado de cerca de 30 anos de pesquisa. Por isso, o Coq se mostrou uma boa opção para a formalização de teoremas e sistemas.

A adição da regra do corte facilita algumas provas, e.g. invertibilidade de regras como descrevemos no Teorema 2.2.10. Porém a adição de uma expressão F arbitrária provoca prejuízos em busca de prova, uma vez que é muito difícil encontrar um F apropriado. Em consequência do teorema de eliminação do corte, para qualquer derivação que use a regra do corte, há outra que dispensa tal regra.

Não precisamos usar as provas de generalizações (Teorema 2.2.11, Teorema 2.2.12, Teorema 2.2.13 e Teorema 2.2.14) e invertibilidades (Teorema 2.2.10) para provar o teorema de eliminação do corte, porém elas nortearam as táticas e outorgaram muitas horas práticas que ajudaram ainda mais na familiarização da linguagem. Além disso, todas as provas servem de apoio à construção de uma biblioteca mais robusta para LL em Coq.

Mesmo com a ajuda de táticas e lemas intermediários, escrevemos uma quantidade considerável de código e a compilação da prova consome bastante memória principal, às vezes consumia mais de 2Gb de RAM com cerca de 40 (quarenta) minutos de compilação.

É evidente a diferença entre uma prova informal e outra formal. Aquela serve de esboço para esta. Provas formais podem ser verificadas automaticamente por computador. Por outro lado, uma prova informal requer muito tempo de revisão por pares para ser verificada, e ainda assim pode conter erros.

Tivemos muitos contratempos durante o desenvolvimento do trabalho, uma vez que há várias induções e inúmeras hipóteses na prova da eliminação do corte. Além disso, durante o processo, tivemos de modificar o sistema original proposto na literatura: adicionamos a altura para podermos realizar indução na altura da derivação e provamos vários lemas, depois notamos que precisaríamos do número de cortes, daí modificamos as táticas e tais lemas, e depois a definição mutual de *cut* e *ccut*. Isso acarretou muitos

problemas em cascata. Porém, diferentemente da prova informal, o assistente de provas identifica claramente os lemas que devem ser provados novamente após as modificações.

Por fim, provamos a consistência da lógica linear como consequência do teorema da eliminação do corte. Podemos afirmar que alcançamos os objetivos deste trabalho.

Podemos apontar como possíveis trabalho futuros: Integrar com linguagens de programação, e.g. Ocaml, para criar provar semiautomáticas que podem ser verificadas em Coq e Formalizar a Lógica Linear com Subexponeciais (NIGAM; OLARTE; PIMENTEL, 2016) em Coq.

Referências

ALEXIEV, V. Applications of linear logic to computation: An overview. *Logic Journal of IGPL*, v. 2, n. 1, p. 77–107, 1994. Disponível em: <http://jigpal.oxfordjournals.org/content/2/1/77.abstract>. Citado na página 28.

ANDREOLI, J.-M. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, Oxford Univ Press, v. 2, n. 3, p. 297–347, 1992. Citado na página 33.

BERTOT, Y.; CASTÉRAN, P. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 24.

BORNAT, R. Proof and disproof in formal logic: an introduction for programmers. Oxford University Press Inc, v. 2, 2005. Citado na página 15.

BUSS, S. R. *Handbook of proof theory*. [S.l.]: Elsevier, 1998. v. 137. Citado na página 23.

CARDONE, F.; HINDLEY, J. R. History of lambda-calculus and combinatory logic. *Handbook of the History of Logic*, Elsevier, Amsterdam, to appear, v. 5, p. 723–817, 2006. Citado na página 18.

CHAUDHURI, K.; DESPEYROUX, J. A hybrid linear logic for constrained transition systems with applications to molecular biology. *CoRR*, abs/1310.4310, 2013. Disponível em: <http://arxiv.org/abs/1310.4310>. Citado na página 28.

CLARKE, E. M.; WING, J. M. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, ACM, v. 28, n. 4, p. 626–643, 1996. Citado na página 10.

COQUAND, T.; HUET, G. The calculus of constructions. *Information and Computation*, v. 76, n. 2, p. 95 – 120, 1988. ISSN 0890-5401. Disponível em: <http://www.sciencedirect.com/science/article/pii/0890540188900053>. Citado na página 22.

GIRARD, J.-Y. Linear logic: its syntax and semantics. *London Mathematical Society Lecture Note Series*, Cambridge University Press, p. 1–42, 1995. Citado na página 29.

GIRARD, J.-Y.; TAYLOR, P.; LAFONT, Y. *Proofs and Types*. New York, NY, USA: Cambridge University Press, 1989. ISBN 0-521-37181-3. Citado 2 vezes nas páginas 10 e 17.

IEMHOFF, R.; METCALFE, G. Proof theory for admissible rules. *Annals of Pure and Applied Logic*, v. 159, n. 1–2, p. 171 – 186, 2009. ISSN 0168-0072. Disponível em: <//www.sciencedirect.com/science/article/pii/S0168007208001851>. Citado na página 39.

KLEENE, J. B. R. S. C. The inconsistency of certain formal logics. *Annals of Mathematics*, Annals of Mathematics, v. 36, n. 3, p. 630–636, 1935. ISSN 0003486X. Disponível em: <http://www.jstor.org/stable/1968646>. Citado na página 19.

MACKIE, I. Lilac: A functional programming language based on linear logic. *Journal of Functional Programming*, Cambridge Univ Press, v. 4, n. 04, p. 395–433, 1994. Citado na página 28.

MATSKIN, M. Application of linear logic to web service composition. 2003. Citado na página 28.

MOOT, R.; PIAZZA, M. Linguistic applications of first order intuitionistic linear logic. *Journal of Logic, Language and Information*, v. 10, n. 2, p. 211–232, 2001. ISSN 1572-9583. Disponível em: <http://dx.doi.org/10.1023/A:1008399708659>. Citado na página 28.

NEDERPELT, R.; GEUVERS, H. *Type Theory and Formal Proof: An Introduction*. [S.l.]: Cambridge University Press, 2014. Citado 2 vezes nas páginas 20 e 22.

NEGRI, S.; PLATO, J. V.; RANTA, A. *Structural proof theory*. [S.l.]: Cambridge University Press, 2008. Citado 2 vezes nas páginas 16 e 23.

NIGAM, V.; OLARTE, C.; PIMENTEL, E. On subexponentials, focusing and modalities in concurrent systems. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, feb 2016. Citado na página 61.