Introdução à Lógica Clássica para a Ciência da Computação

Benjamín René Callejas Bedregal

Laboratório de Lógica e Inteligência Computacional - LabLIC
Departamento de Informática e Matemática Aplicada - DIMAp
Universidade Federal do Rio Grande do Norte - UFRN
bedregal@dimap.ufrn.br

Benedito Melo Acióly

Departamento de Ciências Exatas - DCE
Universidade Estadual do Sudoeste da Bahia - UESB
bma@uesb.br

NATAL, Junho de 2007

Prefácio

Apesar de ser uma área antiga de conhecimento, não há uma definição precisa nem única do que é lógica, pois essa definição dependerá dos aspectos particulares e da lógica específica que se tenha em mente. De fato, no Novo Dicionário Aurélio da Lingua Portuguesa (2a edição, 1986) são apresentadas diversas definições de lógica, indicando qual ótica está sendo considerada. Por exemplo, na visão de lógica simbólica, lógica é o "conjunto de estudos tendentes a expressar em linguagem matemática as estruturas e operações do pensamento, deduzindo-as de (um) número reduzido de axiomas, com a intenção de criar uma linguagem rigorosa, adequada ao pensamento científico tal como o concebe a tradição empírico-positivista". Na tradição clássica da lógica formal, o "Aurélio" define lógica como sendo o "estudo das formas (conceitos, juízos e raciocínios) e leis do pensamento" e na lógica material como sendo o "estudo da relação entre as formas e leis do pensamento e da verdade, i.e., estudo das operações do pensamento que conduzem a conhecimentos verdadeiros". Assim, uma vez que a lógica clássica como será vista neste texto é uma lógica formal e simbólica, tem como principais características modelar o pensamento válido através do estudo de mecanismos válidos de inferência se valendo de uma linguagem formal capaz de representar e abstrair o conhecimento.

Assim, lógica pode servir de subsídio, por exemplo, às áreas de Inteligência Artificial e Ontologia da *Web*, uma vez que a representação e processamento do conhecimento é básico nestas áreas. De fato, a lógica (clássica e não clássica) está presente de forma direta ou indireta na maioria das áreas da computação, por exemplo na Engenharia de software ela pode ser usada para especificação e verificação de software, em banco de dados para deduzir informações não presentes explicitamente no banco de dados, em hardware é usada como uma linguagem de descrição de componentes, etc. Além disso, lógica não só foi a base para o desenvolvimento dos primeiros modelos matemáticos de computação, mas foi substancial para a construção dos computadores, pois todos eles (pelo menos até o presente momento) são construídos na base de circuitos lógicos e portanto todo o que um computador faz é uma seqüência de operações lógicas. Todos estes fatos, fazem como que a lógica seja hoje em dia uma disciplina básica de qualquer curso de graduação sério de ciência e engenharia da computação.

Este livro é resultado de dez anos de ensino, pelos autores e outros colegas, da disciplina de "Lógica para a Ciência da Computação" para os cursos de Ciências e Engenharia da Computação da Universidade Federal do Rio Grande do Norte (UFRN), assim como da disciplina de "Lógica" para o Mestrado em Sistema e Computação da UFRN. Deixamos o

nosso profundo agradecimento a todos os alunos que cursaram estas disciplinas seguindo versões anteriores deste texto. Eles com suas dúvidas e comentários ajudaram certamente para o melhoramento técnico e didático deste texto.

No transcorrer desse período a disciplina também foi lecionada pelo nosso amigo e colega Regivan Hugo Nunes Santiago e a partir deste ano também pelo amigo e colega João Marcos Almeida. Eles, com suas críticas, sugestões e revisões, contribuíram imensamente na detecção e correção de alguns erros assim como na melhoria do texto como um todo, tornando-o assim mais acessível e correto. Portanto, os nossos sinceros agradecimentos a ambos.

Finalmente a todos os colegas do Departamento de Informática e Matemática Aplicada (DIMAp) da UFRN, os nossos agradecimentos pelo incentivo e amizade.

Conteúdo

Dedicatória do primeiro autor

Com todo o meu amor,
Dedico este livro
A minhas duas lindas filhas:
Berta Letícia e
Natália Tainá.

Capítulo 1

Introdução

Neste capítulo faremos uma breve descrição de que se entende por lógica e de sua importância em ciência e engenharia da computação, assim como faremos um breve histórico da lógica, desde suas origens até os dias atuais. Neste histórico apresentaremos alguns dos principais acontecimentos e descobertas que ocorreram nesta área. Finalmente daremos uma breve descrição de cada um dos capítulos subseqüentes.

1.1 Que é Lógica?

A palavra lógica é usada cotidianamente pelas pessoas, e na maioria das vezes ela está ligada à idéia de obviedade (por exemplo, algumas pessoas falam "é lógico que vou na festa") ou de estratégia (por exemplo muitas vezes se fala que a lógica agora é o time X atacar enquanto o time Y se defender). Outras vezes se fala do raciocínio lógico para se referir ao raciocínio estruturado. Estas formas de entendermos lógica não estão completamente erradas, pois lógica é a ciência que estuda como raciocinar ou como deduzir certas conclusões baseados em algumas hipóteses. Assim, as coisas óbvias em determinadas lógicas podem ser deduzidas trivialmente e as estratégias a serem executadas que levam a vitórias segundo a leitura que o estrategista tem do jogo e de seu raciocínio lógico.

O termo lógica (*logike* em grego) foi dado pelo filósofo grego Alexandre de Afrodisias no século 3 d.c., porém sistemas de organizar o pensamento já tinham sido desenvolvidos antes na própria Grécia, assim como na China e na Índia. Etimologicamente este termo vem do grego *lógos* que significava originalmente a palavra, seja escrita ou falada. Porém filósofos gregos como Heráclito lhe deram um significado mais amplo: o de razão, tanto no sentido de capacidade de racionalização individual como de um princípio cósmico de ordem e beleza. Na teologia cristã o conceito filosófico de *Logos* seria adotado na versão grega do Evangelho de João, o evangelista, quem se refere a Deus como o *logos*, isto é, a Palavra: "No princípio era a Palavra, e a Palavra estava com o Deus, e a Palavra era deus" João 1:1. De fato, algumas traduções do Evangelho em que *Logos* é o "verbo". Assim, a palavra lógica, em seus primórdios, tem um significado muito amplo, que não correspondem completamente ao entendimento do conceito de lógica nos dias de hoje.

Porém, nos tempos modernos, lógica pode ser vista como a ciência do pensamento correto, o que não significa que seja a ciência da verdade, pois mesmo que certas afirmações dentro da lógica sejam verdadeiras num determinado contexto, as mesmas afirmações podem ser falsas se aplicadas ao mundo real [Pal97]. Os filósofos afirmam que "para entender o que realmente acontece no mundo, precisamos entender o que não acontece", isto é as propriedades invariantes ou objetos que a compõem [Pal97]. Assim, podemos dizer que qualquer conjunto de afirmações que são verdadeiras ou falsas, independente do tempo e do espaço (lugar que ocupa no universo) são consideradas lógicas.

Outra forma de entender lógica é como a ciência que estuda como deduzir ou concluir certas verdades a partir de certas hipóteses ou premissas. Ou seja como uma máquina dedutiva. Mas dependendo da visão de mundo, podem existir diversos tipos de lógicas, cada qual tentando extrair noções e preceitos considerados no pensamento "lógico" do cotidiano.

As lógicas podem ser classificadas em formais e informais ou em clássicas e as não clássicas. A lógica clássica, que é a abordada neste texto, se refere à primeira lógica estudada de modo rigoroso. As principais características que possuem os sistemas lógicos clássicos são:

1. Os enunciados possuem um valor verdade, ou seja são afirmações sobre propriedades que objetos satisfazem;
2. Só há dois possíveis valores verdade: verdadeiro e falso;
3. Não há matizações entre estes valores verdades;
4. As conexões entre enunciados dão lugar a enunciados compostos, cujo valor verdade está em função, ou seja é completamente determinado, pelos valores verdade dos enunciados conectados.

Em outras palavras, a lógica clássica é apofántica, bivalente, assertórica e extensional. Em contraposição a esta lógica surgiram diversas outras que questionam alguns destes princípios.

1.2 Importância da Lógica Clássica na Ciência da Computação

A lógica clássica é a base da computação, uma vez que os computadores para realizar as computações o fazem baseados em operações lógicas (da lógica proposicional clássica) codificadas através de circuitos lógicos. Mas além disto, temos que a nível de software as linguagens de programação possuem também operações lógicas para realizar testes.

A formalização do conhecimento é indispensável para se poder automatizar as formas de raciocínio. A grande vantagem do formalismo lógico é proporcionar um método poderoso, a dedução matemática, para a geração de novos conhecimentos a partir de velhos conhecimentos. Segundo Robert Moore [Moo95] duas convicções fundamentais do inicio da civilização e que perduram até hoje são: 1) A maioria das formas superiores de condutas tidas como inteligentes requerem a representação explícita do conhecimento e 2) A lógica formal constitui a pedra fundamental da representação do conhecimento. Neste sentido, o paradigma lógico de linguagens de programação permite representar e gerar conhecimento de forma automática. Este paradigma tem como principal expoente a linguagem de programação PROLOG (vista no capítulo 8 deste livro). Linguagens de programação no paradigma lógico podem ser baseados em lógica clássica (por exemplo PROLOG) ou não (por exemplo as linguagem de programação baseados em lógica linear[Gir87]: LoLLI [Hod92, Hod95], LLP [TB01] e Lygon [HPW96]). Todas podem ser usadas para desenvolver sistemas baseados em conhecimento, isto é, sistemas que apliquem mecanismos automatizado de raciocínio para a representação e inferência de conhecimento. Assim, as linguagens de programação em lógica aplicam de modo direto os conceitos e mecanismos da lógica formal, no caso do PROLOG da lógica clássica.

A lógica clássica é usada nas mais variadas áreas e disciplinas da computação. Por exemplo,

1. Em Engenharia de software uma das questões centrais desta área é o uso de formalismos no processo de desenvolvimento de programas[MM88], sendo que o formalismo lógico por sua fácil compreensivilidade é um dos mais usados [HR99]. Dentro de engenharia de software ainda, provadores automáticos de teoremas, tais como HOL (do inglês High Order Logic), têm sido empregados para especificação e verificação de propriedades de sistemas computacionais, entre outras coisas [BGG98, HR99, Ben03]. Em Engenharia de software lógica clássica também é usada para descrever a semântica formal de linguagens de programação [DS90].

2. Em Banco de dados relacionais na descrição de consultas e no relacionamento de tabelas e em banco de dados dedutivos [Dow98, EN99].

3. Em hardware, a lógica clássica é usada como uma linguagem de descrição de componentes de hardware [Mer93, TM96] além claro das unidades lógicas de computadores.

4. Em ontologia da web, representação do conhecimento e lógica são usadas para detalhar conceitos dentro de ontologias, ou seja como uma linguagem para descrever ontologias [DOS03, Lac05].

5. Mas, a área onde talvez lógica seja mais usada é em inteligência artificial. De fato ela é o principal formalismo de representação do conhecimento e portanto é muito útil no desenvolvimento de sistemas especialistas e sistemas multi-agentes [Ric88, Fir88, RK94, RN02] assim como para descrever de forma simbólica e precisa redes neurais [CS94, BLC00]. O conhecimento extraído desde uma rede neural pode

ser formulado em regras do tipo if-then-else as quais ainda podem ser posteriormente usadas em um sistema de inferência lógica para a resolução de problemas [BLC00]. Mas lógica clássica, também tem sido usada no processamento de linguagens naturais [PM95], em algoritmos genéticos [Muh04], robótica [Min00], etc.

Mas também tem sido usada direta ou indiretamente, em outras áreas ou disciplinas da Ciência e da engenharia da Computação, tais como em criptografia [Nie02], sistemas de automação e controle [Woj99], processamento digital de imagens [GW87], Bioinformática [Sch95, Man99], Mineração de dados [Thu98, Fre02], sistemas distribuídos [BSR97, BBC97], etc.

Além disso, os conhecimentos em lógica clássica servem de base para as outras lógicas algumas das quais têm sido aplicadas em diversas áreas da computação. Assim, podemos afirmar sem medo de errar que, lógica clássica é uma materia básica que serve as vezes como fundamentação ou como ferramenta técnica para praticamente qualquer área da ciência da computação. Portanto, lógica clássica é de fundamental importância para profissionais de ciência e engenharia da computação que pretendam terem sólidos conhecimentos nas suas áreas de atuação. De fato esta disciplina faz parte das grades curriculares da grande maioria dos cursos de ciência e engenharia da computação.

1.3 Importância das Lógicas Não Clássicas na Ciência da Computação

O termo lógica não clássica é dado a qualquer lógica que não seja a lógica proposicional ou de predicado clássica. As lógicas não clássicas podem ser classificadas em antagônicas à lógica clássica ou em extensões da mesma.

Lógicas antagônicas usam a mesma linguagem da lógica clássica, porém nem todo teorema (verdade absoluta nessa lógica, isto é independente da interpretação) válidos na lógica clássica valem numa lógica antagônica. Por exemplo, em lógica intuicionista [Hey56, Dum77] a fórmula $\alpha \vee \neg\alpha$ não é um teorema. Outros exemplos de lógicas antagônicas são as lógicas multivaloradas e fuzzy ou nebulosa [Zad65, BB95, Kas96, Ngu99] as quais lidam com graus de verdade e onde algumas verdades clássicas nem sempre são verdades dessas lógicas (embora existam certas lógicas nebulosas onde as verdades coincidam com as clássicas [BC06]).

Em extensões da lógica clássica todo teorema da lógica clássica é um teorema da lógica. Porém, geralmente, complementa esta em uma das seguintes formas: enriquece a linguagem, dando a ela um maior poder de expressão ou se enriquece a teoria formal da lógica clássica com axiomas ou regras não válidas em lógica clássica. Por exemplo na lógica modal [Zem73, MP79, Cos92] são adicionados dois novos operadores $\Box$ (lido como: "em todo mundo possível" ou "é necessário que") e $\Diamond$ (lido como: "em algum mundo possível" ou "é possível que").

As lógicas não clássicas tem sido importantes em diversas áreas e aplicações de ciência da computação e inteligência artificial.

Por exemplo:

1. A lógica modal tem sido usada em [Har79] para facilitar declarar e provar propriedades de programas, onde programas são vistos como relações entre estados. Assim, cada programa induce implicitamente um operador modal permitindo expressar, numa maneira natural, propriedades de programas tais como corretude parcial.

2. A lógica temporal pode ser usada para especificação e verificação de programas concorrentes [MP79] ou para especificar circuitos de hardware [HMM83].

3. A lógica fuzzy, por tratar com graus de verdade são apropriadas para lidar com incertezas e raciocínio aproximado. Esta lógica atualmente é uma das mais usadas em intelig encia artificial, seja para desenvolvimento de sistemas especialistas, sistemas multi agentes, reconhecimento de padrões, robótica, sistemas de control inteligentes, sistemas de apoio à tomada de decisões, algoritmos genéticos, data mining, etc. [Cox05, Ros04, SB05]

Certamente, existem muitas outras lógicas não clássicas que têm aplicações ou ainda servem de fundamentação para diversas áreas ou disciplinas da computação. Porém, como este texto é de lógica clássica, nesta subseção só fizemos um levantamento superficial das mais importantes destas lógicas para a ciência da computação.

1.4 Histórico

A lógica, enquanto ciência, foi criada por Aristóteles (384-322 a.C.), sobre a base do pensamento socrático-platônico. Ele se baseou nas definições universais usadas por Sócrates (469-399 a.C.), no uso da redução ao absurdo de Zenão de Eléia (490-420 a.C.), na estrutura proposicional e negação de Parmenides(515-445 a.C.) e Platão (428-347 a.C.), e nas técnicas argumentativas encontradas no raciocínio legal e provas geométricas. Os escritos lógicos de Aristóteles, cujo conjunto foi denominado posteriormente de "órganon", constituem cinco tratados, eles são: "Categorias", "Da interpretação", "Primeiros Analíticos", "Segundos Analíticos" e "Refutações Sofísticas" [xre01, OUS01].

Aristóteles afirmava que uma proposição é um complexo envolvendo dois termos: um sujeito e um predicado, cada um dos quais é representado gramaticalmente com um substantivo. Aristóteles, em suas teorias de oposição e conversão, pesquisava as relações entre duas proposições contendo os mesmos termos. O análise da forma lógica de proposições são combinadas na maior invenção de Aristóteles em lógica, o silogismo, o qual consiste de três proposições: as duas primeiras são premissas que compartilham exatamente

um termo e elas permitem deduzir logicamente uma terceira proposição, chamada de conclusão, a qual contém os dois termos não compartilhados das premissas. Aristóteles também foi quem formulou diversas teses metalógicas, entre as mais importantes temos a lei da não contradição, o princípio do terceiro excluído e a lei da bivalência. Aristóteles se referiu a alguns princípios da lógica proposicional e para raciocínio envolvendo proposições hipotéticas. Aristóteles também criou duas teorias lógicas não formais: técnicas e estratégias para argumentos e uma teoria de falácias (raciocínios, argumentos ou afirmações aparentemente corretas, porém incorretas).

Eudemus de Rodes (350-290 a.C.) e Teofrastus de Lesbos (378-287 a.C.) , que eram alunos de Aristóteles, modificaram e desenvolveram a lógica Aristotélica em diversas maneiras. A seguinte maior inovação em lógica é devida às escolas Megariana e Estóica. Eles desenvolveram uma forma alternativa ao silogismo e elaboraram uma lógica completamente proposicional que complementava a Aristótelica. Eles também pesquisaram várias antinomias lógicas[1] entre as quais tem-se os paradoxos[2]. A escola megariana foi fundada por Euclides de Megara (435-365 a.C.) e teve como discípulo Eubulides de Mileto (384-322 a.C.), a quem se atribui o paradoxo do mentiroso, que em uma de suas diversas formas

"Se um homem diz que está mentindo.
O que ele diz é verdade ou mentira?"

Já o estoicismo, que propõe viver de acordo com a lei racional da natureza e aconselha a indiferença (*apathea*) em relação a tudo que é externo ao ser, foi fundado por Zenão de Citio (334-262 a.C.) por volta do ano 300 a.c. e teve esse nome devido ao lugar onde ele costumava ensinar: pórtico, que em grego é *stoá*. Entre os argumentos típicos do estoicismo temos o seguinte [BRA98]:

"Se Você sabe que está morto, você está morto.
Mas se voce sabe que está morto, você não está morto,
portanto você não sabe se está morto ou não."

O mais famoso dos paradoxos devido a esta escola é o paradoxo do barbeiro que diz assim

"Numa pequena aldeia só existia um barbeiro.
O barbeiro faz a barba de todas as pessoas da aldeia
que não se barbeiam a si próprio e a mais ninguém."

[1]Uma antinomia é a afirmação simultânea de duas proposições (teses, leis, etc.) contraditórias.

[2]Um paradoxo é uma afirmação aparentemente verdadeira que leva a uma contradição lógica, ou a uma situação que contradiz a intuição comum.

Mas quem barbeia o barbeiro? Se ele mesmo se barbeasse, estaria barbeando uma pessoa que barbeia a si mesmo e se não se barbear ele não estaria barbeando a uma pessoa da aldeia que não se barbeia a si mesmo.

Esses argumentos alimentaram ricas e extensas discussões que teve como um corolário a lei do terceiro excluído, na qual uma proposição somente pode ter dois possíveis valores verdade, sendo excluídos os valores intermediários.

Nos períodos seguintes houveram poucos desenvolvimentos em lógica. O estudo e pesquisa da lógica só ressurgiu na era moderna, por trabalhos desenvolvidos por matemáticos como o alemão Gottfried Wilhelm Leibniz (1646-1716), quem pretendia desenvolver uma linguagem universal a ser especificada com precisão matemática cujo objetivo era reduzir especulações científicas e filosóficas a computações. Embora este grandioso projeto tenha ficado longe de ser desenvolvido com êxito e ainda mais não tenha sido diretamente muito influente, ele pode ser considerado um precursor para muitos dos trabalhos subseqüentes em lógica matemática.

A princípios do século IXX, o filósofo e matemático checo, Bernhard Bolzano (1781-1848), desenvolveu algumas noções fundamentais para lógica, tais como: conseqüência lógica e analítica. Já na segunda metade do século IXX, o matemático e lógico inglês George Boole (1815-1864), o matemático indiano Augustus De Morgan (1806-1871) e outros pesquisadores menos célebres, usaram a lógica como formalismo para representar os processos de raciocínio. Esta linha de pesquisa se focalizava sobre as relações entre regularidades entre o raciocínio correto e operações como soma e multiplicação. Outros membros desta escola, chamada de algébrica, desenvolveram quantificadores rudimentares os quais eram extensões, ainda que infinitárias, das conjunções e disjunções.

A seguinte escola lógica foi a logisista e tinha por objetivo codificar o entendimento lógico de todo discurso científico ou racional num único sistema. Para eles lógica não é o resultado de abstrações de raciocínios em disciplinas e contextos específicos, pois lógica trata das características gerais do discurso preciso em si, independente das características do contexto. O maior expoente desta escola foi o matemático e lógico alemão Gottlob Frege (1848-1925) quem desenvolveu uma linguagem formal com rigor matemático e introduziu a noção de variáveis, quantificadores universal e existencial, relações e funções, lógica proposicional e axiomatizações da lógica. Frege demonstra seu princípio de indução a partir de seus princípios lógicos. Um dos princípios que nortearam Frege foi o de tentar desenvolver a matemática como parte da lógica. Alguns outros importantes lógicos desta escola são Bertrand Russell (1872-1970), Alfred Whitehead (1861-1947) e o filósofo austríaco Ludwig Wittgenstein (1889-1951). Os dois primeiros publicaram em conjunto o livro Principia Mathematica, no qual tentam mostrar, influenciados na visão de Frege, que a a matemática pode ser desenvolvida como parte da lógica. Sua axiomatização e formalismos foram referência para a lógica por mais de 20 anos. Já Wittgenstein, no seu *Tractatus Logico-Philosophicus*, apresenta por primeira vez a lógica proposicional na forma do que hoje em dia como tabelas verdades. Wittgenstein afirma que qualquer proposição é o resultado de sucessivas aplicações de operações lógicas sobre proposições elementares.

Já a escola matemática propunha a axiomatização de alguns ramos da matemática, como geometria, aritmética, análises e teoria dos conjuntos. O matemático alemão Erns Zermelo (1871-1953), por exemplo, descreveu uma axiomatização da teoria dos conjuntos, posteriormente foi enriquecida com trabalhos de Thoralf Skolem (1887-1963), Ludwig Fraenkel-Conrad (1910-1999), John von Neumman (1903-1957) e outros. Alguns membros desta escola pensaram na importância de incluir uma formulação explícita das regras de inferência no desenvolvimento axiomático. Por exemplo, o holandês Arend Heyting (1898-1980) produziu versões axiomáticas da lógica intuicionista.

Uma variante da escola matemática ocorreu em Polônia, sobre a liderança de Jan Lukasiewics (1878-1956) . Nesta escola, chamada de escola polaca lógica, quando vista como uma teoria axiomática, é considerada uma área da matemática. Eles desenvolveram e analisaram os sistemas axiomáticos da lógica proposicional, lógica modal e álgebra booleana.

Quando a atenção foi centrada na linguagem e na axiomatização enquanto objetos para um estudo matemático direto levou a um grupo de matemáticos desta escola, que embasados com o advento da geometria não-euclideana, a considerar interpretações alternativas das linguagens e algumas questões metalógicas dos sistemas lógicos, tais como independência, consistência e completude. Nesta escola algumas noções sintáticas foram diferenciadas de sua contrapartida semântica. Assim, neste ponto houve uma clara divisão da lógica entre os lógicos que viam a lógica como linguagem e os matemáticos e algebristas que viam a lógica como um cálculo. A pesar dos eventuais conflitos entre estas correntes, houveram diversas interações. A lógica contemporânea é de alguma forma uma mistura de ambas visões.

O intensivo trabalho sobre problemas matemáticos culminaram nos resultados do lógico e matemático alemão Kurt Gödel (1906-1978) quem na sua teses de doutorado em 1929 mostrou que uma fórmula é dedutível num sistema lógico (teoria formal) clássico de 1^a ordem se e somente se é universalmente válida, ou seja é verdadeira em toda interpretação. Este teorema é conhecido como teorema da completude da lógica de predicados (também conhecida como lógica clássica de 1^a ordem). Em 1931, Gödel publicou seu resultado mais famoso hoje conhecido como teorema da incompletude de Gödel o qual diz que para qualquer tentativa de axiomatização da aritmética sempre é possível construir uma sentença aritmética que é verdade mas a qual não poderá ser deduzida (provada) a partir dessa axiomatização. Ou seja ele mostra que não há um meio mecânico (procedimento efetivo) que dado os axiomas da teoria produza outras sentenças. Mas que é um procedimento efetivo? esta questão levou ao desenvolvimento de máquinas teóricas por parte de diversos matemáticos que desencadearam na teoria da computabilidade (ou recursividade) que é base fundamental da computação enquanto ciência.

Em 1900, durante o Segundo Congresso Internacional de Matemática em Paris, o matemático David Hilbert (1862-1943), propõe 23 problemas para o desenvolvimento da matemática no século que iniciava. O segundo de tais problemas pedia para provar que os "axiomas aritméticos"são não contraditórios, ou seja que uma quantidade finita de passos lógicos baseados nele jamais levariam a uma contradição. Hilbert acreditava que qualquer

problema matemático tinha solução e que ela poderia ser encontrada pela pura razão porque em matemática não existiria *ignorabimus*. Em 1928 Wilhelm Ackermann (1896-1962), discípulo de Hilbert, revisou as aulas de Hilbert de 1917, levando-o a publicar, junto com o seu mestre, uma sucinta apresentação da lógica matemática (*Grundzüge der theoretischen Logik*) a qual fez bastante sucesso na época, principalmente por ser mais acessível que o *Principia Mathematica* de Whitehead e Ruseel. Este livro colacava a necessidade de se saver se toda fórmula universalmente válida poderia ser derivada do sistema axiomático do *Principia Mathematica* (completude do sistema). Também em 1928, no Congresso Internacional de Matemática, David Hilbert levantou três questões, a terceira das quais é hoje conhecida com "Hilbert's *Entscheidungsproblem*"(problema de decisão em alemão) [Hod83]. O Entscheidungsproblem pede para encontrar um algoritmo capaz de decidir quando uma fórmula da lógica de 1^a ordem é universalmente válida ou não. Posteriormente em 1930 ele manifestou que acreditava que esse problema não tinha solução ou seja que não existia tal algoritmo [Hod83].

O matemático e filósofo norte-americano, Alonzo Church (1903-1995) influenciou tanto a lógica matemática como a filosófica. Um dos principais resultados em lógica de Church foi provar em 1936 que as verdades universais não são recursivas, ou seja não existe uma forma computacional de enumerar todas as verdades universais da lógica de 1^a ordem. Desta forma Church prova que o problema *Entscheidungsproblem* não tem solução e portanto é indecidível. De forma independente o lógico matemático inglês Alan Turing (1912-1954), também prova este mesmo resultado em 1936. Nesses trabalhos, eles introduziram modelos teóricos diferentes e argumentaram que esses modelos capturavam a noção intuitiva de procedimento efetivo (tese de Church-Turing). Esta tese tem-se constituído no cerne da teoria da computabilidade, teoria que permite determinar limites à ciência da computação [SB04].

Church, junto com seus estudantes Stephen Kleene (1909-1994) e Leon Henkin desenvolveram uma grande diversidade de temas em lógica tanto do ponto de vista da filosofia como da matemática. Entre os tópicos por eles abordados tem-se completude, definibilidade, computabilidade, lógica de segunda ordem, sentido, referência, etc.

O lógico e matemático polonês, Alfred Tarski (1902-1983), entre seus muitos aportes à lógica moderna, está sua definição de verdade, conseqüência lógica (semântica) e de satisfação. Porém, isto é só uma pequena parte de seu trabalho. Sua principal contribuição foi ter iluminado a metodologia de sistemas dedutivos e noções lógicas fundamentais tais como completude, decibilidade, consistência, satisfabilidade e definibilidade.

Sintetizando, podemos dizer que a lógica contemporânea se caracteriza pela tendência da matematização da lógica e pelo reconhecimento da necessidade de se ter lógicas diferentes da clássica [BRA98].

1.5 Apresentação dos Próximos Capítulos

Capítulo 2: Neste capítulo são abordas as noções básicas de linguagens formais e de Σ-álgebras. Linguagens formais são introduzidas de modo diferente ao usual dado em disciplinas de linguagens formais [ABL02], pois aqui só precisamos das intuições que estão por traz da noção de linguagem formal e que nos serão úteis para descrever as linguagens formais das lógicas abordadas neste texto. Por outro lado Σ-álgebras, são estruturas matemáticas adequadas para se dar semântica a linguagens formais. Aqui só vemos Σ-álgebras monosortidas pois é o suficiente para dar semântica às linguagens formais das lógicas aqui abordadas. Para o bom entendimento do restante do texto é imprescindível um bom entendimento de linguagens formais, já Σ-álgebras só é aconselhado para quem deseja ter um entendimento mais profundo do aspecto semântico das linguagens lógicas assim como para quem quer fazer um estudo inicial de Σ-álgebra, conhecimento este que lhe será útil para desenvolver uma teoria algébrica de especificação de tipos abstratos de dados [GTW77] assim como para ter um entendimento melhor e mais abstrato da disciplina de algebra, pre-requisito básico da disciplina de Lógica nos cursos de Ciências da Computação. O importante para o leitor é que fique claro que Σ-álgebra, neste contexto, são ambientes matemáticos adequados para se dar uma interpretação formal à linguagem proposicional, já no caso da linguagem de predicado esta semântica é dada através da noção de Σ-domínios (uma noção estendida de Σ-álgebras).

Capítulo 3: Neste capítulo veremos a linguagem da lógica de proposicional clássica, primeiro como uma linguagem para descrever de forma sucinta, afirmações simples (atômicas) e composta de uma realidade, e depois como uma linguagem formal, cujas palavras (fórmulas) podem ser interpretadas através de σ-álgebras. Depois será visto o conceito de Σ-álgebra booleana e como uma de tais Σ-álgebras booleanas (a dos valores verdadeiro e falso) pode ser vista como catalisadora de todas as interpretações. Finalmente veremos os conceitos e propriedades de tautologias, contradições, conseqüência lógica e equivalência lógica, entre outro.

Capítulo 4: Neste capítulo veremos o conceito de teorias formais, de forma geral, e conceitos correlatos tais como prova, teorema e conseqüência sintática, assim como propriedades gerais de tais sistemas. Depois apresentaremos uma teoria formal para a lógica proposicional clássica. Veremos que realizar provas nesta teoria pode em alguns casos ser difícil, pelo que para facilitar encontrar tais provas, introduzimos a idéia de re-utilizar provas de teoremas já provados assim como o uso de uma ferramenta poderosa: O teorema da dedução. Finalmente mostramos quatro outras teorias equivalentes para a lógica proposicional.

Capítulo 5: Este capítulo tem por objetivo fornecer um método computacional, chamado de resolução, para decidir quando uma fórmula da linguagem proposicional é ou não uma

tautologia. Este método se baseia na idéia de refutação, ou seja primeiro se nega a fórmula a ser verificada para depois provar que ela é uma contradição. Para isto depois de negada é transformada a uma forma normal (a forma normal conjuntiva) a qual garante que a formula normal e a original são logicamente equivalentes. Depois são eliminados literais complementares das cláusulas que fazem parte da fórmula na forma normal conjuntiva até se chegar à cláusula vazia. Finalmente neste capítulo é mostrado que uma fórmula da linguagem proposicional é uma tautologia se e somente se é um teorema da teoria formal proposicional se e somente se sua negação pode derivar a cláusula vazia, ou seja os três métodos vistos até este ponto para lidar com verdades na lógica proposicional: semântico (tabelas verdades), sintático (teoria de provas) e sintático-computacional (resolução) são equivalentes.

Capítulo 6: A lógica proposicional porém, não é suficientemente poderosa para expressar afirmações envolvendo quantificações de objetos, como por exemplo "para todo grande homem existe uma grande uma mulher" ou "nem todo o que brilha é ouro". Neste capítulo será apresentada a linguagem de predicados e de forma análoga ao capítulo 3 será visto como usar esta linguagem para se descrever afirmações como as anteriores. Depois é descrita esta linguagem como uma linguagem formal para em seguida mostrar como fórmulas desta linguagem podem ser interpretadas através de Σ-domínios. Aqui veremos quando uma fórmula é verdadeira, falsa ou contingente numa interpretação e quando é sempre verdadeira (universalmente válida) ou sempre falsa (contradição ou insatisfatível), independente da interpretação. Introduziremos as noções de conseqüência e equivalência lógica e serão vistas algumas propriedades e meta-teoremas desta lógica.

Capítulo 7: Um dos maiores problemas da visão semântica da lógica de predicados é que na maioria das vezes pode resultar inviável se mostrar que uma fórmula é universalmente válida de forma direta de sua definição, ou seja através de analisar todas as possíveis interpretações. Neste capítulo é apresentado um sistema de provas para a lógica de predicados o qual, como veremos no capítulo 8, terá como teoremas exatamente a fórmula universalmente válidas. Porém onde a noção de conseqüência sintática não coincide com a noção de conseqüência lógica. Veremos também que o teorema da dedução para esta teoria formal vai requerer que a prova satisfaça certas propriedades, o qual não ocorria no caso proposicional nem na versão semântica deste teorema.

Capítulo 8: Embora o método de provas seja suficiente para determinar de maneira segura quando uma fórmula é universalmente válida, ele não fornece um algoritmo para se obter uma tal prova. Assim, neste capítulo veremos uma generalização do método de resolução visto no capítulo 5 para a lógica de predicados. Neste sentido, veremos que para se chegar ao conjunto de cláusulas que garante que a fórmula original é uma contradição se e somente se esse conjunto for insatisfatível, precisaremos de vários passos extras. O primeiro é levar os quantificadores para o lado mais externo da fórmula (forma normal prenex), depois fazer o fecho existencial da fórmula (completar com quantificadores

existenciais daquelas variáveis não quantificadas), depois eliminar as quantificações existenciais substituindo suas variáveis por funções (skolemização) e finalmente transformar a parte da fórmula sem quantificadores na sua forma normal conjuntiva seguindo o mesmo algoritmo do capítulo 5 para este fim. A partir deste ponto pode-se ir eliminando os literais complementares, porém a diferença do caso proposicional antes deveremos fazer a unificação deles. Finalmente é provada a completude das três visões da lógica de predicados, ou seja que uma fórmula é universalmente válida se e somente se é um teorema da teoria formal de 1^a ordem se e somente se sua negação pode ser refutada pelo método de resolução.

Capítulo 9: Neste capítulo apresentaremos uma breve introdução às linguagens de programação em lógica, ou seja linguagens de programação que permitem descrever problemas ou situações através de um conjunto finito de sentenças lógicas. Assim, um programa em lógica não descreve um procedimento de como achar a solução mas o problema que ser quer resolver. A máquina de inferência da linguagem será a encarregada de achar as respostas às perguntas que o usuário faz ao programa. A mais popular das linguagens deste paradigma de programação, e que será vista neste capítulo, é a linguagem Prolog. Esta linguagem é baseada na lógica clássica e usa o método de resolução para determinar as soluções dos problemas descritos pelos programas Prolog. Apresentaremos também algumas estruturas extra-lógica desta linguagem, como comandos de controle e maneiras de lidar com dados numéricos e listas.

Capítulo 10: Neste capítulo apresentaremos quatro outras maneiras de apresentar a lógica proposicional e de predicados. As duas primeiras, dedução natural e calculo de sequentes de Gentzen, são formas alternativas ao sistema de prova, ou seja permitem determinar se uma fórmula é universalmente válida ou uma fórmula é conseqüência sintática de um conjunto de fórmulas sem precisar de recorrer a interpretações (semântica), mas sem fornecer um algoritmo. O terceiro e o quarto método apresentado, o tableau e tableau com unificação, se baseiam no princípio da refutação, são métodos computacionais alternativos à resolução, ou seja podem ser usados para se provar automaticamente teoremas. Finalmente mostraremos que estes métodos são equivalentes aos outros três (semântico, teoria de provas e resolução).

Apêndice: Cada capítulo, a partir do segundo, tem ao final uma lista de exercícios. Neste apêndice apresentaremos a solução de mais de 100 de tais exercícios.

Capítulo 2

Linguagens Formais

Em Português existem três tipos de entidades diferentes: letras, palavras e sentenças. Existe um certo paralelismo entre elas, no sentido de que grupos de letras constituem uma palavra, e grupos de palavras uma sentença. Mas, nem toda concatenação de letras forma uma palavra, nem toda seqüência de palavras uma sentença. A analogia pode se estendida para parágrafos, histórias, e assim por diante. A situação, se dá, também, para linguagens de programação, na qual certos agrupamentos de palavras chaves são um comando e algumas seqüências de comandos são programas.

Para construir uma teoria geral que unifique todos estes casos, será necessário adotar uma definição para a estrutura de linguagens mais universal, isto é, uma estrutura na qual a decisão de quando uma cadeia de unidades constitui uma unidade maior válida seja baseada na aplicação de regras explicitamente definidas.

Em linguagens tipo Português, Inglês, alguns dialetos tipo Guarani, Aymara, etc. não podemos dar regras que nos permitam descrever todas as frases coerentes da linguagens. Isto porque são linguagens ditas naturais, na qual depende de nossa habilidade para interpretar metáforas poéticas de sentenças mal escritas. Isto já não acontece com linguagens de programação, pois elas são "formais", e o compilador de uma linguagem de programação é um procedimento que analisa a corretude de uma seqüência de símbolos e determina se ela constitui um programa válido ou não.

Neste capítulo estudaremos alguns aspectos básicos do que se conhece como "teoria de linguagens formais". A palavra "formal" diz respeito a que todas as regras para a linguagem são explicitamente declaradas em termos das cadeias de símbolos que podem ocorrer nela. Nenhuma liberdade é tolerada e nenhum "entendimento profundo" é preciso. Sentenças serão consideradas como meros símbolos e não como expressões de idéias na mente humana. Neste modelo básico, Linguagens (formais) não é uma comunicação entre intelectos, mas um jogo de símbolos com regras formais. O termo "formal" aqui usado enfatiza que é a *forma* da cadeia de símbolos que nos interessa e não seu significado.

2.1 Linguagens Formais

Toda linguagem é constituída de dois elementos básicos, um **alfabeto** que especifica um conjunto contável de símbolos usados na linguagem e uma **gramática** que caracteriza sua sintaxe, isto é, que especifica como estes símbolos podem ser agrupado formando as expressões admissíveis da linguagem. O que diferencia uma linguagem formal de uma linguagem natural é que na linguagem formal a gramática é especificada precisamente, enquanto na linguagem natural quase sempre isso não é possível.

Seja Σ um conjunto contável, isto é, que esteja em correspondência biunívoca com um subconjunto dos números naturais. Denotaremos por Σ^* o conjunto de todas as cadeias finitas em Σ, incluindo a cadeia vazia. Por exemplo, se $\Sigma = \{a, b\}$ então $\Sigma^* = \{\lambda, a, b, aa, ab, bb, ba, aaa, aab, \ldots\}$, onde λ simboliza a cadeia vazia.

Definiremos como uma **linguagem** L, sobre Σ, qualquer subconjunto de Σ^*. Observe que, segundo essa definição, o próprio Σ^* e o conjunto vazio são linguagens sobre Σ. Em geral, estaremos interessados em obter L de forma algorítmica. Por isso, de ora em diante daremos, também, a seguinte definição de linguagem formal.

Definição 2.1.1 *Uma* **Linguagem formal** *é um par $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, onde Σ é um conjunto contável, denominado* **alfabeto**, *e $\mathcal{G}$ é um conjunto finito de regras de derivação, denominado* **gramática**,*cujo fim é dizer como os símbolos do alfabeto podem ser agrupados de modo a formar as expressões admissíveis da linguagem.*

Descreveremos as regras de derivação em $\mathcal{G}$ por

$$\frac{x_1, \ldots, x_n}{x},\ n \in \mathbb{N},$$

onde, os objetos do numerador e do denominador podem conter metavariáveis, que podem ser instanciadas por qualquer termo da linguagem. Se entende que os objetos do numerador dão origem ao do denominador. No caso em que $n = 0$ estaremos gerando um elemento x a partir do conjunto vazio, o qual denotaremos por $\frac{-}{x}$. Chamaremos a estes x's de **axiomas da linguagem formal** $\mathcal{L}$. As variáveis, $x_1, \ldots, x_n$ usadas nas regras de derivação representam qualquer cadeia em Σ^* e x é uma expressão constituída das variáveis x_i's com elementos de Σ.

Exemplo 2.1.2 *Seja o alfabeto $\Sigma = \{a, b\}$ e o conjunto de regras $\mathcal{G} = \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3\}$ definidas a seguir:*

$$\mathsf{g}_1 : \frac{-}{a} \qquad \mathsf{g}_2 : \frac{t}{at} \qquad \mathsf{g}_3 : \frac{t}{bt}$$

onde at é a cadeia resultante de concatenar o símbolo a com a cadeia t. Isto é, se $t = bba$ então $at = abba$.

Podemos ver as regras de derivação de $\mathcal{G}$ como um **cálculo** ou **algoritmo** para gerar os elementos da linguagem L. No exemplo 2.1.2 o cálculo $\mathcal{G} = \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3\}$ gera um conjunto de cadeias a partir da constante a.

Assim, uma cadeia $w \in \Sigma^*$ é gerada pela linguagem formal $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, se existem cadeias $w_1, \ldots, w_n$ geradas por $\mathcal{L}$ e uma regra de derivação

$$g.\ \frac{x_1, \ldots, x_n}{x},\ n \in \mathbb{N},$$

tal que $g \in \mathcal{G}$ e substituindo cada ocorrência x_i por w_i em g, obtemos um x igual a w. Observe que para esta recursão na definição de cadeias geradas por uma linguagem formal parar, devemos ter pelo menos uma regra de derivação sem numerador ($n = 0$).

Definição 2.1.3 *Seja $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ uma linguagem formal. Um elemento $x \in \Sigma^*$ é* **gerado** *por $\mathcal{L}$ se existe uma seqüência finita de elementos de Σ^*, $x_1, x_2, \ldots, x_n$ tal que $x_n = x$ e cada x_k, $1 \leq k \leq n$, é o resultado da aplicação de uma regra de derivação de $\mathcal{G}$ aos elementos x_i, $i < k$. A seqüência $x_1, \ldots, x_n$, anterior, é chamada uma* **prova** *ou dedução de x em $\mathcal{L}$ e x é chamado de* **fórmula bem formada** *(**fbf**) de $\mathcal{L}L$.*

Exemplo 2.1.4 *No caso da linguagem formal do exemplo 2.1.2 a cadeia baa pode ser gerada, pois ela é obtida aplicando a regra g_3 à cadeia ba a qual, por sua vez também é gerada pela linguagem formal $\mathcal{L}$, pois ela pode ser obtida aplicando a regra g_2 à cadeia a, a qual por sua vez também faz parte da linguagem, pois ela pode ser obtida diretamente aplicando a regra g_1. Ou seja, uma prova de que a cadeia baa faz parte da linguagem descrita pela linguagem formal $\mathcal{L}$ é a seguinte seqüência de cadeias:*

$$\begin{array}{lll} (1) & a & (g_1) \\ (2) & aa & (1, g_2) \\ (3) & baa & (2, g_3) \end{array}$$

Usaremos a notação $\vdash_{\mathcal{L}} w$, ou simplesmente $\vdash w$, se o contexto tornar claro o cálculo, para indicar que w foi derivado, usando uma prova.

Exemplo 2.1.5 *Considere a linguagem formal $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, onde $\Sigma = \{zero, suc, soma, (\ ,\ )\}$ e a gramática $\mathcal{G}$ é constituída das seguintes regras de derivação:*

$$\mathsf{g}_1 : \frac{-}{zero} \qquad \mathsf{g}_2 : \frac{t}{suc(t)} \qquad \mathsf{g}_3 : \frac{t_1, t_2}{t_1\ soma\ t_2}$$

Nas regras g_1 e g_2, t, t_1 e t_2 são variáveis as quais podem tomar como valor qualquer elemento já conhecido da linguagem. Assim, $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, onde $\mathcal{G} = \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3\}$ especifica uma linguagem $L \subseteq \Sigma^*$ que contém os seguintes elementos: zero, suc(zero), zero soma zero, suc(zero) soma zero, zero soma suc(zero), zero soma zero soma zero, ...

Nesta linguagem formal pode ser gerada a cadeia zero soma suc(suc(zero)), isto é $\vdash_{\mathcal{L}}$ *zero soma suc(suc(zero)). Uma prova disto é a seguinte*

(1)	$zero$	(g_1)
(2)	$suc(zero)$	$(1, g_2)$
(3)	$suc(suc(zero))$	$(2, g_2)$
(4)	$zero\ soma\ suc(suc(zero))$	$(1, 3, g_3)$

Diremos, também, que esta prova é uma **construção** *do elemento*

$$zero\ soma\ suc(suc(zero)).$$

Assim, toda linguagem formal $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ especifica uma linguagem que consiste em todas as cadeias geradas pela linguagem formal $\mathcal{L}$. Especificar uma linguagem $L \subseteq \Sigma^*$ através de uma linguagem formal, significa fornecer um algoritmo para obter L. é claro que nem sempre é possível dar um algoritmo para descrever uma linguagem, nesse caso dizemos que a linguagem é uma **linguagem natural**.

A **linguagem gerada ou especificada por uma linguagem formal** $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, denotado por $Ling(\mathcal{L})$, é definida pelo seguinte conjunto de cadeias:

$$Ling(\mathcal{L}) = \{x \in \Sigma^* / \ \vdash_{\mathcal{L}} x\}$$

Exemplo 2.1.6 *No exemplo 2.1.2, o conjunto de todas as possíveis cadeias que podem ser geradas pela linguagem formal* $\mathcal{L}$*, isto é* $Ling(\mathcal{L})$*, é a seguinte:*

$$Ling(\mathcal{L}) = \{a, aa, ba, aaa, baa, aba, bba, aaaa, baaa, abaa, bbaa, aaba, baba, abba, bbba, \ldots\}$$

ou intencionalmente o conjunto

$$Ling(\mathcal{L}) = \{w \in \Sigma^* \ / \ w = va \text{ para algum } v \in \Sigma^*\}$$

ou seja a linguagem composta de todas as cadeias no alfabeto $\{a, b\}$ *cujo último símbolo seja* a*.*

Exemplo 2.1.7 *Considere a linguagem de todos os palíndromos*[1] *no alfabeto* $\{a, b\}$*. Uma linguagem formal que especifica (ou gera) esta linguagem seria a seguinte:* $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$*, onde* $\Sigma = \{a, b\}$ *e* $\mathcal{G} = \{\mathtt{g_1}, \mathtt{g_2}, \mathtt{g_3}, \mathtt{g_4}, \mathtt{g_5}\}$ *definidas como segue:*

[1] Palíndromos são cadeias que quando lidas de esquerda a direita é a mesma que quando lidas de direita a esquerda. Mais formalmente, uma cadeia $w = a_1 a_2 \ldots a_n$ é um palíndromo se,e somente se, $w = a_n a_{n-1} \ldots a_1$

$$\mathsf{g}_1 : \frac{-}{\lambda} \qquad \mathsf{g}_2 : \frac{-}{a} \qquad \mathsf{g}_3 : \frac{-}{b}$$

$$\mathsf{g}_4 : \frac{t}{ata} \qquad \mathsf{g}_5 : \frac{t}{btb}$$

Naturalmente, duas linguagens formais são **equivalentes** se elas geram a mesma linguagem. Por exemplo, se à linguagem formal do exemplo 2.1.5 adicionarmos a regra

$$\mathsf{g}_4 : \frac{t}{t\ somat}$$

não vai ser gerada nenhuma cadeia nova, uma vez que ela pode ser obtida aplicando a regra g_3 ao mesmo termo, isto é, fazendo $t_1 = t_2 = t$. Assim, a linguagem gerada por esta "nova" linguagem formal vai ser exatamente a mesma que a gerada pela linguagem formal do exemplo 2.1.5. Então, generalizando, duas linguagens formais $\mathcal{L}_1 = \langle \Sigma_1, \mathcal{G}_1 \rangle$ e $\mathcal{L}_2 = \langle \Sigma_2, \mathcal{G}_2 \rangle$ são equivalentes, se elas possuem o mesmo alfabeto, isto é $\Sigma_1 = \Sigma_2$ e cada regra de $q \in \mathcal{G}_1$ pode ser derivada em $\mathcal{L}_2$ e vice-versa, cada regra $g \in \mathcal{G}_2$ pode ser derivada em $\mathcal{L}_1$.

2.2 Σ-álgebras

Seja L um conjunto não vazio. Diremos que $(L, O_1, O_2, \ldots, O_n)$ é uma **álgebra** se $O_1, O_2, \ldots, O_n$ são operações sobre L. Lembre que O é uma operação sobre L, de aridade k, cuja notação será $arid(O) = k$, se O é uma função de L^{k2} em L; $O : L^k \longrightarrow L$. As constantes de L são consideradas operações de aridade zero.

Uma linguagem L especificada por $\langle \Sigma, \mathcal{G} \rangle$ pode ser vista como uma álgebra, onde cada regra de derivação

$$\frac{x_1, \ldots, x_n}{x},$$

é interpretada como uma operação $O : L^n \longrightarrow L$. Na realidade, podemos estudar as linguagens formais de uma perspectiva exclusivamente algébrica como **semântica** ou **interpretação** da abordagem dedutiva. Na ciência da computação a abordagem algébrica de linguagens de programação é conhecida como **semântica denotacional** e a abordagem dedutiva como **semântica operacional**. Como neste trabalho abordaremos, também, os

[2]A notação L^k denota k-vezes o produto cartesiano de L com ele mesmo, isto é,

$$L^k = \underbrace{L \times \ldots \times L}_{k-vezes}$$

.

aspectos de interpretação, a seguir desenvolveremos os princípios de Σ-álgebras necessários nas discussões de semântica denotacional.

Uma **assinatura** é um conjunto contável, Σ, de símbolos de funções. Por exemplo, os três símbolos de função, $\Sigma = \{zero, suc, plus\}$, seria uma assinatura apropriada para os números naturais. Cada símbolo de função tem associado uma aridade que fornece o número de argumentos que qualquer função que interpreta este símbolo deve receber. No caso a aridade de *zero* é 0, de *succ* é 1 e de *plus* é 2. *zero* é uma constante, vista como uma função de aridade 0.

Definição 2.2.1 *Uma* **assinatura** *é uma par* $\langle \Sigma, arid \rangle$*, onde* Σ *é um conjunto contável de símbolos de função e arid é uma função, chamada* **aridade**, $arid : \Sigma \longrightarrow \mathbb{N}$*, tal que a cada símbolo* $s \in \Sigma$*, associa sua aridade* $arid(s) \in \mathbb{N}$*.*

Quando fôr claro do contexto omitiremos a função *arid*, designando uma assinatura simplesmente por Σ.

Definição 2.2.2 *Seja* Σ *uma assinatura. Uma* **Σ-álgebra** *é um par* $\mathbf{A} = \langle A, \Sigma_A \rangle$ *onde:*

1. *A é um conjunto não vazio, chamado* **conjunto básico**.
2. Σ_A *é um conjunto de operações sobre A,* $\{f_A \ / \ f \in \Sigma\}$ *tal que se* $arid(f) = n$*, então* f_A *é uma operação de* $A^n \longrightarrow A$*.*

Assim, cada Σ-álgebra **A** é vista como uma interpretação, no conjunto A, dos símbolos de função por funções em A. Observe que aqui devemos distinguir entre o símbolo usado para a função (o nome, por assim dizer) e a função propriamente dita.

Exemplo 2.2.3 *Considere a assinatura* $\langle \Sigma, arid \rangle$ *onde* $\Sigma = \{zero, suc, soma\}$ *e* $arid(zero) = 0$*,* $arid(suc) = 1$ *e* $arid(soma) = 2$*. Então,* $\langle \mathbb{N}, \Sigma_{\mathbb{N}} \rangle$ *é uma Σ-álgebra, onde:*

1. $\mathbb{N}$ *é o conjunto dos números naturais* $\{0, 1, 2, \ldots\}$*.*
2. $\Sigma_{\mathbb{N}}$ *é dada por:*
   (a) $zero_{\mathbb{N}}$ *é o número 0.*
   (b) $suc_{\mathbb{N}} : \mathbb{N} \longrightarrow \mathbb{N}$ *é definida por* $suc(n) = n + 1$
   (c) $soma_{\mathbb{N}} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ *é definida por* $soma_{\mathbb{N}}(m, n) = m + n$

Aqui "+" denota a adição da aritmética usual.

As únicas restrições para definir Σ-álgebras, é que as operações que interpretação os símbolos de operação tenham a mesma aridade. Desse modo, o símbolo de operação soma é um mero nome, e portanto, pode ser interpretado de diversas maneiras, desde que sua interpretação seja uma operação binária. Assim, uma outra Σ-álgebra para esta assinatura, pode ser $\langle \mathbb{N}, zero_{\mathbb{N}}, suc_{\mathbb{N}}, mult_{\mathbb{N}} \rangle$*, onde* $\mathbb{N}$*,* $zero_{\mathbb{N}}$ *e* $suc_{\mathbb{N}}$ *são como antes, mas* $mult_{\mathbb{N}} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ *é definido por* $mult_{\mathbb{N}}(x, y) = x \cdot y$*. Aqui,* $\cdot$ *denota a multiplicação da aritmética.*

$\langle \mathbb{N}, \Sigma_{\mathbb{N}} \rangle$ representa uma interpretação "natural" dos símbolos de função de Σ, sobre os números naturais. Por isso podemos dizer que uma Σ-álgebra é simplesmente uma interpretação de uma assinatura Σ. Ela consiste de um conjunto A e uma interpretação sobre A, dos símbolos de função em Σ. Uma Σ-álgebra pode ser vista como um conjunto com estrutura. O conjunto A tem uma estrutura imposta pelas funções em Σ_A, no sentido de que os elementos de A só podem ser manipulados usando essas funções.

2.3 Relação entre Linguagens Formais e Σ-álgebras

Dada uma assinatura $\langle \Sigma, arid \rangle$, se acrescentarmos a Σ os símbolos de parênteses e pontuação, teremos, automaticamente, uma linguagem $Ling(\mathcal{L})$ especificada pelo algoritmo $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$, onde a gramática $\mathcal{G}$ seria obtida, simplesmente, considerando as aridades dos símbolos de Σ, em $\langle \Sigma, arid \rangle$. Por exemplo, se $s \in \Sigma$ é tal que $arid(s) = 2$, então

$$\frac{t_1, t_2}{s(t_1, t_2)}$$

seria uma regra de derivação de $\mathcal{G}$. Em muitos casos poderíamos agir no sentido contrário, isto é, dado o alfabeto Σ, despojá-lo dos símbolos de parênteses e pontuação, obtendo assim, uma assinatura Σ, onde a função aridade seria obtida, simplesmente, observando as regras de derivação. Por exemplo, a uma regra como

$$\frac{t_1, \ldots, t_n}{t}$$

lhe seria associada um símbolo de função de aridade n. Portanto, existe uma relação estreita entre linguagens formais, isto é, linguagens obtidas pela especificação do algoritmo $\langle \Sigma, \mathcal{G} \rangle$ e assinaturas, no sentido de que sempre que uma linguagem é obtida dedutivamente é possível, usando as regras de derivação, obtê-la algebricamente, isto é como uma Σ-álgebra. Nesse sentido, podemos considerar a linguagem gerada por $\langle \Sigma, \mathcal{G} \rangle$ como o conjunto básico da Σ-álgebra que tem como operações os próprios símbolos funcionais. A Σ-álgebra assim obtida, será chamada **Σ-álgebra dos termos** e será denotada por $\langle T_\Sigma, \Sigma_{T_\Sigma} \rangle$. Assim, T_Σ seria a própria linguagem gerada pela linguagem formal associada

à assinatura Σ. Formalmente, podemos definir a Σ-álgebras dos termos indutivamente, como segue

1. Se $a \in \Sigma$ e $arid(a) = 0$ então $a \in T_\Sigma$
2. Se $f \in \Sigma$, $arid(f) = n$ e $t_1, \ldots, t_n \in T_\Sigma$ então $f(t_1, \ldots, t_n) \in T_\Sigma$

Assim T_Σ é um conjunto, o conjunto de todas as fbf da linguagem gerada pela linguagem formal associada à assinatura $\langle \Sigma, arid \rangle$. Por outro lado, se $f \in \Sigma$ tem aridade n então $f_{T_\Sigma} : T_\Sigma^n \longrightarrow T_\Sigma$ é definido por

$$f_{T_\Sigma}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n).$$

Seja $\Sigma_{T_\Sigma} = \{f_{T_\Sigma} \ / \ f \in \Sigma\}$. O par $\langle T_\Sigma, \Sigma_{T_\Sigma} \rangle$ é uma Σ-álgebra.

Esta Σ-álgebra é particularmente importante no estudo da sintaxe abstrata de linguagens.

Exemplo 2.3.1 *Seja a assinatura do exemplo 2.2.3. O conjunto dos termos dessa Σ-álgebra é dado pelo seguinte conjunto:*

$$T_\Sigma = \{zero, suc(zero), zero\ soma\ zero, suc(suc(zero)), zero\ soma\ suc(zero), \ldots\}$$

Observe, que T_Σ é igual à linguagem gerada pela linguagem formal do exemplo 2.1.5.

As operações sobre T_Σ, isto é Σ_{T_Σ}, são as seguintes:

- $zero_{T_\Sigma} = zero$
- $suc_{T_\Sigma} : T_\Sigma \longrightarrow T_\Sigma$ *definida por* $suc_{T_\Sigma}(t) = suc(t)$.
- $soma_{T_\Sigma} : T_\Sigma \times T_\Sigma \longrightarrow T_\Sigma$ *é definida por* $soma_{T_\Sigma}(t_1, t_2) = t_1\ soma\ t_2$.

Observe, que $suc_{T_\Sigma}(t)$ mapeia um termo (t) num outro termo (suc(t)).

As observações acima esclarecem a relação entre as perspectivas formal e algébrica de uma linguagem. No caso de linguagens de programação a sua sintaxe, geralmente, é especificada por uma gramática livre de contexto. Mas, muitas linguagens de programação podem ser construídas de tal modo que a sua sintaxe é a Σ-álgebra dos termos, para algum Σ. Definir uma semântica denotacional para uma linguagem $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ consiste em construir uma Σ-álgebra particular satisfazendo certas exigências.

No sentido de explicar o significado matemático da Σ-álgebra dos termos introduziremos o conceito de Σ-homomorfismo, que são, simplesmente, funções entre os conjuntos básicos das Σ-álgebras que preservam as operações.

Definição 2.3.2 *Sejam $\langle A, \Sigma_A\rangle$ e $\langle B, \Sigma_B\rangle$ Σ-álgebras. Uma função $h : A \longrightarrow B$ diz-se um* **Σ-homomorfismo** *se para todo $f \in \Sigma$, de aridade n,*

$$h(f_A(a_1, \ldots, a_n)) = f_B(h(a_1), \ldots, h(a_n)) \tag{2.1}$$

Claramente, na definição acima, se $n = 0$, isto é, se aplicamos o Σ-homomorfismo a uma constante a_A, se deve satisfazer

$$h(a_A) = a_B$$

Exemplo 2.3.3

1. *Seja $\Sigma = \{zero, suc, soma\}$, como anteriormente. Então $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}}\rangle$, onde $\mathbb{Z}$ é o conjunto dos números inteiros e as funções de $\Sigma_{\mathbb{Z}}$ são definidas como no exemplo 2.2.3, é uma Σ-álgebra. A função $em : \mathbb{N} \longrightarrow \mathbb{Z}$ tal que $em(n) = n$, é um Σ-homomorfismo entre as Σ-álgebras $\langle \mathbb{N}, \Sigma_{\mathbb{N}}\rangle$ e $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}}\rangle$.*

2. *Seja Σ como acima e seja $\mathbb{P} = \{0, 2, 4, 6, \ldots\}$. Defina $\Sigma_{\mathbb{P}}$ por:*

   (a) $zero_{\mathbb{P}} = 0$

   (b) $suc_{\mathbb{P}} : \mathbb{P} \longrightarrow \mathbb{P}$ *é definida por* $suc_{\mathbb{P}}(n) = n + 2$

   (c) $soma_{\mathbb{P}} : \mathbb{P}^2 \longrightarrow \mathbb{P}$ *é definida por* $soma_{\mathbb{P}}(m, n) = m + n$

   Então $\langle \mathbb{P}, \Sigma_{\mathbb{P}}\rangle$ é uma Σ-álgebra.

   Seja $dobro : \mathbb{N} \longrightarrow \mathbb{P}$ a função bijetiva $dobro(n) = 2n$, para todo $n \in \mathbb{N}$. Então dobro é um Σ-homomorfismo. Para mostrar isto precisamos mostrar que (2.1) se verifica com "dobro" no lugar de h para toda função $f \in \Sigma$.

   (a)
   $$\begin{aligned} dobro(zero_{\mathbb{N}}) &= dobro(0) \\ &= 2 \cdot 0 \\ &= 0 \\ &= zero_{\mathbb{P}} \end{aligned}$$

   (b)
   $$\begin{aligned} dobro(suc_{\mathbb{N}}(n)) &= dobro(n+1) \\ &= 2(n+1) \\ &= 2n + 2 \\ &= dobro(n) + 2 \\ &= suc_{\mathbb{P}}(dobro(n)) \end{aligned}$$

(c)

$$\begin{aligned} dobro(soma_{\mathbb{N}}(m,n)) &= dobro(m+n) \\ &= 2(m+n) \\ &= 2m+2n \\ &= dobro(m)+in(n) \\ &= soma_{\mathbb{P}}(dobro(m), dobro(n)) \end{aligned}$$

3. *Seja* $\Sigma = \{zero, soma\}$ *com aridade 0 e 2, respectivamente e sejam* $\langle \wp^{fin}(\mathbb{N}), \{\emptyset, \oplus\}\rangle$ *e* $\langle \mathbb{N}, \{0, +\}\rangle$ *duas* Σ*-álgebras tais que*

   (a) $\wp^{fin}(\mathbb{N}) = \{X \subseteq \mathbb{N} \ / \ X$ *é um conjunto finito*$\}$

   (b) $\oplus : \wp^{fin}(\mathbb{N}) \times \wp^{fin}(\mathbb{N}) \longrightarrow \wp^{fin}(\mathbb{N})$ *é definido por*

   $$\oplus(X,Y) = \{x+y \ / \ x \in X \ \text{ e } \ y \in Y\} \cup X \cup Y$$

   (c) $\emptyset, 0$ *e* $+$ *tem a interpretação usual.*

   Mostraremos que $max : \wp^{fin}(\mathbb{N}) \longrightarrow \mathbb{N}$ *definido por "*$max(X)$ *é o maior elemento do conjunto* $X \cup \{0\}$*" é um* Σ*-homomorfismo.*

   (a) $max(\emptyset) = 0$

   (b) $\begin{aligned} max(\oplus(X,Y)) &= max(\{x+y \ / \ x \in X \ \text{ e } \ y \in Y\} \cup X \cup Y) \\ &= max(X) + max(Y) \end{aligned}$

Proposição 2.3.4 *Sejam* $\langle A, \Sigma_A\rangle$*,* $\langle B, \Sigma_B\rangle$ *e* $\langle C, \Sigma_C\rangle$ Σ*-álgebras.*

1. *Se* $h_1 : A \longrightarrow B$ *e* $h_2 : B \longrightarrow C$ *são* Σ*-homomorfismo, então* $h_2 \circ h_1 : A \longrightarrow C$ *é um* Σ*-homomorfismo.*

2. $i : A \longrightarrow A$ *tal que* $i(x) = x$ *para todo* $x \in A$ *é um* Σ*-homomorfismo.*

Demonstração: (exercício). ■

A mais importante propriedade da Σ-álgebra dos termos $\langle T_\Sigma, \Sigma_{T_\Sigma}\rangle$ é expressa no seguinte teorema.

Teorema 2.3.5 *Para toda* Σ*-álgebra* $\langle A, \Sigma_A\rangle$ *existe um único* Σ*-homomorfismo* $i_A : T_\Sigma \longrightarrow A$*.*

Quando existe um único Σ-homomorfismo de uma Σ-álgebra A a qualquer Σ-álgebra de uma determinada classe $\mathcal{C}$ de Σ-álgebras, dizemos que a Σ-álgebra A é **inicial** na classe $\mathcal{C}$.

A definição de T_Σ é indutiva. T_Σ é o menor conjunto de expressões em Σ^* que contém os símbolos de constantes e é fechado com relação às operações de Σ. A natureza indutiva de T_Σ provê um método de prova para derivar propriedades da linguagem especificada pela assinatura Σ. Para mostrar que uma propriedade P se verifica para todos os elementos em T_Σ é suficiente estabelecer:

1. P se verifica para todos os símbolos de constantes de Σ.
2. Assumindo que P se verifica para $a_1, \ldots, a_n \in T_\Sigma$, prove que P também se verifica para $f(a_1, \ldots, a_n)$, para todo $f \in \Sigma$, $arid(f) = n$.

Isso é chamado indução estrutural, pois a indução é sobre a estrutura dos elementos de T_Σ. é possível, também, usar indução estrutural para se definir relações ou funções sobre T_Σ. Por exemplo, para definir uma função g sobre T_Σ é suficiente:

1. Definir o resultado da aplicação de g aos símbolos de constante.
2. Definir o resultado da aplicação de g a $f(a_1, \ldots, a_n)$ em termos de $g(a_1), \ldots, g(a_n)$, para todo $f \in \Sigma$, de aridade n.

De fato, ambas as condições serão resumidas em uma única:

Assumindo o resultado da aplicação de g aos elementos $a_1, \ldots, a_n$, defina o resultado da aplicação de g a $f(a_1, \ldots, a_n)$, para $f \in \Sigma_{T_\Sigma}$, de aridade $n \geq 0$.

Demonstração: (teorema 2.3.5) Devemos mostrar que o Σ-homomorfismo existe e que ele é único.

1. Definiremos i_A por indução estrutural sobre T_Σ. Todo objeto em T_Σ é da forma $f(a_1, \ldots, a_n)$ para algum $f \in \Sigma$ de aridade n. Assumiremos, por indução, que $i_A(a_1), \ldots, i_A(a_n)$ estão definidos. Então definamos $i_A(f(a_1, \ldots, a_n))$ como sendo $f_A(i_A(a_1), \ldots, i_A(a_n))$. Desse modo, definimos i_A para todo elemento em T_Σ. Direto, da definição de i_A, i_A é um Σ-homomorfismo.
2. Para provarmos que i_A é único provemos que ele coincide com todo Σ-homomorfismo de T_Σ em A. Seja h um qualquer de tais Σ-homomorfismos. Mostremos, por indução estrutural, que $i_A(a) = h(a)$ para todo $a \in T_\Sigma$. Um elemento típico de T_Σ é da forma $f(a_1, \ldots, a_n)$, onde $f \in \Sigma$ tem aridade n. A hipóteses indutiva é que $i_A(a_i) = h(a_i)$, para todo $i = 1, \ldots, n$. Então

$$\begin{array}{lll} i_A(f(a_1,\ldots,a_n)) & = f_A(i_A(a_1),\ldots,i_A(a_n)) & \text{por definição de } i_A \\ & = f_A(h(a_1),\ldots,h(a_n)) & \text{por indução estrutural} \\ & = h(f_{T_\Sigma}(a_1,\ldots,a_n)) & \text{pois } h \text{ é um } \Sigma\text{-homomorfismo} \\ & = h(f(a_1,\ldots,a_n)) & \text{por definição de } f_{T_\Sigma} \end{array}$$

Como todo elemento de T_Σ é da forma $f(a_1,\ldots,a_n)$, para algum símbolo de função f, segue que i_A coincide com h. ■

Vendo T_Σ como a sintaxe de uma linguagem e a Σ-álgebra $\langle A, \Sigma_A\rangle$ como um domínio semântico ou interpretação, este teorema nos diz que todo objeto da linguagem tem um único significado em $\langle A, \Sigma_A\rangle$, isto é, só existe uma maneira de interpretar a linguagem no domínio semântico.

Um Σ-homomorfismo $f : A \longrightarrow B$ diz-se um **Σ-isomorfismo** se, e somente se, ele é uma bijeção. Neste caso dizemos que as Σ-álgebras $\langle A, \Sigma_A\rangle$ e $\langle B, \Sigma_B\rangle$ são isomorfas. Ou seja, são essencialmente as mesmas, no sentido de que elas não podem ser diferenciadas usando Σ. Embora, elas possam ser intrinsecamente diferentes, no sentido que a natureza dos elementos de A e de B podem ser completamente diferentes, do ponto de vista estrutural elas são as mesmas.

Teorema 2.3.6 *Sejam $\langle A, \Sigma_A\rangle$ e $\langle B, \Sigma_B\rangle$ duas Σ-álgebras isomorfas. Então existem dois Σ-homomorfismos, $h_1 : A \longrightarrow B$ e $h_2 : B \longrightarrow A$ tais que*

1. $h_2 \circ h_1 = id_A$[3]
2. $h_1 \circ h_2 = id_B$.

Demonstração: 1) Suponha que $h_2 : B \longrightarrow A$ é um Σ-isomorfismo. Defina a função $h_1 : A \longrightarrow B$ por $h_1 = h_2^{-1}$, isto é a função inversa de h_2. Isto é possível, uma vez que h_2 é uma função bijetiva. Portanto, $h_2 \circ h_1 = id_A$. Assim, só devemos mostrar que h_1 é um Σ-isomorfismo. Seja um $f \in \Sigma$ arbitrário de aridade k, e $a_1,\ldots,a_k \in A$. Então

$$\begin{array}{ll} h_1(f_A(a_1,\ldots,a_k)) & = h_1(f_A(id_A(a_1),\ldots,id_A(a_k))) \\ & = h_1(f_A(h_2 \circ h_1(a_1),\ldots,h_2 \circ h_1(a_k))) \\ & = h_1(f_A(h_2(h_1(a_1)),\ldots,h_2(h_1(a_k)))) \\ & = h_1(h_2(f_B(h_1(a_1),\ldots,h_1(a_k)))) \\ & = h_1 \circ h_2(f_B(h_1(a_1),\ldots,h_1(a_k))) \\ & = id_B(f_B(h_1(a_1),\ldots,h_1(a_k))) \\ & = f_B(h_1(a_1),\ldots,h_1(a_k)) \end{array}$$

[3] id_A é a função identidade no conjunto A. Formalmente. $id_A : A \longrightarrow A$ é definida por $id_A(a) = a$ para todo $a \in A$.

Portanto, h_1 também é um Σ-homomorfismo bijetivo, isto é, um Σ-homomorfismo.

A outra parte desta proposição é análoga à primeira. ■

Como um corolário imediato deste teorema de caracterização temos que as álgebras iniciais são únicas a menos de isomorfismo.

Corolário 2.3.7 *Sejam $\langle A, \Sigma_A\rangle$ e $\langle B, \Sigma_B\rangle$ duas Σ-álgebras iniciais. Então elas são isomorfas.*

DEMONSTRAÇÃO: (Exercício). ■

Seja X um conjunto de variáveis. Usaremos as letras $x, y, z, x_1, x_2, \ldots$ para designar qualquer elemento de X. às variáveis assim introduzidas, quando interpretadas numa Σ-álgebra, lhe são atribuídos valores no conjunto base dessa Σ-álgebra.

Definição 2.3.8 *Seja $\langle A, \Sigma_A\rangle$ uma Σ-álgebra e X um conjunto de variáveis. Uma* **A-atribuição** *de valores para X é uma aplicação $\rho_A : X \longrightarrow A$ tal que associa a cada variável $x \in X$ um elemento $\rho_A(x) \in A$.*

Seja $\Sigma(X) = \Sigma \cup X$. Podemos construir um tipo de Σ-álgebra dos termos, $\langle T_{\Sigma(X)}, \Sigma(X)_{T_\Sigma(X)}\rangle$, que considere estas variáveis, da seguinte forma:

1. $T_{\Sigma(X)}$ é o menor subconjunto de $\Sigma(X)^*$ satisfazendo as propriedades
   (a) $x \in T_{\Sigma(X)}$ para cada $x \in X$.
   (b) Se $a \in \Sigma$ e $arid(a) = 0$ então $a \in T_{\Sigma(X)}$.
   (c) Se $f \in \Sigma$, $arid(f) = n$ e $t_1, \ldots, t_n \in T_{\Sigma(X)}$ então $f(t_1, \ldots, t_n) \in T_{\Sigma(X)}$.
2. Para cada símbolo de função $f \in \Sigma$, se f tem aridade n, defina a função $f_{T_{\Sigma(X)}} : T^n_{\Sigma(X)} \longrightarrow T_{\Sigma(X)}$ por

$$f_{T_{\Sigma(X)}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

   Assim, $\Sigma_{T_{\Sigma(X)}} = \{f_{T_{\Sigma(X)}} \ / \ f \in \Sigma\}$.

Esta Σ-álgebra é diferente da Σ-álgebra dos termos, pois ela contém novos elementos (aqueles que contém variáveis). No entanto, ela é "inicial" em algum sentido.

Proposição 2.3.9 *Se $\langle A, \Sigma_A\rangle$ é uma Σ-álgebra e ρ_A é uma A-atribuição para X então existe um único Σ-homomorfismo $h_A : T_{\Sigma(X)} \longrightarrow A$, tal que $h_A(x) = \rho_A(x)$ para todo $x \in X$.*

DEMONSTRAÇÃO: Defina h_A por indução estrutural sobre $T_{\Sigma(X)}$. Se $t \in X$, então seja $h_A(t) = \rho_A(t)$. Caso contrário t tem a forma $f(t_1, \ldots, t_n)$, para algum $f \in \Sigma$. Neste caso, seja $h_A(f(t_1, \ldots, t_n)) = f_A(h_A(t_1), \ldots, h_A(t_n))$. Assim, h_A é definida para todo $f \in \Sigma_{T_{\Sigma(X)}}$.

A prova de que h_A é um Σ-homomorfismo é análoga à prova de que i_A é um Σ-homomorfismo, no teorema 2.3.5.

Seja $h' : T_{\Sigma(X)} \longrightarrow A$ um outro Σ-homomorfismo que coincide com ρ_A em X. Mostremos por indução estrutural sobre $t \in T_{\Sigma(X)}$ que $h(t) = h'(t)$, para todo t.

1. Se t é uma variável. Então ambas $h(t)$ e $h'(t)$ coincidem com $\rho_A(t)$.
2. Se t tem a forma $f(t_1, \ldots, t_n)$ para algum $f \in \Sigma$. Então

$$\begin{array}{lll} h(f(t_1, \ldots, t_n)) & = f_A(h(t_1), \ldots, h(t_n)) & \text{pois } h \text{ é um } \Sigma\text{-homomorfismo} \\ & = f_A(h'(t_1), \ldots, h'(t_n)) & \text{por indução estrutural} \\ & = h'(f(t_1, \ldots, t_n)) & \text{pois } h' \text{ é um } \Sigma\text{-homomorfismo.} \end{array}$$

Portanto h é único. ■

Observe que esta proposição pode ser vista como uma generalização do teorema 2.3.5. Fazendo $X = \emptyset$, nesta proposição, obtemos o teorema 2.3.5. O que importa, na proposição acima, é que todo objeto com variável em $T_{\Sigma(X)}$ pode ser interpretado de modo único numa Σ-álgebra $\langle A, \Sigma_A \rangle$, uma vez que as variáveis de X sejam associadas a elementos de A.

2.4 Classes Equacionais

Muitas vezes, é desejável obrigar que as interpretações de certos símbolos de função satisfaçam determinadas propriedades. Isto é, queremos obrigar que as Σ-álgebras possuam certas características. Podemos descartar coleções de Σ-álgebras usando equações. Por exemplo, se na assinatura $\Sigma = \{zero, suc, soma\}$ do exemplo 2.2.3 obrigamos às Σ-álgebras satisfazer a equação

$$soma(x, zero) = x$$

A Σ-álgebra $\langle \mathbb{N}, zero_{\mathbb{N}}, suc_{\mathbb{N}}, soma_{\mathbb{N}} \rangle$, do exemplo 2.2.3, sobre o conjunto dos números naturais satisfaz essa equação. Mas a Σ-álgebra $\langle \mathbb{N}, zero_{\mathbb{N}}, suc_{\mathbb{N}}, mult_{\mathbb{N}} \rangle$ não.

Uma **equação** é determinada por dois termos os quais podem conter variáveis. A avaliação desses termos numa Σ-álgebra é com respeito a uma atribuição de valores a essas variáveis.

Definição 2.4.1 *Uma Σ-álgebra* **satisfaz uma equação** *se para cada possível atribuição de valores às variáveis no conjunto base, a avaliação destes dois termos coincide.*

Para determinar a classe de Σ-álgebras que satisfazem um sistema de equações devemos introduzir um tipo de relação de equivalência, chamada de Σ-congruência, entre os membros básicos das Σ-álgebras de tal modo que a relação de equivalência preserve a estrutura. Posteriormente veremos como um sistema de equações gera estas Σ-congruências.

Relações de equivalências são estendidas de modo natural para Σ-álgebras. Uma Σ-congruência é uma relação de equivalência que preserva a estrutura induzida por Σ.

Definição 2.4.2 *Seja $\langle A, \Sigma_A \rangle$ uma Σ-álgebra. Uma relação de equivalência*[4] *C sobre A é uma* **Σ-congruência** *se para cada $f \in \Sigma$ de aridade k, $a = (a_1, \ldots, a_k) \in A^k$ e $b = (b_1, \ldots, b_k) \in A^k$, temos que*

$$\textit{se } (a_i, b_i) \in C \textit{ para cada } i = 1, \ldots, k \textit{ então, } (f_A(a), f_A(b)) \in C.$$

Para qualquer Σ-algebra A existem duas Σ-congruências canônicas, ditas **triviais**, estas são a igualdade ($(x, y) \in C$ se, e somente se, $x = y$) e aquela onde todos os elementos são equivalentes ($(x, y) \in C$ para todo $x, y \in A$).

Seja A uma Σ-álgebra e C uma relação de congruência. Então, para cada $a \in A$, existe o conjunto, $[a]_C$, dos elementos de A que são equivalentes a a, isto é,

$$[a]_C = \{b \in A \ / \ (a, b) \in C\}.$$

Chamaremos este conjunto de **classe de equivalência** de a.

Exemplo 2.4.3 *Seja $\langle \mathbb{N}, zero_{\mathbb{N}}, suc_{\mathbb{N}}, soma_{\mathbb{N}} \rangle$ a Σ-álgebra do exemplo 2.2.3. A relação $C \subseteq \mathbb{N} \times \mathbb{N}$ definida definida por*

$$(x, y) \in C \textit{ se, e somente se, } x \textit{ tem a mesma paridade de } y$$

é trivialmente uma relação de equivalência sobre $\mathbb{N}$. Para demonstrar que é uma Σ-congruência devemos mostrar que preserva as operações.

1. *Se x e y tem a mesma paridade, digamos "ímpar", então $x+1$ e $y+1$ têm a mesma paridade, neste caso "par". Assim,*

$$\textit{se } (x, y) \in C \textit{ então } (suc_{\mathbb{N}}(x), suc_{\mathbb{N}}(y)) \in C.$$

[4] Uma relação de equivalência sobre um conjunto qualquer é uma relação binária que é reflexiva, simétrica e transitiva.

2. *Se x_1 tem a mesma paridade de y_1 e x_2 têm a mesma paridade de y_2, então, trivialmente, $x_1 + x_2$ tem a mesma paridade de $y_1 + y_2$. Assim,*

$$se\ (x_1, y_1), (x_2, y_2) \in C\ então\ (x_1\ soma_{\mathbb{N}}\ x_2, y_1\ soma_{\mathbb{N}}\ y_2) \in C.$$

Teríamos as seguintes classes de equivalências:

- $[0]_C = \{0, 2, 4, 6, \ldots\} = \{2x\ /\ x \in \mathbb{N}\}$
- $[1]_C = \{1, 3, 5, 7, \ldots\} = \{2x + 1\ /\ x \in \mathbb{N}\}$
- $[2]_C = \{0, 2, 4, 6, \ldots\} = \{2x\ /\ x \in \mathbb{N}\}$
- $[3]_C = \{1, 3, 5, 7, \ldots\} = \{2x + 1\ /\ x \in \mathbb{N}\}$
- $\vdots$

Observe que neste caso, $[0]_C = [2]_C = [4]_C = \ldots$ e $[1]_C = [3]_C = [5]_C = \ldots$. Assim, podemos dizer que a relação de congruência C determina, basicamente, duas classes de equivalência: a dos números pares e a dos números ímpares. Portanto, tanto faz representar a classe dos pares, por exemplo, por $[0]_C$ ou por $[2]_C$, pois 0 e 2 são meros representantes da classe dos números pares.

Exemplo 2.4.4 *Seja a Σ-álgebra $\langle \wp^{fin}(\mathbb{N}), \emptyset, \oplus \rangle$ do exemplo 2.3.3. Podemos definir a seguinte Σ-congruência C:*

$$(X, Y) \in C\ \textit{se, e somente se,}\ max(X \cup \{0\}) = max(Y \cup \{0\}).$$

Teríamos as seguintes classes de equivalência:

- $[\{0\}]_C = \{\emptyset, \{0\}\}$
- $[\{1\}]_C = \{\{1\}, \{0, 1\}\}$
- $[\{2\}]_C = \{\{2\}, \{2, 1\}, \{2, 0\}, \{2, 1, 0\}\}$
- $\vdots$

Para demonstrar que a relação de equivalência C é realmente uma Σ-congruência, devemos mostrar que preserva as operações. Se $(X_1, X_2) \in C$ e $(Y_1, Y_2) \in C$ então

$$\begin{aligned} max(\oplus(X_1, Y_1)) &= max(X_1) + max(Y_1) \\ &= max(X_2) + max(Y_2) \\ &= max(\oplus(X_2, Y_2)) \end{aligned}$$

Denotaremos por A/C o **conjunto de classes de equivalências** em A induzidas por C, isto é

$$A/C = \{[a]_C \ / \ a \in A\}.$$

Analogamente, para cada $f \in \Sigma$ de aridade k, podemos definir a função $f_{A/C}$ de aridade k sobre A/C por

$$f_{A/C}([a_1]_C, \ldots, [a_k]_C) = [f_A(a_1, \ldots, a_k)]_C.$$

Denotaremos por $\Sigma_{A/C}$ ao conjunto das funções $f_{A/C}$, isto é

$$\Sigma_{A/C} = \{f_{A/C} \ / \ f \in \Sigma\}.$$

Lema 2.4.5 *Seja $\langle A, \Sigma_A \rangle$ uma Σ-álgebra.*

1. *$\langle A/C, \Sigma_{A/C} \rangle$ é uma Σ-álgebra*
2. *$em : A \longrightarrow A/C$ definido por $em(a) = [a]_C$ é um Σ-homomorfismo.*

Demonstração:

1. Como A/C é um conjunto e $\Sigma_{A/C}$ tem uma "função" $f_{A/C}$ para cada $f \in \Sigma$ com a mesma aridade, poderíamos concluir que $\langle A/C, \Sigma_{A/C} \rangle$ é uma Σ-álgebra. Porém, para isto devemos estar certos que cada $f_{A/C}$ é realmente uma função bem definida. Logo, devemos mostrar que o resultado de $f_{A/C}([a_1]_C, \ldots, [a_n]_C)$ não depende do representante escolhido nos $[a_i]_C$. Seja a_i' em $[a_i]_C$, com $1 \leq i \leq n$. Portanto, $(a_i, a_i') \in C$ para cada $1 \leq i \leq n$. Como C é uma Σ-congruência, segue que $(f_A(a_1, \ldots, a_n), f_A(a_1', \ldots, a_n')) \in C$ e portanto $f_A(a_1', \ldots, a_n') \in [f_A(a_1, \ldots, a_n)]_C$. Logo,

$$f_{A/C}([a_1]_C, \ldots, [a_k]_C) = [f_A(a_1, \ldots, a_k)]_C = [f_A(a_1', \ldots, a_k')]_C.$$

2. A prova de que $em : A \longrightarrow A/C$ é um Σ-homomorfismo segue imediatamente da definição de $f_{A/C}$. ■

Σ-congruências são definidas sobre uma Σ-álgebra específica e não sobre uma classe de Σ-álgebras, o qual vá encontra a idéia de caracterizar uma subclasse das Σ-álgebras. No entanto, se definirmos a Σ-congruência sobre a Σ-álgebra dos termos, $\langle T_\Sigma, \Sigma_{T_\Sigma} \rangle$, podemos, usando a inicialidade da Σ-álgebra dos termos, estender a Σ-congruência para outras Σ-álgebras.

Definição 2.4.6 *Seja C uma Σ-congruência sobre T_Σ. Dizemos que uma* **Σ-álgebra** *A* **satisfaz** *C se $i_A(t) = i_A(s)$ quando $(t, s) \in C$.*

Teorema 2.4.7 *Seja C uma Σ-congruência sobre T_Σ e seja $\mathcal{C}(C)$ a classe de todas as Σ-álgebras satisfazendo C. $\langle T_\Sigma/C, \Sigma_{T_\Sigma/C}\rangle$ é inicial em $\mathcal{C}(C)$.*

Demonstração: Primeiro devemos mostrar que T_Σ/C está de fato em $\mathcal{C}(C)$, depois devemos descrever um Σ-homomorfismo h_A de T_Σ/C em qualquer Σ-álgebra A satisfazendo C. Finalmente devemos mostrar que h_A é único.

1. A injeção natural de T_Σ em T_Σ/C é um Σ-homomorfismo e como $\langle T_\Sigma, \Sigma_{T_\Sigma}\rangle$ é inicial na classe de todas as Σ-álgebras este deve coincidir com $i_{T_\Sigma/C}$. Logo, seja $(t, t') \in C$, então

$$\begin{aligned} i_{T_\Sigma/C}(t) &= [t]_C \\ &= [t']_C \\ &= i_{T_\Sigma/C}(t'). \end{aligned}$$

   Logo, $\langle T_\Sigma/C, \Sigma_{T_\Sigma/C}\rangle$ satisfaz C e portanto $\langle T_\Sigma/C, \Sigma_{T_\Sigma/C}\rangle \in \mathcal{C}(C)$.

2. Seja $\langle A, \Sigma_A\rangle \in \mathcal{C}(C)$. Defina $h_A : T_\Sigma/C \longrightarrow A$ por

$$h_A([t]_C) = i_A(t).$$

   Esta função está bem definida, pois se $t' \in [t]_C$, então $(t, t') \in C$ e como A satisfaz C, $i_A(t) = i_A(t')$. Por outro lado, para todo $f \in \Sigma$ de aridade k e $t \in T_\Sigma^k$ temos que

$$\begin{aligned} h_A(f_{T_\Sigma/C}([t]_C)) &= h_A([f(t)]_C) \\ &= i_A(f(t)) \\ &= f_A(i_A(t)) \\ &= f_A(h_A([t]_C)). \end{aligned}$$

   Assim, h_A é um Σ-homomorfismo.

3. Seja h'_A um outro Σ-homomorfismo de T_Σ/C em A. Defina $i'_A : T_\Sigma \longrightarrow A$ por $i'_A(t) = h'_A([t]_C)$. Então,

$$\begin{aligned} i'_A(f_{T_\Sigma}(t)) &= h'_A([f(t)]_C) \\ &= h'_A(f_{T_\Sigma/C}([t]_C)) \\ &= f_A(h'_A([t]_C)) \\ &= f_A(i'_A(t)). \end{aligned}$$

   Logo, i'_A é um Σ-homomorfismo. Como, T_Σ é inicial na classe das Σ-álgebras, i'_A e i_A coincidem. Portanto,

$$\begin{aligned} h'_A([t]_C) &= i'_A(t) \\ &= i_A(t) \\ &= h_A([t]_C). \end{aligned}$$

Logo, h_A e h'_A são os mesmos Σ-homomorfismos e portanto h_A é único. ■

Observe que temos duas Σ-álgebras iniciais diferentes. Uma, $\langle T_\Sigma, \Sigma_{T_\Sigma}\rangle$, é inicial com respeito à classe de todas as Σ-álgebras enquanto que a outra, $\langle T_\Sigma/C, \Sigma_{T_\Sigma/C}\rangle$, é inicial com respeito a um subconjunto delas, a classe de todas as Σ-álgebras satisfazendo a Σ-congruência C.

Estamos interessados num tipo especial de Σ-congruência, aquelas obtidas a partir de um conjunto ou sistema de equações. Para definir formalmente como equações geram Σ-congruências, necessitamos introduzir variáveis à assinatura como no final da seção 1.3.

Uma **Σ-equação** é uma par $(t, s) \in T_{\Sigma(X)} \times T_{\Sigma(X)}$. Usualmente escritas por $t = s$.

Definição 2.4.8 *Seja $\langle A, \Sigma_A\rangle$ uma Σ-álgebra e E um conjunto de Σ-equações com variáveis em X. A Σ-álgebra $\langle A, \Sigma_A\rangle$* **satisfaz** *$E$ se, para cada Σ-equação (t, s) em E e cada A-atribuição ρ_A, $h_A(t) = h_A(s)$, onde h_A é o único Σ-homomorfismo de $T_{\Sigma(X)}$ em A tal que $h_A(x) = \rho_A(x)$ para todo $x \in X$ (proposição 2.3.9).*

Exemplo 2.4.9 *Seja a assinatura $\langle \Sigma, arid\rangle$, onde $\Sigma = \{soma, menos, zero\}$ e $arid(soma) = 2$, $arid(menos) = 1$ e $arid(zero) = 0$. Considere $X = \{x, y, z\}$ e o seguinte conjunto E de Σ-equações com variáveis em X:*

1. $soma(x, soma(y, z)) = soma(soma(x, y), z)$
2. $soma(zero, x) = soma(x, zero)$
3. $soma(x, zero) = x$
4. $soma(x, menos(x)) = soma(menos(x), x)$
5. $soma(x, menos(x)) = zero$

Note que toda Σ-álgebra satisfazendo este conjunto E de Σ-equações é um grupo.

Teorema 2.4.10 *Seja E um conjunto de Σ-equações e $\mathcal{C}(E)$ a classe de Σ-álgebras satisfazendo E. $\mathcal{C}(E)$ tem uma Σ-álgebra inicial.*

Demonstração: Aqui só daremos o esboço da demonstração. A Σ-álgebra inicial de $\mathcal{C}(E)$ é T_Σ/C, onde C é a Σ-congruência definida por

$(t, s) \in C$ se e somente se para alguma Σ-equação $(t', s') \in E$ existe uma $T_{\Sigma(X)}$-atribuição tal que $h_{T_\Sigma}(t') = t$ e $h_{T_\Sigma}(s') = s$, onde $h_{T_\Sigma} : T_{\Sigma(X)} \longrightarrow T_\Sigma$ é o único Σ-homomorfismo tal que $h_{T_\Sigma}(x) = \rho_{T_\Sigma(x)}$ para cada $x \in X$. ■

2.5 Σ-Domínios

Podemos estender a noção de assinatura por considerar além de símbolos funcionais, símbolos relacionais, isto é, símbolos cujas interpretações sejam relações sobre o conjunto base.

Definição 2.5.1 *Uma* **assinatura relacional** *é uma tripla* $\langle \Sigma_F, \Sigma_R, arid \rangle$*, tal que* Σ_F *e* Σ_R *são conjuntos contáveis de símbolos funcionais e de predicados, respectivamente e* $arid$ *é uma função de aridade,* $arid : \Sigma_F \cup \Sigma_R \longrightarrow \mathbb{N}$*, tal que a cada símbolo* $s \in \Sigma_F \cup \Sigma_R$ *associa sua aridade.*

Definição 2.5.2 *Um* **Σ-domínio** *é um par* $\langle D, \Sigma_D \rangle$*, onde*

- D *é um conjunto não vazio, chamado* **conjunto base**.
- $\Sigma_D = \{f_1^D, f_2^D, \ldots, R_1^D, R_2^D, \ldots\}$ *é um conjunto de funções e relações que interpretam os símbolos correspondentes em* Σ_F *e* Σ_R*, respectivamente, de tal modo que se* $arid(f_i) = n$*, em* Σ_F*, então* $f_i^D : D^n \longrightarrow D$ *é uma função de* D^n *em* D*, ou seja, é uma operação n-ária em* D *(as funções de aridade zero são as constantes). Se* $arid(R_i) = n$*, em* Σ_R*, então* R_i^D *é uma relação n-ária em* D*, isto é* $R_i^D \subseteq D^n$*, que interpreta* R_i*.*

Exemplo 2.5.3 *Considere a assinatura relacional* $\langle \Sigma_d = \Sigma_F \cup \Sigma_R, arid \rangle$*, onde* $\Sigma_F = \{zero, um, soma, div\}$*,* $\Sigma_R = \{igual\}$ *e* $arid(zero) = 0$*,* $arid(um) = 0$*,* $arid(soma) = 2$*,* $arid(div) = 2$ *e* $arid(igual) = 2$*. Seja* $=_{\mathbb{N}}$ *a igualdade usual sobre os números naturais,* $\mathbb{N}$ *e* $0_{\mathbb{N}}$*,* $1_{\mathbb{N}}$*,* $+_{\mathbb{N}}$ *e* $/_{\mathbb{N}}$*, o número zero, o número um, a soma e a divisão sobre os números naturais, respectivamente.*

Então, $\langle \mathbb{N}, +_{\mathbb{N}}, 0_{\mathbb{N}}, 1_{\mathbb{N}}, /_{\mathbb{N}}, =_{\mathbb{N}} \rangle$ *é um* Σ_d*-domínio que interpreta* Σ_d *em* $\mathbb{N}$*.*

A teoria dos Σ-domínios pode ser desenvolvida do mesmo modo que a teoria das Σ-álgebras, com as modificações óbvias, que consistem em considerar, os símbolos de relação na assinatura Σ, os quais são interpretados, em D, como relações. Por exemplo, um Σ-domínio homomorfismo é um Σ-homomorfismo que preserva a estrutura extra de relações. Sejam $\langle D_1, \Sigma_1 \rangle$ e $\langle D_2, \Sigma_2 \rangle$ Σ-domínios. Uma função $h : D_1 \longrightarrow D_2$ é um **Σ-domínio homomorfismo** se:

1. Para cada $f \in \Sigma$, $h(f_{D_1}(d_1, \ldots, d_n)) = f_{D_2}(h(d_1), \ldots, h(d_n))$, onde $d_1, \ldots, d_n \in D_1$ e $arid(f) = n$.
2. Se $(d_1, \ldots, d_n) \in R_i^{D_1}$, então $(h(d_1), \ldots, h(d_n)) \in R_i^{D_2}$, onde $arid(R_i) = n$ e $R_i^{D_1}, R_i^{D_2}$ denotam relações n-árias sobre D_1 e D_2, respectivamente.

Um Σ-domínio homomorfismo $h : D_1 \longrightarrow D_2$ diz-se um **isomorfismo**, mais precisamente um **Σ-isomorfismo**, se h, além de ser um Σ-domínio homomorfismo, for bijetiva. Neste caso dizemos que D_1 e D_2 são isomorfos.

Analogamente a Σ-álgebras, também temos um **Σ-domínio dos termos com variáveis** em X, $\langle T_{\Sigma(X)}, \Sigma_{T_{\Sigma(X)}} \rangle$. Como as relações tomam, quando interpretadas num Σ-domínio, valores verdade e não elementos do domínio básico, elas não constituem um termo. Assim, termos de uma assinatura Σ_d são compostos de variáveis em X e funções em Σ_F. Logo, $T_\Sigma(X)$ é definido da seguinte maneira:

1. se $x \in X$ então $x \in T_{\Sigma(X)}$.
2. se $f \in \Sigma_F$, $arid(f) = n$ e $t_1, \ldots, t_n \in T_{\Sigma(X)}$ então $f(t_1, \ldots, t_n) \in T_{\Sigma(X)}$.

Por outro lado, $\Sigma_{T_{\Sigma(X)}}$ é definido da seguinte maneira:

1. se $f \in \Sigma_F$, $arid(f) = n$ e $t_1, \ldots, t_n \in T_{\Sigma(X)}$ então $f_{T_{\Sigma(X)}} : T^n_{\Sigma(X)} \longrightarrow T_{\Sigma(X)}$ é definida por:

$$f_{T_{\Sigma(X)}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

2. se $R \in \Sigma_R$ e $arid(R) = n$ então $R_{T_\Sigma(X)} = T^n_{\Sigma(X)}$.

Proposição 2.5.4 *Seja $\langle D, \Sigma_D \rangle$ um Σ-domínio e ρ_D uma D-atribuição para X, em Σ. Então existe um único Σ-domínio homomorfismo $h_D : T_{\Sigma(X)} \longrightarrow D$ tal que $h_D(x) = \rho_D(x)$, para todo $x \in X$.*

Demonstração: A prova é virtualmente idêntica àquela do teorema 2.3.5. Seja h_D como definido naquele teorema. Neste teorema foi mostrado que h_D é um Σ-homomorfismo. Com a única relação sobre $T_{\Sigma(X)}$ é a relação trivial, isto é, a igualdade sintática, onde dois termos são iguais se e somente se são idênticos, h_D, trivialmente, preserva essa relação sobre D. Seja h'_D um outro Σ-domínio homomorfismo de $T_{\Sigma(X)}$ em D, que coincide com ρ_D sobre X. Então h'_D é, também, um Σ-domínio homomorfismo da Σ-álgebra $T_{\Sigma(X)}$ em D. Logo pelo teorema 2.3.5 h'_D coincide com h_D. ■

O Σ-domínio homomorfismo h_D, nesta proposição coincide com aquele do teorema 2.3.5, quando o Σ-domínio $\langle D, \Sigma_D \rangle$ é apenas uma Σ-álgebra. Se $X = \emptyset$, existe um único Σ-domínio homomorfismo de T_Σ em D. Caso $X \neq \emptyset$, existe uma infinidade de tais Σ-domínio homomorfismos, um para cada $\rho_D : X \longrightarrow D$.

2.6 Exercícios

1. Defina uma linguagem formal $\langle \Sigma, \mathcal{G} \rangle$ para os números binários. Considere somente aquelas cadeias de 1's e 0's que comecem com 1.

2. Defina uma linguagem formal que gere a linguagem de todas as cadeias, no alfabeto $\{a, b, c\}$, que não são palíndromos.

3. Defina uma linguagem formal $\langle \Sigma, \mathcal{G} \rangle$ que gere a linguagem de todas expressões sobre o alfabeto $\Sigma = \{(,)\}$ que estejam corretas com a convenção do uso de parênteses.

4. Dê uma prova de que $(()())((()()))$ faz parte da linguagem do exemplo anterior.

5. Defina uma linguagem formal $\langle \Sigma, \mathcal{G} \rangle$ para a linguagem de todas as expressões aritméticas, construídas a partir do símbolo E, os operadores binários +,-,* e /. Considere o uso de todos os parênteses.

6. Dê uma prova de que $((E + (E + E)) * ((E + E) * E))$ faz parte da linguagem do exemplo anterior.

7. Descreva as linguagens dos exercícios 1,2 e 5 como assinaturas e dê duas Σ-álgebras para cada uma delas.

8. Dê uma outra Σ-álgebra que tenha como conjunto base o conjunto dos números naturais e operações usuais sobre ele.

9. Dê um Σ-homomorfismo entre as duas Σ-álgebras do exercício 6.

10. Sejam $\langle \wp^{fin}(\mathbb{N}), \{\emptyset, \oplus\} \rangle$ e $\langle \mathbb{N}, \{0, +\} \rangle$ as duas Σ-álgebras do exemplo 2.3.3.3.

    Mostre que $z : \wp^{fin}(\mathbb{N}) \longrightarrow \mathbb{N}$ definido por $z(x) = 0$ é um Σ-homomorfismo. A função $Som : \wp^{fin}(\mathbb{N}) \longrightarrow \mathbb{N}$ definida por

    $$Som(X) = \begin{cases} 0 & \text{se } X = \emptyset \\ x + Som(Y) & \text{se } X = \{x\} \cup Y \end{cases}$$

    é um Σ-homomorfismo?

11. Sejam $\langle \mathbb{P}, \Sigma_{\mathbb{P}} \rangle$, $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}} \rangle$ e $\langle \mathbb{N}, \Sigma_{\mathbb{N}} \rangle$ as Σ-álgebras do exemplo 2.3.3. Demonstre que:

    (a) a função $p_1 : \mathbb{P} \longrightarrow \mathbb{N}$ definida por $p_1(x) = \frac{x}{2}$ é um Σ-homomorfismo.

    (b) a função $p_2 : \mathbb{Z} \longrightarrow \mathbb{N}$, definidas por

    $$p_2(x) = \begin{cases} 2\,|x| - 1 & \text{, se } x < 0 \\ 2x & \text{, se } x \geq 0 \end{cases}$$

    não é um Σ-homomorfismo.

12. Que poderia ser mudado na Σ-álgebra $\langle \mathbb{N}, \Sigma_{\mathbb{N}} \rangle$ para que p_2, do exercício anterior, seja um Σ-homomorfismo.

13. Dê um exemplo de duas Σ-álgebras para as quais exista mais de um Σ-homomorfismo.

14. Dê a Σ-álgebra dos termos da seguinte assinatura (descreva pelo menos 10 termos de T_Σ).

    - $\Sigma = \{zero, suc, pred, plus, prod\}$
    - $arid(zero) = 0$, $arid(suc) = 1$, $arid(pred) = 1$, $arid(soma) = 2$ e $arid(prod) = 2$

15. Seja $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}} \rangle$ onde

    - $zero_{\mathbb{Z}} = 0$
    - $suc_{\mathbb{Z}} : \mathbb{Z} \longrightarrow \mathbb{Z}$ é definida por $suc_{\mathbb{Z}}(x) = x + 1$
    - $pred_{\mathbb{Z}} : \mathbb{Z} \longrightarrow \mathbb{Z}$ é definida por $pred_{\mathbb{Z}}(x) = x - 1$
    - $soma_{\mathbb{Z}} : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$ é definida por $soma_{\mathbb{Z}}(x, y) = x + y$
    - $prod_{\mathbb{Z}} : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$ é definida por $prod_{\mathbb{Z}}(x, y) = xy$.

    Dê o único Σ-homomorfismo de $\langle T_\Sigma, \Sigma_{T_\Sigma} \rangle$ em $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}} \rangle$.

16. Demonstre que $\langle T_\Sigma, \Sigma_{T_\Sigma} \rangle$ e $\langle \mathbb{Z}, \Sigma_{\mathbb{Z}} \rangle$ do exercício anterior são isomorfas.

17. Seja $\langle \mathbb{N}, \{zero_{\mathbb{N}}, suc_{\mathbb{N}}, soma_{\mathbb{N}}\} \rangle$ a Σ-álgebra do exemplo 2.2.3. Mostre que a relação $Mod5 \subseteq \mathbb{N} \times \mathbb{N}$ definida por

    $$(x, y) \in Mod5 \text{ se, e somente se, } x \; mod \; 5 = y \; mod \; 5,$$

    onde $x \; mod \; 5$ é o resto da divisão de x por 5, é uma relação de congruência.

18. Seja $\langle \wp^{fin}(\mathbb{N}), \emptyset, \oplus \rangle$ a Σ-álgebra do exemplo 2.3.3. Mostre que a relação

    $$(X, Y) \in C \text{ se, e somente se, } Card(X) = Card(Y)$$

    onde $Card(X)$ é a cardinalidade do conjunto X, é uma relação de equivalência, porém não é uma Σ-congruência.

19. Defina uma Σ-congruência não trivial para a Σ-álgebra anterior.

20. Seja $\Sigma = \{+, *, 1, 0\}$ com aridades 2, 2, 0 e 0 respectivamente. Defina um conjunto de Σ-equações E de tal modo que as Σ-álgebras satisfazendo E sejam a classe de todos os anéis.

21. Seja E o conjunto de Σ-equações do exercício anterior. Descreva duas Σ-álgebras em $\mathcal{C}(E)$ e mostre um Σ-homomorfismo entre elas. Alguma delas é inicial em $\mathcal{C}(E)$?

22. Seja a assinatura relacional $\Sigma = \{0, 1, +, *, \leq\}$, onde 0 e 1 são constantes, $+$, e $*$ são símbolos de função de aridade 2, e $\leq$ é um símbolo de relação binário. Sejam os seguintes Σ-domínios:

    (a) Σ-domínio dos pares ordenados onde o primeiro elemento é menor ou igual ao segundo

        i. Conjunto base: $\mathbb{I}(\mathbb{R}) = \{(a, b) \ / \ a, b \in \mathbb{R} \text{ e } a \leq_{\mathbb{R}} b\}$, onde $\mathbb{R}$ é o conjunto dos números reais e $\leq_{\mathbb{R}}$ é a ordem usual sobre esse conjunto.
        ii. $0_{\mathbb{I}(\mathbb{R})} = (0, 0)$,
        iii. $1_{\mathbb{I}(\mathbb{R})} = (1, 1)$,
        iv. $(a, b) +_{\mathbb{I}(\mathbb{R})} (c, d) = (a +_{\mathbb{R}} c, b +_{\mathbb{R}} d)$,
        v. $(a, b) *_{\mathbb{I}(\mathbb{R})} (c, d) = (min(ac, ad, bc, bd), max(ac, ad, bc, bd))$ e
        vi. $(a, b) \leq_{\mathbb{I}(\mathbb{R})} (c, d)$ se, e somente se, $c \leq_{\mathbb{R}} a$ e $b \leq_{\mathbb{R}} d$.

    (b) Σ-domínio dos intervalos reais fechados. Um intervalo real fechado é um conjunto $[a, b] = \{x \in \mathbb{R} \ / \ a \leq_{\mathbb{R}} x \leq_{\mathbb{R}} b\}$.

        i. Conjunto base: $\mathbb{IR} = \{X \subseteq \mathbb{R} \ / \ X = [a, b]$ para algum intervalo real fechado $[a, b]\}$,
        ii. $0_{\mathbb{IR}} = [0, 0] = \{0\}$,
        iii. $1_{\mathbb{IR}} = [1, 1] = \{1\}$,
        iv. $[a, b] +_{\mathbb{IR}} [c, d] = \{x + y \ / \ x \in [a, b] \text{ e } y \in [c, d]\}$,
        v. $[a, b] *_{\mathbb{IR}} [c, d] = \{x * y \ / \ x \in [a, b] \text{ e } y \in [c, d]\}$, e
        vi. $[a, b] \leq_{\mathbb{IR}} [c, d]$ se, e somente se, $[a, b] \subseteq [c, d]$

    Mostre que estes dois Σ-domínios são isomorfos.

Capítulo 3

Lógica Proposicional: Linguagem e Semântica

As linguagens são expressões simbólicas de entidades significativas de um "fragmento da realidade" a qual suas expressões representam ou denotam. Desse modo, as linguagens são constituídas de dimensões sintáticas e semânticas. Existe, ainda, uma terceira dimensão nos sistemas lingüísticos, qual seja, a dimensão pragmática, que trata da questão do uso das linguagens por uma comunidade de indivíduos. Do ponto de vista matemático as duas primeiras dimensões estão relativamente avançadas, enquanto a dimensão pragmática ainda se encontra num estágio primitivo de seu desenvolvimento. Ultimamente, com o advento da ciência da computação esta dimensão tem-se tornado uma área de intensa pesquisa. Neste trabalho somente abordaremos os níveis sintáticos e semânticos das linguagens. Mais precisamente, usaremos essas duas dimensões para formalizar a teoria da lógica proposicional. Portanto, no capítulo que segue, a realidade para nós é constituída de proposições, isto é afirmações que podem ser verdadeiras ou falsas.

Quando argumentamos, isto é, quando fazemos uma prova, o fazemos numa linguagem. Se a linguagem está formalizada os argumentos podem, também, ser formalizados, resultando, assim, uma lógica formal em contrapartida a uma lógica especulativa ou informal. O argumento, objeto de estudo da lógica clássica, é uma entidade composta de entidades mais simples chamadas proposições, isto é, frases que são declarativas e conotam um valor verdade, verdadeiro ou falso. De ora em diante só consideraremos frases que são proposições, portanto não consideraremos como proposições frases interrogativas, exclamativas, etc.

Para entender melhor o que é uma proposição considere a frase "1 mais 1 é igual a 10", ou simbolicamente, "1+1 = 10". Esta frase é uma proposição no sentido de que ela é uma asserção declarativa, ou seja, afirma ou nega um fato, e tem um valor verdade, que pode ser verdadeiro ou falso. Neste caso, num sistema de numeração de base 2 a proposição anterior seria verdadeira, enquanto que no sistema decimal seria falsa. Um outro exemplo é a afirmação "hoje é um dia quente", cujo valor verdade vai depender de vários fatores: o local sobre o qual implicitamente se esta falando (pois um dia quente em Natal é muito

diferente de um dia quente em Punta Arenas-Chile), os instrumentos de medidas (esta afirmação foi baseada numa percepção natural ou numa medição com um termômetro) e de comparação (quais os dados estatísticos de temperatura dessa região), e principalmente de quem esta avaliando (Duas pessoas, mesmo considerando as mesmas condições nos items anteriores, podem avaliar diferente). Ou seja, valor verdade de uma proposição não é um conceito absoluto, mas depende de um contexto interpretativo. Inclusive há proposições, que mesmo num contexto interpretativo claro e não ambíguo, para as quais não é possível estabelecer de forma inquestionável sua veracidade ou falsidade (pelo menos com o conhecimento atual da humanidade), por exemplo a conjectura de que a classe dos problemas não determinísticos polinomiais (**NP**) é igual à classe dos problemas determinísticos polinomiais (**P**) é uma afirmação da qual não conhecemos se é verdadeira ou falsa. Mas, em lógica o importante não é o valor verdade que uma proposição possa tomar num determinado contexto interpretativo, mas a possibilidade de que "em princípio" seja possível atribuir um valor verdade, e que seja possível raciocinar com estas proposições.

A lógica proposicional estuda **como** raciocinar com afirmações que podem ser verdadeiras ou falsas, isto é **como** deduzir de um certo conjunto de hipóteses (proposições verdadeiras num determinado contexto) uma prova de que uma determinada conclusão é verdadeira no mesmo contexto. Assim, são fundamentais as noções de proposição, verdade, dedução e prova. A lógica proposicional clássica é um dos exemplos mais simples de lógica formal. Esta lógica leva em conta, somente, os valores verdades (verdadeiro ou falso) e a forma das proposições. O estudo detalhado dessa lógica, é importante porque ela contém quase todos os conceitos importantes necessários para o estudo de lógicas mais complexas.

3.1 A Linguagem da Lógica Proposicional

Uma proposição é um sentença declarativa sobre objetos a qual pode assumir valores verdade Verdadeiro (1) ou Falso (0), dependendo da interpretação. Assim, por exemplo, as sentenças:

1. "Maria gosta de João e de Pedro"
2. "Todos os seres humanos têm uma mãe"
3. "Cinco é maior que quatro"

são, dependendo da interpretação (quem é Maria?, quem é João?, que significa gosta?, etc.), verdadeiras ou falsas.

Os objetos aos quais alude uma proposição (Maria, João, seres humanos, mãe, cinco, um filme, etc.) são chamados de termos.

Conectivos

é possível juntar ou conectar algumas proposições formando proposições mais complexas: Por exemplo, a proposição "Maria gosta de João e é irmã de Pedro" é a junção, mediante o conectivo "e", das proposições "Maria gosta de João" e "Maria é irmão de Pedro".

Em lógica proposicional distinguimos 5 conectivos ou juntores:

Negação

Em português, a palavra "Não", indica que algo (ao que se refer o "não") não ocorre ou não é verdade. Por exemplo, a proposição "Maria não gosta de João" está dizendo que não é verdade que "Maria gosta de João". Assim, a negação de um proposição é verdadeira se a proposição que está sendo negada é falsa. Em lógica proposicional, usaremos o símbolo "$\neg$" para indicar este conectivo lógico.

Conjunção

Em português freqüentemente é usado um "e" como conectivo de duas proposições. Este conectivo é conhecido como conjunção. Por exemplo, a conjunção das proposições "Maria gosta de João" e "João gosta de Maria" é a proposição "Maria gosta de João e João gosta de Maria" o qual poderia ser abreviado para "Maria e João se gostam". Em lógica proposicional, usaremos o símbolo $\wedge$ para indicar a conjunção de proposições. Assim, a partir de duas proposições, p_1 e p_2, podemos obter uma terceira proposição, $p_1 \wedge p_2$, a qual será verdadeira se, e somente se, as duas proposições (p_1 e p_2) são em conjunto ambas verdadeiras.

Disjunção

Em português também com freqüência é usando um "ou" como conectivo de duas proposições. Este conectivo é chamado de disjunção. Por exemplo, a disjunção das proposições "Maria gosta de João" e "João gosta de Maria" é a proposição "Maria gosta de João ou João gosta de Maria". Em lógica proposicional, usaremos o símbolo $\vee$" para indicar a disjunção de proposições. A partir de duas proposições, p_1 e p_2, podemos obter uma terceira proposição, $p_1 \vee p_2$, a qual vai ser verdadeira se, e somente se, ao menos uma das duas proposições p_1 ou p_2) for verdadeira.

Condicional ou implicação

Em português a expressão "se ... então ...." ou a palavra "implica" são usadas para conectar duas proposições. Este conectivo é chamado de condicional ou implicação. Por exemplo, a implicação das proposições "Maria e João são irmãos" e "João e Maria são

parentes" é a proposição "Se Maria e João são irmãos então João e Maria são parentes". Claramente, na vida real esta implicação é correta (verdadeira), mesmo sem sabermos quem são Maria e João. Assim, se de fato Maria e João forem irmãos então necessariamente eles serão parentes. Porém se Maria e João não forem irmãos nada podemos afirmar do parentesco deles. Outro exemplo, de implicação de duas proposições é a seguinte: "cachorro que late, não morde" a qual pode ser re-escrita, de tal modo a salientar o conectivo usado, como "Se o cachorro late, então ele não morde". Em lógica proposicional, usaremos o símbolo $\rightarrow$" para indicar a implicação de proposições. Assim, a partir de duas proposições, p_1 e p_2, podemos obter uma terceira proposição, $p_1 \rightarrow p_2$, a qual vai ser verdadeira se, e somente se, a proposição p_1 é falsa ou p_2 é verdadeira. Ela é chamada de condicional, pois para ela tomar o valor verdade verdadeiro, o valor verdade verdadeiro de p_2 deve estar condicionado ao valor verdade verdadeiro de p_1. Ou seja, toda vez que a proposição p_1 for verdadeira a proposição p_2 também deve ser verdadeira, já quando p_1 for falsa, p_2 pode tomar qualquer valor verdade.

A proposição à esquerda do conectivo $\rightarrow$ é denominada de premissa ou antecedente , e a proposição à direita do conectivo $\rightarrow$ é denominada de conclusão ou conseqüente .

Observe, que nada impede que o conseqüente de uma implicação não tenha relação com o antecedente. Por exemplo, a proposição "Se Maria gosta de João então 1+1=2" é uma implicação desta natureza.

Bi-condicional ou bi-implicação

Em português, a expressão "se, e somente se," é usada também para conectar duas proposições. Este conectivo é conhecido como bi-implicação. Por exemplo, a frase "Josué vai se formar se, e somente se, defender sua monografia" pode ser encarada como a bi-implicação de "Josué vai se formar" com "Josué vai defender sua monografia". Em lógica proposicional, usaremos o símbolo $\leftrightarrow$" para indicar a bi-implicação de proposições.

Assim, a partir de duas proposições, p_1 e p_2, podemos obter uma terceira proposição, $p_1 \leftrightarrow p_2$, a qual vai ser verdadeira se, e somente se, ambas proposições (p_1 e p_2) tiverem o mesmo valor verdade. Ela é chamada de bi-implicação pois ela é equivalente à implicação de p_1 com p_2 e de p_2 com p_1.

Proposições atômicas

Proposições atômicas são proposições, como a palavra diz, indivisíveis, isto é, proposições que não podem ser quebradas em proposições mais simples. Portanto, uma proposição é atômica se ela não possui nenhum conectivo lógico. Por exemplo, as proposições "Rosa gosta de estudar lógica", "todos os corvos são pretos" e "Ronaldo gosta de jogar bola" são proposições atômicas, já as proposições "Rosa gosta de estudar lógica e teoria da computação", "Maria gosta de João ou de Pedro", "Ele não gosta de fumar", "Se Roberto não estuda para a prova de lógica, tirará uma nota baixa", etc. são proposições não atômicas

ou proposições compostas, pois elas envolvem algum conectivo lógico sobre proposições menores.

Uma maneira semi-formal de escrever proposições atômicas é colocar o verbo da proposição, seguido de uma lista de sujeitos entre parênteses. Por exemplo, as proposições "João é pai de Ana" e "Ana, maria e Rosa são irmãs" podem ser colocadas como: Pai(João,Ana) e Irmãs(Ana,Maria,Rosa), respectivamente. Usando esta convenção, podemos descrever afirmações do dia a dia numa nomenclatura mais facilmente compreensível. Por exemplo a afirmação "Se João e Gabriela são os pais de Ana e Ana é irmã de Maria, então João é pai de Maria ou Gabriela é mãe de Maria", pode ser re-escrita numa linguagem mais lógica da seguinte forma:

$$(\text{Pais(João,Gabriela,Ana)} \wedge \text{Irmão(Ana,Maria)}) \rightarrow (\text{Pai(João,Maria)} \vee \text{Mãe(Gabriela,Maria)}),$$

ou ainda

$$(\text{Pai(João,Ana)} \wedge \text{Mãe(Gabriela,Ana)} \wedge \text{Irmão(Ana,Maria)}) \rightarrow (\text{Pai(João,Maria)} \vee \text{Mãe(Gabriela,Maria)}),$$

Esta forma de escrever afirmações é a base para usar lógica como uma forma de representar o conhecimento em sistemas inteligentes, para especificar sistemas, para escrever programas baseados no paradigma de programação em lógica que veremos sucintamente no capítulo 9 deste texto, etc. No entanto, como por enquanto estamos preocupados em lidar com as características e propriedades da linguagem proposicional independente de como são interpretadas as sentenças num contexto real, daremos a seguir a linguagem lógica desde uma perspectiva formal.

3.1.1 Sutilezas com o uso de conectivos em linguagem natural

Existem alguns termos que claramente são o oposto de outro, e que por isso podem ser considerados como sua negação: por exemplo, a proposição "José está desocupado" pode ser interpretada claramente como "José não está ocupado". Porém, a proposição "Pedro está em desvantagem" não é o mesmo que dizer que "Pedro não está em vantagem", pois a segunda admite a possibilidade de Pedro estar empatado.

Como não é possível trabalhar e descansar ao mesmo tempo, a proposição "Maria está trabalhando ou descansando", caso seja verdadeira, significa que ou Maria está trabalhando ou Maria está descansando, porém não ambos. Já em outras situações o "ou" pode deixar aberta a possibilidade de ambos serem verdadeiras ao mesmo tempo, por exemplo a proposição "Maria está correndo ou escutando música", caso verdadeira, significa que Maria está correndo ou Maria está escutando música ou ambos. O primeiro caso, é chamado de ou exclusivo e o segundo é a disjunção como vista aqui. O ou exclusivo,

usualmente denotado pelo símbolo $\oplus$, pode ser obtido a partir da disjunção, negação e conjunção como segue: $p \oplus q \stackrel{\text{def}}{=} p \vee q \wedge \neg(p \wedge q)$.

Devemos ter certos cuidados no uso corriqueiro do conectivo "e" quando tentemos ver ele como um conectivo lógico. Por exemplo, as proposições "*ABC* perdeu para ao *América* e desceu para a segunda divisão" e "*ABC* desceu para a segunda divisão e perdeu para o *America*" são diferentes, pois se indagarmos: *ABC* quando jogou com *América* já tinha sido rebaixado? na primeira proposição fica implícito que ele desceu por causa da derrota com *América*, enquanto que pela segunda proposição fica implícito que ao momento de jogar com a *América*, o *ABC* já estava descendido. Isto se deve a que em linguagem corriqueira (natural) as vezes o "e" traz consigo uma certa ordem temporal Neste casos, seria aconselhável ver a primeira como uma implicação enquanto que a segunda deve ser acrescido o caráter temporal da afirmação.

E como interpretar a locução: "João gosta de Maria mesmo que ela goste de Pedro"?.

Esta locução pode implicar que João gosta de Maria independente de Maria gostar de Pedro ou não, e portanto seria equivalente à sentença: "João gosta de Maria", mas isto deixaria de lado a informação que "Maria gosta de Pedro".

Por outro lado, esta sentença está afirmando que Maria gosta de Pedro e que mesmo assim, João gosta dela, e portanto poderia ser equivalente à sentença: "João gosta de Maria e Maria gosta de Pedro", mas isto desconsideraria o fato implícito na sentença de que João gosta de Maria mesmo que lhe doa o fato de Maria gostar de Pedro, ou seja retira a dependência dada pelo palavra "mesmo que " existente entre as proposições atômicas "João gosta de Maria" e "Maria gosta de Pedro".

Esta sentença também poderia ser entendida como equivalente a "Se Maria gosta de Pedro, então João gosta de Maria", e portanto neste caso se for falso que Maria goste de Pedro, nada poderemos dizer sobre se João gosta de Maria ou não, o que parece não ser a intenção desta sentença, pois nas entrelinhas a sentença original estaria dizendo que João gosta de Maria, gostando ou não Maria de Pedro. Portanto, uma re-formulação mais razoável parece ser "João gosta de Maria e não é o caso que se Maria gosta de Pedro, então João não gosta de Maria".

O problema que este tipo de sentença tem várias conotações que podem ser associadas e que portanto qualquer forma de tratar via conectivos lógicos só conseguiria capturar só um de tais aspectos. Assim a Lógica Clássica não consegue refletir em todas as suas dimensões locuções carregadas com certos aspectos implícitos nela, como tempo ou modalidades, as quais seriam melhor tratadas se usando lógicas temporais e modais. Assim, a lógica clássica não é capaz de refletir todos os argumentos válidos na linguagem usual, mas certamente ela é capaz de representar adequadamente diversas instâncias freqüentes nos raciocínios.

3.2 A Linguagem Formal da Lógica Proposicional

De ora em diante usaremos as letras minúsculas indexadas, $p_1, p_2, \ldots$, para representar proposições atômicas e usaremos letras gregas minúsculas, talvez indexadas, tais como α, β, α_1, etc., como meta-variáveis proposicionais, isto é são usadas para representar qualquer proposição (atômica ou composta).

O **alfabeto da linguagem da lógica proposicional** Σ é constituído de três conjuntos disjuntos.

1. O conjunto enumerável Σ_V das variáveis proposicionais, $\Sigma_V = \{p_1, \ldots, p_n, \ldots\}$.
2. O conjunto Σ_C dos conectivos proposicionais, $\Sigma_C = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.
3. O conjunto Σ_P dos parênteses $\Sigma_P = \{(,)\}$.

Portanto $\Sigma = \Sigma_V \cup \Sigma_C \cup \Sigma_P$, o qual denotaremos, simplesmente, por

$$\Sigma = \{p_1, \ldots, p_n, \ldots, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}.$$

A **linguagem proposicional**, $\mathsf{L_P}$, é a linguagem gerada pela linguagem formal $\mathcal{L}_P = \langle \Sigma, \mathcal{G} \rangle$, onde Σ é o alfabeto proposicional acima e $\mathcal{G}$ é a gramática constituída pelas seguintes regras.

$$\mathsf{g_1}. \ : \frac{-}{p_i} \ , \ p_i \in \Sigma_V \qquad \mathsf{g_2}. \ : \frac{\alpha}{(\neg\alpha)}$$

$$\mathsf{g_3}. \ : \frac{\alpha, \beta}{(\alpha \wedge \beta)} \qquad \mathsf{g_4}. \ : \frac{\alpha, \beta}{(\alpha \vee \beta)}$$

$$\mathsf{g_5}. \ : \frac{\alpha, \beta}{(\alpha \rightarrow \beta)} \qquad \mathsf{g_6}. \ : \frac{\alpha, \beta}{(\alpha \leftrightarrow \beta)}$$

Observações:

1. $\mathsf{L_P} = Ling(\mathcal{L}_P)$.
2. As variáveis proposicionais são chamadas **fórmulas atômicas**, pois elas denotam proposições atômicas.
3. Os elementos da linguagem proposicional são chamados de proposições, fórmulas ou fórmulas bem formadas.
4. A regra $\mathsf{g_1}$ é na verdade um conjunto de regras (uma para cada símbolo proposicional).

5. É usual, nos textos clássicos de lógica, denominar-se de linguagem proposicional ao conjunto das fórmulas bem formadas (fbf) do cálculo proposicional.

Exemplo 3.2.1 *"$\stackrel{\text{def}}{=}$" é usado com o sentido "é definido por". α_1, α_2 e α_3, abaixo, são exemplos de proposições em $\mathsf{L_P}$.*

$$\alpha_1 \stackrel{\text{def}}{=} p_i, \quad \alpha_2 \stackrel{\text{def}}{=} ((p_1 \vee p_2) \to p_3), \quad \alpha_3 \stackrel{\text{def}}{=} ((p_1 \wedge p_2) \leftrightarrow (p_1 \to p_2))$$

pois, elas podem ser geradas pela linguagem formal da lógica proposicional. No caso de α_3 temos a seguinte prova (junto com o argumento a cada passo da prova) de que $\alpha_3 \in L_P$:

$w_1 \stackrel{\text{def}}{=}$	p_1	*Direto da regra g_1*
$w_2 \stackrel{\text{def}}{=}$	p_2	*Direto da regra g_1*
$w_3 \stackrel{\text{def}}{=}$	$(p_1 \wedge p_2)$	*Aplicando g_3 a w_1 e w_2*
$w_4 \stackrel{\text{def}}{=}$	$(p_1 \to p_2)$	*Aplicando g_5 a w_1 e w_2*
$w_5 \stackrel{\text{def}}{=}$	$((p_1 \wedge p_2) \leftrightarrow (p_1 \to p_2))$	*Aplicando g_6 a w_3 e w_4*

O conceito de subfórmula é intuitivo e natural. Basicamente, uma fórmula α é uma subfórmula de uma fórmula β se ela ocorre em β, no sentido que ela foi usada como premissa de uma das regras gramaticais na geração de β. Mas, para tornar este conceito mais preciso, daremos uma definição indutiva.

Definição 3.2.2 *Uma fórmula α é uma* **subfórmula** *de uma fórmula β se*

1. *α é o próprio β,*
2. *β é $(\neg\gamma)$ e α é uma subfórmula de γ, ou*
3. *β é $(\beta_1 \otimes \beta_2)$ para algum $\otimes \in \{\vee, \wedge, \to, \leftrightarrow\}$ e α é uma subfórmula de β_1 ou de β_2.*

Exemplo 3.2.3 *Seja α_3 a fórmula descrita no exemplo 3.2.1. Então, α_3 tem as seguintes subfórmulas:*

1. p_1
2. p_2
3. $(p_1 \wedge p_2)$
4. $(p_1 \to p_2)$
5. $((p_1 \wedge p_2) \leftrightarrow (p_1 \to p_2))$

O **escopo** de um conectivo de Σ_C é a proposição ou proposições às quais o conectivo se aplica quando visto como uma operação sobre $\mathsf{L_P}$. Aqui o conectivo $\neg$ é uma operação unária, isto é, pede apenas um argumento, enquanto os demais são binários, requerem, portanto, dois argumentos.

Para que as expressões complicadas se tornem mais legíveis, adotaremos algumas convenções para eliminação de parênteses. Primeiro omitiremos os pares de parênteses mais externos. Assim, $(\alpha \to \beta)$ pode ser escrito como $\alpha \to \beta$ e $(\neg\alpha)$ como $\neg\alpha$.

Segundo, quando uma fórmula contém repetidas aplicações de um conectivo binário, os parênteses serão omitidos usando associação à esquerda. Por exemplo, a fórmula $(((\alpha \to (\beta \wedge \alpha)) \to \alpha) \to \gamma)$ se torna $\alpha \to (\beta \wedge \alpha) \to \alpha \to \gamma$

Por último, os conectivos são ordenados do maior ao menor escopo, ou da mais baixa precedência para a mais alta como segue:

$$\leftrightarrow, \quad \to, \quad \vee, \quad \wedge, \quad \neg$$

Assim, parênteses são eliminados seguindo a regra de que $\leftrightarrow$ tem o maior escopo e $\neg$ tem o menor, ou dito de outro modo, $\neg$ tem a mais alta precedência e $\leftrightarrow$ a menor. Ou seja, $\neg$ se aplica à menor proposição que a segue, em seguida $\wedge$, e assim até $\leftrightarrow$. Por exemplo, a fórmula

$$(((\neg\alpha) \vee (\neg(\beta \vee \gamma))) \leftrightarrow (\alpha \to (\beta \vee ((\neg\alpha) \wedge \alpha)))).$$

pode ser simplificada, eliminando parênteses de acordo com a convenção acima adotada, resultando na fórmula

$$\neg\alpha \vee \neg(\beta \vee \gamma) \leftrightarrow \beta \vee \neg\alpha \wedge \alpha$$

Claramente, quando eliminamos parênteses de uma fórmula não atômica, o resultado não será uma fórmula bem formada. Porém, devemos encarar estas fórmulas sem todos os parêntese, como abreviações ou simplificações, das respectivas fórmula com todos seus parênteses. De aqui para frente, as fbf's serão escritas sem considerar, provavelmente, todos seus parênteses. No entanto, com o intuito de preservar o entendimento da fórmula não usaremos, necessariamente, a convenção para tirar parêntese de modo exaustivo.

Uma linguagem L, sobre um alfabeto Σ, é decidível se existe um algoritmo tal que tendo como entrada uma expressão $\alpha \in \Sigma^*$ ele fornece como saída "sim" se $\alpha \in L$ e "não" se $\alpha \notin L$. Embora não entraremos em detalhes, observaremos que a linguagem $\mathsf{L_P}$, das proposições sobre o alfabeto Σ, acima especificada, é **decidível**.

3.3 Álgebras Boolenas

Em álgebra abstrata, **álgebra booleana** é uma estrutura algébrica, $\langle A, +, \cdot, \sim, 0, 1\rangle$, onde A um conjunto não vazio, $+$ e $\cdot$ são operações binárias sobre A, $\sim$ uma operação unária sobre A e 0 e 1 constantes, satisfazendo as propriedades:

1. *Comutatividade:* $x + y = y + x$ e $x \cdot y = y \cdot x$
2. *Associatividade:* $x + (y + z) = (x + y) + z$ e $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
3. *Distributividade:* $x \cdot (y + z) = x \cdot y + x \cdot z$ e $x + (y \cdot z) = (x + y) \cdot (x + z)$
4. *Complemento:* $x\cdot \sim x = 0$ e $x+ \sim x = 1$
5. *Absorção:* $x + (x \cdot y) = x$ e $x \cdot (x + y) = x$

Este nome foi dado em homenagem ao matemático inglês George Boole (1815-1864) quem introduziu está estrutura para estudar algebricamente a teoria dos conjuntos e a lógica proposicional. Observe que para qualquer conjunto A, $\langle \wp(A), \cup, \cap, \_, \emptyset, A\rangle$ é uma álgebra booleana.

As propriedades de associatividade, comutatividade e absorção, mostram que $\langle A, +, \cdot\rangle$ é um reticulado e portanto álgebras booleanas podem ser também definidas como sendo reticulados algébricos com complemento.

Existem outras propriedades que se desprendem das anteriores:

1. *Idempotência:* $x + x = x$ e $x \cdot x = x$
2. *Elemento neutro:* $x + 0 = x$ e $x \cdot 1 = x$
3. *Involução:* $\sim\sim x = x$
4. *Lei De Morgan:* $\sim (x + y) =\sim x\cdot \sim y$ e $\sim (x \cdot y) =\sim x+ \sim y$

Considere o conjunto $\mathbb{B} = \{0, 1\}$, no qual 0 será pensado como o valor verdade "falso" e 1 como o valor verdade "verdadeiro". Defina sobre $\mathbb{B}$ as seguintes operações: $\sim$, $+$, e $\cdot$ especificadas pelas seguintes tabelas, denominadas **tabelas verdade**:

x	$\sim y$
1	0
0	1

x	y	$x \cdot y$
1	1	1
1	0	0
0	1	0
0	0	0

x	y	$x + y$
1	1	1
1	0	1
0	1	1
0	0	0

é fácil ver que, de fato, $\sim$, $+$, e $\cdot$ são operações sobre $\mathbb{B}$, com $\sim$ sendo uma operação unária e $+$ e $\cdot$ binárias. Além disso, é fácil demonstrar que $\mathcal{B} = \langle \mathbb{B}, +, \cdot, \sim, 0, 1 \rangle$ é uma álgebra booleana. Por exemplo, se $x = 1$ então $x + (x \cdot y) = 1 + (1 \cdot y) = 1$ e se $x = 0$, então $x \cdot y = 0$ e portanto, $x + (x \cdots y) = 0 + (0 \cdot y) = 0 + 0 = 0$. Logo, podemos afirmar que para todo x e y em $\mathbb{B}$, $x + (x \cdot y) = x$.

Claramente, se pensarmos em termo de Σ-álgebras (onde $\Sigma = \{+, \cdot, \sim, 0, 1\}$), a álgebra booleana $\langle \mathbb{B}, +, \cdot, \sim, 0, 1 \rangle$ é inicial na classe de todas Σ-álgebras satisfazendo as Σ-equações anteriores com variáveis em $X = \{x, y, z\}$. Assim por exemplo, o único Σ-homomorfismo da álgebra booleana $\mathbb{B}$ em $\wp(\mathbb{N}\}$ é a função

$$h(1) = \mathbb{N} \text{ e } h(0) = \emptyset$$

Note que existem infinitos Σ-homomorfismos de $\wp\mathbb{N}$ em $\mathbb{B}$.

A partir das operações em uma álgebra boolena $\langle A, +, \cdot, \sim, 0, 1 \rangle$ podem-se obter funções $f : A^n \longrightarrow A$ mais complexas.

No caso específico da álgebra booleana $\mathbb{B}$, é possível perceber melhor o comportamento desta função através de sua tabela verdade, isto é uma tabela que forneça o valor de saída para todos os valores possíveis que possa receber como entrada a função (ao todo 2^n combinações possíveis). Este valor de saída pode ser obtido, em caso de funções complexas, passo a passo.

Por exemplo a tabela verdade a seguir fornece os valores intermediários necessários para se especificar a função $f : \mathbb{B}^3 \longrightarrow \mathbb{B}$ definida por

$$f(x, y, z) = (x+ \sim z) \cdot (\sim x+ \sim y$$

x	y	z	$\sim z$	$z+ \sim z$	$\sim z$	$\sim y$	$\sim x+ \sim y$	$(x+ \sim z) \cdot (\sim x+ \sim y)$
1	1	1	0	1	0	0	0	0
1	1	0	1	1	0	0	0	0
1	0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	1	1
0	1	1	0	0	1	0	1	0
0	1	0	1	1	1	0	1	1
0	0	1	0	0	1	1	1	0
0	0	0	1	1	1	1	1	1

Duas funções são particularmente interessantes: $\Rightarrow\colon \mathbb{B}^2 \longrightarrow \mathbb{B}$ e $\Leftrightarrow\colon \mathbb{B}^2 \longrightarrow \mathbb{B}$ definidas por $\Rightarrow(\alpha,\beta) = \sim\alpha + \beta$ e $\Leftrightarrow(\alpha,\beta) = \Rightarrow(\alpha,\beta)\cdot\Rightarrow(\beta,\alpha)$. De ora em diante nos usaremos para esta funções a notação infixa. Essas funções têm o seguinte comportamento:

x	y	$x \Rightarrow y$
1	1	1
1	0	0
0	1	1
0	0	1

x	y	$x \Leftrightarrow y$
1	1	1
1	0	0
0	1	0
0	0	1

Assim, nós podemos pensar em $x \Rightarrow y$ e $x \Leftrightarrow y$ como novas operações ou como abreviações de $\sim x + y$ e $(\sim x + y)\cdot(\sim y + x)$.

3.4 Valoração de fórmulas proposicionais

De posse da álgebra booleana $\mathbb{B}$ é possível definir uma interpretação para a linguagem proposicional $\mathsf{L_P}$. Antes porém, daremos a seguinte definição.

Definição 3.4.1 *Seja V o subconjunto das variáveis proposicionais do alfabeto da linguagem proposicional $\mathsf{L_P}$. Uma* **atribuição de valores verdade** *é uma função $\rho : V \longrightarrow \mathbb{B}$.*

Como $\mathsf{L_P}$ é definida indutivamente a partir de V, usando a proposição 2.3.9, é possível estender a atribuição de valores verdade ρ para uma função $\mathcal{V}_\rho : \mathsf{L_P} \longrightarrow \mathbb{B}$. Esta função é chamada **função de valoração** associada à atribuição ρ, ou simplesmente, valoração. $\mathcal{V}_\rho$, também, conhecida como **função semântica**. Observe que uma vez que V é um conjunto infinito de variáveis proposicionais é possível definir infinitas atribuições de valores verdade ρ de V em $\mathbb{B}$, e cada valoração $\mathcal{V}_\rho$ depende da atribuição ρ. Assim, $\mathcal{V}_\rho$ é definida recursivamente tendo como base da recursão a função ρ, como mostrado abaixo:

1. $\mathcal{V}_\rho(p) = \rho(p)$ para todo $p \in V$
2. $\mathcal{V}_\rho(\neg\alpha) = \sim \mathcal{V}_\rho(\alpha)$
3. $\mathcal{V}_\rho(\alpha \vee \beta) = \mathcal{V}_\rho(\alpha) + \mathcal{V}_\rho(\beta)$
4. $\mathcal{V}_\rho(\alpha \wedge \beta) = \mathcal{V}_\rho(\alpha) \cdot \mathcal{V}_\rho(\beta)$
5. $\mathcal{V}_\rho(\alpha \rightarrow \beta) = \mathcal{V}_\rho(\alpha) \Rightarrow \mathcal{V}_\rho(\beta)$
6. $\mathcal{V}_\rho(\alpha \leftrightarrow \beta) = \mathcal{V}_\rho(\alpha) \Leftrightarrow \mathcal{V}_\rho(\beta)$

Um caso particularmente interessante de atribuição se dá no seguinte caso. Seja a proposição $\alpha \stackrel{\text{def}}{=} (p_1 \vee p_2) \rightarrow p_3$. Neste caso dizemos que α é constituída das proposições atômicas p_1, p_2 e p_3. Em geral, se α é uma proposição cujo conjunto de proposições atômicas é $\{p_1, \ldots, p_n\}$, então para determinar o valor verdade de α, os valores que tomem as variáveis proposicionais p_i, com $i > n$, é absolutamente irrelevante. Assim, podemos dizer que uma atribuição de valores verdade para α é uma função $\rho : \{p_1, \ldots, p_n\} \longrightarrow \mathbb{B}$ e portanto só teríamos uma quantidade finita (2^n) de possíveis atribuições. Analogamente, $\mathcal{V}_\rho$ é uma valoração para α, isto é, $\mathcal{V}_\rho$ associa o valor 0 ou 1 a α dependendo da atribuição ρ.

Exemplo 3.4.2 *Seja* $\alpha \stackrel{\text{def}}{=} (p_1 \vee p_2) \rightarrow p_3$. *Achar* $\mathcal{V}_\rho(\alpha)$ *para o seguinte* ρ: $\rho(p_1) = 0$, $\rho(p_2) = 1$ *e* $\rho(p_3) = 0$. *Então,*

$$\begin{aligned}
\mathcal{V}_\rho((p_1 \vee p_2) \rightarrow p_3) &= \mathcal{V}_\rho(p_1 \vee p_2) \Rightarrow \mathcal{V}_\rho(p_3) \\
&= (\mathcal{V}_\rho(p_1) + \mathcal{V}_\rho(p_2)) \Rightarrow \mathcal{V}_\rho(p_3) \\
&= (\rho(p_1) + \rho(p_2)) \Rightarrow \rho(p_3) \\
&= (0 + 1) \Rightarrow 0 \\
&= 1 \Rightarrow 0 \\
&= 0.
\end{aligned}$$

Em alguns casos é desejável se saber o comportamento da fórmula em todas as possíveis interpretações, isto é considerar cada possível função de atribuição de valores verdade para as variáveis proposicionais.

Exemplo 3.4.3 *Seja a mesma fórmula do exemplo anterior, isto é* $\alpha \stackrel{\text{def}}{=} (p_1 \vee p_2) \rightarrow p_3$. *As possíveis funções de atribuição de valores verdade para as variáveis proposicionais* p_1, p_2 *e* p_3 *são:*

1. $\rho_1(p_1) = 1$, $\rho_1(p_2) = 1$ *e* $\rho_1(p_3) = 1$;
2. $\rho_2(p_1) = 1$, $\rho_2(p_2) = 1$ *e* $\rho_2(p_3) = 0$;
3. $\rho_3(p_1) = 1$, $\rho_3(p_2) = 0$ *e* $\rho_3(p_3) = 1$;
4. $\rho_4(p_1) = 1$, $\rho_4(p_2) = 0$ *e* $\rho_4(p_3) = 0$;
5. $\rho_5(p_1) = 0$, $\rho_5(p_2) = 1$ *e* $\rho_5(p_3) = 1$;
6. $\rho_6(p_1) = 0$, $\rho_6(p_2) = 1$ *e* $\rho_6(p_3) = 0$;
7. $\rho_7(p_1) = 0$, $\rho_7(p_2) = 0$ *e* $\rho_7(p_3) = 1$;
8. $\rho_8(p_1) = 0$, $\rho_8(p_2) = 0$ *e* $\rho_8(p_3) = 0$.

Para cada função de atribuição ρ_i temos uma função semântica que da o valor verdade que toma a fórmula α para a atribuição ρ_i. Esta função semântica se comporta da seguinte maneira:

$$\begin{aligned}\mathcal{V}_{\rho_i}((p_1 \vee p_2) \to p_3) &= \mathcal{V}_{\rho_i}(p_1 \vee p_2) \Rightarrow \mathcal{V}_{\rho_i}(p_3)\\ &= (\mathcal{V}_{\rho_i}(p_1) + \mathcal{V}_{\rho}(p_2)) \Rightarrow \mathcal{V}_{\rho_i}(p_3)\\ &= (\rho_i(p_1) + \rho_i(p_2)) \Rightarrow \rho_i(p_3)\end{aligned}$$

Por tanto, as possíveis interpretações de α são:

1. $\begin{aligned}\mathcal{V}_{\rho_1}((p_1 \vee p_2) \to p_3) &= (\rho_1(p_1) + \rho_1(p_2)) \Rightarrow \rho_1(p_3)\\ &= (1+1) \Rightarrow 1\\ &= 1 \Rightarrow 1\\ &= 1\end{aligned}$
2. $\begin{aligned}\mathcal{V}_{\rho_2}((p_1 \vee p_2) \to p_3) &= (\rho_2(p_1) + \rho_2(p_2)) \Rightarrow \rho_2(p_3)\\ &= (1+1) \Rightarrow 0\\ &= 1 \Rightarrow 0\\ &= 0\end{aligned}$
3. $\begin{aligned}\mathcal{V}_{\rho_3}((p_1 \vee p_2) \to p_3) &= (\rho_3(p_1) + \rho_3(p_2)) \Rightarrow \rho_3(p_3)\\ &= (1+0) \Rightarrow 1\\ &= 1 \Rightarrow 1\\ &= 1\end{aligned}$
4. $\begin{aligned}\mathcal{V}_{\rho_4}((p_1 \vee p_2) \to p_3) &= (\rho_4(p_1) + \rho_4(p_2)) \Rightarrow \rho_4(p_3)\\ &= (1+0) \Rightarrow 0\\ &= 1 \Rightarrow 0\\ &= 0\end{aligned}$
5. $\begin{aligned}\mathcal{V}_{\rho_5}((p_1 \vee p_2) \to p_3) &= (\rho_5(p_1) + \rho_5(p_2)) \Rightarrow \rho_5(p_3)\\ &= (0+1) \Rightarrow 1\\ &= 1 \Rightarrow 1\\ &= 1\end{aligned}$
6. $\begin{aligned}\mathcal{V}_{\rho_6}((p_1 \vee p_2) \to p_3) &= (\rho_6(p_1) + \rho_6(p_2)) \Rightarrow \rho_6(p_3)\\ &= (0+1) \Rightarrow 0\\ &= 1 \Rightarrow 0\\ &= 0\end{aligned}$
7. $\begin{aligned}\mathcal{V}_{\rho_7}((p_1 \vee p_2) \to p_3) &= (\rho_7(p_1) + \rho_7(p_2)) \Rightarrow \rho_7(p_3)\\ &= (0+0) \Rightarrow 1\\ &= 0 \Rightarrow 1\\ &= 1\end{aligned}$
8. $\begin{aligned}\mathcal{V}_{\rho_8}((p_1 \vee p_2) \to p_3) &= (\rho_8(p_1) + \rho_8(p_2)) \Rightarrow \rho_8(p_3)\\ &= (0+0) \Rightarrow 0\\ &= 0 \Rightarrow 0\\ &= 1\end{aligned}$

É usual nos livros textos de cálculo proposicional apresentar os cálculos acima através da seguinte tabela:

p_1	p_2	p_3	$p_1 \vee p_2$	$p_1 \vee p_2 \rightarrow p_3$
1	1	1	1	1
1	1	0	1	0
1	0	1	1	1
1	0	0	1	0
0	1	1	1	1
0	1	0	1	0
0	0	1	0	1
0	0	0	0	1

Este tipo de tabelas são conhecidas como **tabelas verdade**. A última coluna contém os valores verdade da expressão em todas as possíveis interpretações. Assim, na tabela há uma mistura de contextos, por um lado uma expressão que é de natureza sintática e por outro valores verdade obtidos como avaliações na álgebra booleana que são de natureza semântica. Porém embora $\sim$, $+$, $\cdot$, $\Rightarrow$ e $\Leftrightarrow$ sejam interpretações (dimensão semântica) de $\neg$, $\vee$, $\wedge$, $\rightarrow$ e $\leftrightarrow$ (dimensão sintática), respectivamente, é comum usar em ambos os casos os mesmos símbolos para a dimensão sintática e semântica. Por isso, de ora em diante usaremos $\neg$, $\vee$, $\wedge$, $\rightarrow$ e $\leftrightarrow$ em ambas dimensões, e a distinção será inferida do contexto.

Exemplo 3.4.4 *A tabela verdade da proposição* $p_1 \rightarrow (p_2 \wedge \neg(p_3 \rightarrow p_1))$ *é dada abaixo:*

p_1	p_2	p_3	$p_3 \rightarrow p_1$	$\neg(p_3 \rightarrow p_1)$	$p_2 \wedge \neg(p_3 \rightarrow p_1)$	$p_1 \rightarrow (p_2 \wedge \neg(p_3 \rightarrow p_1))$
1	1	1	1	0	0	0
1	1	0	1	0	0	0
1	0	1	1	0	0	0
1	0	0	1	0	0	0
0	1	1	0	1	1	1
0	1	0	1	0	0	1
0	0	1	0	1	0	1
0	0	0	1	0	0	1

Tabelas verdades de proposições complexas são muito longas, pois devemos ir escrevendo os resultados parciais de cada subfórmula até chegar à fórmula toda. Uma maneira de fazer isto ocupando menos espaço é ir colocando os resultados parciais abaixo de cada conectivo lógico. Para visualizar melhor o resultado final da tabela, usaremos barras duplas para a coluna do conectivo principal da fórmula.

Exemplo 3.4.5 *A tabela verdade compactada da proposição* $p_1 \rightarrow (p_2 \wedge \neg(p_3 \rightarrow p_1))$ *é dada abaixo:*

p_1	$\rightarrow$	$(p_2$	$\wedge$	$\neg$	$(p_3$	$\rightarrow$	$p_1))$
1	0	1	0	0	1	1	1
1	0	1	0	0	0	1	1
1	0	0	0	0	1	1	1
1	0	0	0	0	0	1	1
0	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	0	0	0	1	0

Em algumas fórmulas usaremos meta-variáveis proposicionais (α, β, etc.) em vez de variáveis proposicionais (p_1, p_2, etc.). Nestes casos chamaremos estas fórmulas de meta-proposições. A idéia que uma meta proposição representa um conjunto de proposições, pois as meta-variáveis proposicionais podem ser substituídas por qualquer proposição. Por exemplo, as proposições $p_1 \wedge p_2$, $(p_1 \rightarrow p_1) \wedge \neg(p_1 \vee p_3)$ e $(p_1 \wedge p_2) \wedge p_3$ são instâncias da meta-proposição $\alpha \wedge \beta$. Tabelas verdades de meta-proposições são feitas de modo similar às tabelas verdades das proposições, isto é, como não se sabe qual a possível instância de uma meta-variável proposicional, se considera todas as possíveis combinações de valores verdade que possam ter as meta-variáveis.

Exemplo 3.4.6 *A tabela verdade (compactada) para a meta-proposição* $\alpha \rightarrow (\beta \rightarrow \neg\gamma)$ *é dada a seguir:*

α	$\rightarrow$	$(\beta$	$\wedge$	$\neg$	$\gamma)$
1	*0*	*1*	*0*	*0*	*1*
1	*1*	*1*	*1*	*1*	*0*
1	*0*	*0*	*0*	*0*	*1*
1	*0*	*0*	*0*	*1*	*0*
0	*1*	*1*	*0*	*0*	*1*
0	*1*	*1*	*1*	*1*	*0*
0	*1*	*0*	*0*	*0*	*1*
0	*1*	*0*	*0*	*1*	*0*

3.5 Lógica Proposicional

Nesta seção, a partir da linguagem proposicional $\mathsf{L_P}$, destacaremos um subconjunto próprio dela, o qual chamaremos lógica proposicional. Observaremos em seguida que, de fato, a lógica proposicional não se reduz a toda a linguagem $\mathsf{L_P}$.

Definição 3.5.1 *Seja* α *uma proposição de* $\mathsf{L_P}$. α *diz-se uma* **tautologia** *se qualquer que seja a atribuição de valores verdade para as proposições atômicas que constituem* α*, o valor*

verdade de α é sempre 1. Em outras palavras, se para cada atribuição de valores verdade $\rho : \{p_1, \ldots, p_n\} \longrightarrow \mathbb{B}$, onde $p_1, \ldots, p_n$ são as variáveis proposicionais que ocorrem em α, $\mathcal{V}_\rho(\alpha) = 1$.

Assim, tautologias são fórmulas que tomam o valor verdade verdadeiro independente do valor que atribuamos às proposições atômicas que a compõem. Isto é, numa tautologia o importante é a forma e não sua interpretação. Por exemplo, a conhecida frase de Shakespeare em Hamlet, "ser ou não ser", é uma tautologia, pois ela é verdadeira independente do que se queira dizer com "ser" (ser homem, ser alto[1], ser vítima, etc.), o importante nela é a forma da frase, $p_1 \vee \neg p_1$. Assim, tautologias são fórmulas que sempre são verdadeiras, independente da interpretação, do avaliador e do contexto.

Exemplo 3.5.2 *As seguintes proposições e meta-proposições são tautologias.*

1. $p_1 \rightarrow (p_2 \rightarrow p_1)$
2. $(p_1 \rightarrow (p_2 \rightarrow p_3)) \rightarrow ((p_1 \rightarrow p_2) \rightarrow (p_1 \rightarrow p_3))$
3. $(\neg p_2 \rightarrow \neg p_1) \rightarrow ((\neg p_2 \rightarrow p_1) \rightarrow p_2)$
4. $(\neg p_2 \rightarrow \neg p_1) \rightarrow (p_1 \rightarrow p_2)$
5. $\alpha \wedge \beta \rightarrow \alpha$
6. $\alpha \wedge \beta \rightarrow \beta$
7. $\alpha \rightarrow \neg\neg\alpha$
8. $\neg\neg\alpha \rightarrow \alpha$
9. $\alpha \rightarrow \alpha \vee \beta$
10. $\alpha \wedge \neg\alpha \rightarrow \beta$

Para verificar que estas fórmulas proposicionais são realmente tautologias basta fazer suas tabelas verdades (compactadas).

[1]Neste tipo de conjuntos, a propriedade que satisfazem seus elementos não é bem definida, por exemplo para alguém uma pessoa alta poderia ser qualquer pessoa que mede acima de um metro e oitenta centímetros, mas nesta visão uma pessoa que mede um metro e setenta nove centímetros e 9 milímetros não seria considerada alta enquanto que uma que mede um milímetro a mais sim seria. Uma lógica que consegue modelar melhor este tipo de proposições é a lógica fuzzy [Zad65, BB95, Kas96, Ngu99], onde um proposição não é simplesmente verdadeira ou falsa, mas possui um grau de verdade que varia entre falso (0) e verdadeiro (1).

p_1	$\rightarrow$	$(p_2$	$\rightarrow$	$p_1)$
1	*1*	*1*	*1*	*1*
1	*1*	*0*	*1*	*1*
0	*1*	*1*	*0*	*0*
0	*1*	*0*	*1*	*0*

$(p_1$	$\rightarrow$	$(p_2$	$\rightarrow$	$p_3))$	$\rightarrow$	$((p_1$	$\rightarrow$	$p_2)$	$\rightarrow$	$(p_1$	$\rightarrow$	$p_3))$
1	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
1	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*
1	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
1	*1*	*0*	*1*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*
0	*1*	*1*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*0*	*1*	*1*
0	*1*	*1*	*0*	*0*	*1*	*0*	*1*	*1*	*1*	*0*	*1*	*0*
0	*1*	*0*	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*1*
0	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*

$(\neg$	p_2	$\rightarrow$	$\neg$	$p_1)$	$\rightarrow$	$((\neg$	p_2	$\rightarrow$	$p_1)$	$\rightarrow$	$p_2)$
0	*1*	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
1	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*0*	*0*
0	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*0*	*1*	*1*
1	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*0*

$(\neg$	p_2	$\rightarrow$	$\neg$	$p_1)$	$\rightarrow$	$(p_1$	$\rightarrow$	$p_2)$
0	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
1	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
0	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*1*
1	*0*	*1*	*1*	*0*	*1*	*0*	*1*	*0*

α	$\wedge$	β	$\rightarrow$	α
1	*1*	*1*	*1*	*1*
1	*0*	*0*	*1*	*1*
0	*0*	*1*	*1*	*0*
0	*0*	*0*	*1*	*0*

α	$\wedge$	β	$\rightarrow$	β
1	*1*	*1*	*1*	*1*
1	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*1*
0	*0*	*0*	*1*	*0*

α	$\rightarrow$	$\neg$	$\neg$	α
1	*1*	*1*	*0*	*1*
0	*1*	*0*	*1*	*0*

$\neg$	$\neg$	α	$\rightarrow$	α
1	*0*	*1*	*1*	*1*
0	*1*	*0*	*1*	*0*

α	$\rightarrow$	α	$\vee$	β
1	*1*	*1*	*1*	*1*
1	*1*	*1*	*1*	*0*
0	*1*	*0*	*1*	*1*
0	*1*	*0*	*0*	*0*

α	$\wedge$	$\neg$	α	$\rightarrow$	β
1	*0*	*0*	*1*	*1*	*1*
1	*0*	*0*	*1*	*1*	*0*
0	*0*	*1*	*0*	*1*	*1*
0	*0*	*1*	*0*	*1*	*0*

Uma proposição α de $\mathsf{L_P}$ é um **contradição** se $\mathcal{V}_\rho(\alpha) = 0$ para toda atribuição de valores verdades ρ para as proposições atômicas de α. Um exemplo de contradição é $\alpha \stackrel{\text{def}}{=} p \wedge \neg p$. Observe que, α é uma tautologia se, e somente se, $\neg\alpha$ é uma contradição e, inversamente, α é uma contradição se, e somente se, $\neg\alpha$ é uma tautologia. As fbf's que não são nem tautologia nem contradição, por exemplo $p_1 \rightarrow p_2$, são chamadas de **contingentes**, pois o seu valor verdade delas varia de acordo com a interpretação de seus símbolos proposicionais, ou seja, é falsa em alguma interpretação e verdadeira em outra. Uma proposição α de $\mathsf{L_P}$ é dita **satisfatível** se existe uma atribuição de valores verdades ρ para as proposições atômicas de α, tal que $\mathcal{V}_\rho(\alpha) = 1$, ou seja se existe uma interpretação onde a fórmula for verdadeira. Inversamente, Uma proposição α de $\mathsf{L_P}$ é dita **insatisfatível** se não for satisfatível, ou seja se for uma contradição.

Observe que a proposição $p_1 \rightarrow (p_2 \wedge \neg(p_3 \rightarrow p_1))$ do exemplo 3.4.5 é uma instância da meta-proposição $\alpha \rightarrow (\beta \rightarrow \neg\gamma)$ do exemplo 3.4.6. Observe também que as respectivas tabelas verdades são diferentes. Isto acontece porque a meta-proposição é contingente. Se fosse uma tautologia ou contradição, então qualquer instância dela continuaria sendo uma tautologia ou contradição. De hora em diante, chamaremos meta-proposições de proposições.

Proposição 3.5.3 *Seja $Taut(\mathsf{L_P})$ o conjunto de todas as tautologias na linguagem proposicional $\mathsf{L_P}$. Então, $Taut(\mathsf{L_P})$ é um subconjunto próprio de $\mathsf{L_P}$.*

DEMONSTRAÇÃO: Basta exibir uma proposição α de $\mathsf{L_P}$ que não é uma tautologia. Isto é, uma proposição α e uma atribuição de valores verdade ρ para as proposições atômicas que constituem α tal que $\mathcal{V}_\rho(\alpha) = 0$.

Seja, portanto, $\alpha \stackrel{\text{def}}{=} p_1 \rightarrow p_2$ e tome $\rho(p_1) = 1$ e $\rho(p_2) = 0$. então

$$\mathcal{V}_\rho(p_1 \rightarrow p_2) = \mathcal{V}_\rho(p_1) \Rightarrow \mathcal{V}_\rho(p_2) = 1 \Rightarrow 0 = 0. \blacksquare$$

Definição 3.5.4 *Chamaremos* **lógica proposicional** *sobre a linguagem* $\mathsf{L_P}$ *ao conjunto* $Taut(\mathsf{L_P})$*, de todas as tautologias em* $\mathsf{L_P}$*.*

Definição 3.5.5 *Como* $Taut(\mathsf{L_P}) \neq \mathsf{L_P}$ *dizemos que a lógica proposicional é* **consistente**. *Ou seja, em geral dizemos, que uma lógica sobre uma linguagem L é* **consistente** *se ela não se reduz à linguagem.*

A primeira vista a nossa definição de lógica proposicional como o conjunto de tautologias $Taut(\mathsf{L_P})$ não corresponde a nossa intuição de que "lógica é um instrumento para descobrir que uma proposição, a qual chamamos de tese, decorre de um conjunto de proposições, chamadas hipóteses". Em outras palavras, que "lógica é usada como um aparato dedutivo". Entretanto, veremos que é possível construir esse "aparato dedutivo" a partir de $Taut(\mathsf{L_P})$.

Seja $\mathcal{P}(\mathsf{L_P})$ o conjunto das partes de $\mathsf{L_P}$ e $\models \subseteq \mathcal{P}(\mathsf{L_P}) \times \mathsf{L_P}$ uma relação sobre $\mathsf{L_P}$ que associa a um subconjunto $\Gamma \subseteq \mathsf{L_P}$ uma proposição de $\mathsf{L_P}$. Usaremos as letras gregas maiúsculas, talvez indexadas, para denotar conjuntos de proposições.

Definição 3.5.6 *Seja* $\alpha \in \mathsf{L_P}$ *e* $\Gamma \subseteq \mathsf{L_P}$*. Dizemos que* α *é uma* **conseqüência semântica** *ou* **conseqüência lógica** *de* Γ*, denotado por* $\Gamma \models \alpha$*, se para toda atribuição de valores verdade* ρ *tal que* $\mathcal{V}_\rho(\beta) = 1$ *para toda proposição* $\beta \in \Gamma$*, então* $\mathcal{V}_\rho(\alpha) = 1$*.*

Assim, quando dizemos que $\Gamma \models \alpha$ estamos dizendo que podemos concluir que a proposição α é verdadeira desde que todas as hipóteses em Γ o sejam. Assim, a noção de conseqüência semântica possui essa característica de dedução desejada para a lógica proposicional.

Notação: Quando o conjunto Γ é finito, isto é, $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$, usaremos a notação mais usual, $\alpha_1, \ldots, \alpha_n \models \alpha$ em vez de $\{\alpha_1, \ldots, \alpha_n\} \models \alpha$.

Exemplo 3.5.7

1. $\alpha, \alpha \to \beta \models \beta$

   De fato, $\mathcal{V}_\rho(\alpha \to \beta) = 0$ *se, e somente se,* $\mathcal{V}_\rho(\alpha) = 1$ *e* $\mathcal{V}_\rho(\beta) = 0$*. Portanto, se* $\mathcal{V}_\rho(\alpha \to \beta) = 1$ *e* $\mathcal{V}_\rho(\alpha) = 1$*, então necessariamente* $\mathcal{V}_\rho(\beta) = 1$*.*

2. $\alpha \to \beta, \alpha \to \neg\beta \models \neg\alpha$

   Um modo simples de mostrar isso é usar a tabela verdade

α	β	$\alpha \to \beta$	$\alpha \to \neg\beta$	$\neg\alpha$
1	*1*	*1*	*0*	*0*
1	*0*	*0*	*1*	*0*
0	*1*	*1*	*1*	*1*
0	*0*	*1*	*1*	*1*

Observando a tabela constatamos que sempre que as premissas $\alpha \to \beta$ e $\alpha \to \neg\beta$ são verdadeiras (isto é, têm valor 1) a conclusão $\neg\alpha$ também é verdadeira.

Exemplo 3.5.8 *Considere a seguinte postura de Socrates:*

"Socrates está em tal situação que ele estaria disposto a visitar Platão, só se Platão estivesse disposto a visitá-lo."

e de Platão:

"Platão está em tal situação que ele não estaria disposto a visitar Sócrates, se Socrates estivesse disposto a visitá-lo, mas estaria disposto a visitar Socrates, se Socrates não estivesse disposto a visitá-lo."

Agora respondamos a seguinte pergunta: "Socrates está disposto a visitar Platão?"

Como a Decisão de Socrates depende da postura de Platão, esta pergunta terá como resposta "sim", somente se a afirmação "Socrates está disposto visitar Platão" for conseqüência lógica das posturas de ambos. Mas para ver isso teremos que descrever ambas posições na linguagem proposicional. Para isso, primeiro devemos distinguir quais as proposições atômicas. Distinguimos duas:

- $p =$ *"Socrates está disposto a visitar Platão"*
- $q =$ *"Platão está disposto a visitar Socrates"*

Assim, a postura de Socrates e Platão podem ser re-escritas na linguagem proposicional como:

- *Socrates:* $q \to p$
- *Platão:* $(p \to \neg q) \wedge (\neg p \to q)$

Logo, para responder a pergunta devemos determinar se

$$\{q \to p, (p \to \neg q) \wedge (\neg p \to q)\} \models p$$

ou não. Vejamos isto usando tabelas verdades:

p	q	$q \rightarrow p$	$p \rightarrow \neg q$	$\neg p \rightarrow q$	$(p \rightarrow \neg q) \wedge (\neg p \rightarrow q)$
1	1	1	0	1	0
1	0	1	1	1	1
0	1	0	1	1	1
0	0	1	1	0	0

Assim, colocando lado a lado os valores de cada fórmula

$q \rightarrow p$	$(p \rightarrow \neg q) \wedge (\neg p \rightarrow q)$	p
1	0	1
1	1	1
0	1	0
1	0	0

e identificando as situações em que as posturas de Socrates e de Platão são satisfeitas (verdadeiras), temos que a única situação em que isso ocorre é na segunda linha, mas nesse caso o valor de p é 1. Logo, podemos concluir que $\{q \rightarrow p, (p \rightarrow \neg q) \wedge (\neg p \rightarrow q)\} \models p$*, e portanto a resposta à pergunta é "Sim".*

Na definição 3.5.6 se $\Gamma = \emptyset$, a primeira parte da definição é satisfeita, isto é, como não existe proposição em Γ, então para qualquer que seja a atribuição ρ, temos que $\mathcal{V}_\rho(\beta) = 1$ para todo $\beta \in \Gamma$. Portanto, para α ser um modelo do conjunto vazio, isto é $\emptyset \models \alpha$ ou simplesmente $\models \alpha$, deve-se satisfazer a seguinte condição: para qualquer que seja a atribuição ρ (ou seja, independente dos valores verdade das proposições atômicas que compõem α) o valor verdade de α deve ser 1 (isto é $\mathcal{V}_\rho(\alpha) = 1$). Portanto α deve ser uma tautologia. Assim, a lógica proposicional $Taut(\mathsf{L_P})$ pode, então, ser re-definida como segue.

$$Taut(\mathsf{L_P}) = \{\alpha \in \mathsf{L_P} \ / \ \models \alpha\}.$$

Podemos concluir dos exemplos acima que a tabela verdade é um algoritmo para provar que $\Gamma \models \alpha$, no caso de Γ ser finito. Em particular tabelas verdade também nos permitem decidir se uma proposição é uma tautologia ou não. Isto significa que o conjunto $Taut(\mathsf{L_P})$ é decidível.

Se Γ for infinito ainda assim a tabela verdade é um algoritmo para decidir se $\Gamma \models \alpha$ ou não. Mas para isso usaremos o conhecido **teorema da compacidade** da lógica proposicional cujo enunciado é o seguinte.

Teorema 3.5.9 (Compacidade) *Se* $\Gamma \models \alpha$*, então existe um subconjunto finito* $\Gamma_0 \subseteq \Gamma$ *tal que* $\Gamma_0 \models \alpha$*.*

DEMONSTRAÇÃO: (Veja [End72] página 60). ■

A seguir enunciaremos um dos teoremas fundamentais da lógica clássica, conhecido como o teorema da dedução.

Teorema 3.5.10 (Dedução) *Sejam α e β proposições em $\mathsf{L_P}$. Então*

$$\alpha \models \beta \text{ se, e somente se, } \models \alpha \to \beta$$

DEMONSTRAÇÃO: Basta observar a tabela verdade abaixo e a definição de conseqüência lógica.

α	β	$\alpha \to \beta$
1	1	1
1	0	0
0	1	1
0	0	1

Pois, se $\alpha \models \beta$ então, necessariamente, o segundo caso ($\mathcal{V}_\rho(\alpha) = 1$ e $\mathcal{V}_\rho(\beta) = 0$) não pode ocorrer. Logo, $\mathcal{V}_\rho(\alpha \to \beta)$ sempre será 1, e portanto $\alpha \to \beta$ é uma tautologia. Inversamente, se $\alpha \to \beta$ é uma tautologia, então, necessariamente, o segundo caso e não pode ocorrer. Logo, quando $\mathcal{V}_\rho(\alpha) = 1$, necessariamente, $\mathcal{V}_\rho(\beta) = 1$ (primeiro caso). Portanto, $\alpha \models \beta$. ■

Corolário 3.5.11 *Sejam $\alpha_1, \ldots, \alpha_n, \alpha$ proposições. Então*

$$\alpha_1, \ldots, \alpha_n \models \alpha \text{ se, e somente se, } \alpha_1, \ldots, \alpha_{n-1} \models \alpha_n \to \alpha.$$

DEMONSTRAÇÃO: Usaremos o mesmo argumento da prova do teorema da dedução: a tabela verdade da implicação.

Suponha que $\alpha_1, \ldots, \alpha_n \models \alpha$. Logo por definição de conseqüência lógica, para cada interpretação dos símbolos proposicionais que façam com que todos os α_i, com $1 \leq i \leq n$, tomem o valor verdade verdadeiro, α também tomará o valor verdade verdadeiro. Consideremos, agora, o conjunto de interpretações nas quais todos os α_i, com $1 \leq i < n$, sejam verdadeiro, então se numa dessas interpretações α_n toma o valor verdade verdadeiro, α também tomará o valor verdade verdadeiro, pois por hipótese $\alpha_1, \ldots, \alpha_n \models \alpha$. Por outro lado se numa dessas interpretações α_n toma o valor verdade falso, então ,$\alpha_1, \ldots, \alpha_{n-1} \models \alpha_n \to \alpha$, pois pela tabela verdade da implicação $\alpha_n \to \alpha$ é verdadeiro cada vez que α_n é falso. A reversa desta prova, sai imediatamente usando o raciocínio inverso. ■

Vinculando o conceito de tautologia com a noção de conseqüência lógica, é possível justificar a definição de lógica proposicional como o conjunto de todas as tautologias, sem perder, no entanto, a noção de aparato dedutivo. Para isso, devemos observar que se $\Gamma \models \alpha$, isto é, se α é uma fbf que resulta como conseqüência de um conjunto de fbf's Γ (noção de aparato dedutivo), então usando o teorema da compacidade podemos obter um subconjunto finito $\Gamma_0 = \{\alpha_1, \ldots, \alpha_n\} \subseteq \Gamma$, tal que

$$\alpha_1, \ldots, \alpha_n \models \alpha.$$

Usando, o teorema da dedução repetidamente, podemos concluir que,

$$\models \alpha_1 \rightarrow (\alpha_2 \rightarrow (\ldots(\alpha_n \rightarrow \alpha)\ldots)$$

E portanto $\alpha_1 \rightarrow (\alpha_2 \rightarrow (\cdots(\alpha_n \rightarrow \alpha)\cdots)$ é uma tautologia. Ou seja, demonstrar que é possível deduzir uma fórmula α a partir de um conjunto de fórmulas Γ, se reduz a provar que uma certa fórmula é uma tautologia.

Esclarecido a relação entre tautologia e conseqüência lógica poderemos definir quando duas proposições são logicamente equivalentes.

Definição 3.5.12 *Sejam α e β proposições. Dizemos que α e β são* **logicamente equivalentes** *ou simplesmente* **equivalentes**, *cuja notação é $\alpha \equiv \beta$, se $\alpha \models \beta$ e $\beta \models \alpha$ ou, equivalentemente, pelo teorema da dedução, se $\models \alpha \rightarrow \beta$ e $\models \beta \rightarrow \alpha$ ou ainda, se $\models \alpha \leftrightarrow \beta$.*

Proposição 3.5.13 *A relação de equivalência lógica sobre $\mathsf{L_P}$ é uma relação de equivalência sobre $\mathsf{L_P}$.*

Demonstração:

1. $\alpha \equiv \alpha$ para todo $\alpha \in \mathsf{L_P}$, isto é, $\models \alpha \rightarrow \alpha$. Basta observar a tabela verdade.
2. Suponha que $\alpha \equiv \beta$. Então, trivialmente pela definição de equivalência lógica, $\beta \equiv \alpha$.
3. Suponha que $\alpha \equiv \beta$ e $\beta \equiv \gamma$. Mostraremos que $\alpha \equiv \gamma$. Por hipótese $\models \alpha \leftrightarrow \beta$ e $\models \beta \leftrightarrow \gamma$. Observando a tabela verdade do conectivo lógico $\leftrightarrow$, verifica-se que $\models \alpha \leftrightarrow \gamma$, isto é, $\alpha \equiv \gamma$. ■

Exemplo 3.5.14

1. $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$

2. $\alpha \rightarrow \beta \equiv \neg(\alpha \wedge \neg\beta)$

3. $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

4. Se α *e* β *são tautologias então* $\alpha \equiv \beta$.

O exemplo acima mostra que os conectivos $\rightarrow$ e $\leftrightarrow$ podem ser escritos em termos de $\neg$, $\wedge$ e $\vee$. Neste sentido para estudar a lógica proposicional basta considerar $\neg$, $\wedge$ e $\vee$ como conectivos primitivos e obter os outros dois como definição.

Sejam as seguintes operações $\overline{\neg}$, $\overline{\wedge}$ e $\overline{\vee}$ sobre $\mathsf{L_P}_{/\equiv}$[2]:

1. $\overline{\neg} : \mathsf{L_P}_{/\equiv} \longrightarrow \mathsf{L_P}_{/\equiv}$ é definida por $\overline{\neg}(\alpha) = \neg\alpha$
2. $\overline{\wedge} : \mathsf{L_P}_{/\equiv} \times \mathsf{L_P}_{/\equiv} \longrightarrow \mathsf{L_P}$ é definida por $\overline{\wedge}(\alpha, \beta) = \alpha \wedge \beta$
3. $\overline{\vee} : \mathsf{L_P}_{/\equiv} \times \mathsf{L_P}_{/\equiv} \longrightarrow \mathsf{L_P}_{/\equiv}$ é definida por $\overline{\vee}(\alpha, \beta) = \alpha \vee \beta$

e as constantes: $[p \vee \neg p]_{\equiv}$ e $[p \wedge \neg p]_{\equiv}$. É fácil ver que $\langle \mathsf{L_P}_{/\equiv}, \overline{\neg}, \overline{\vee}, \overline{\wedge}, [p \vee \neg p]_{\equiv}, [p \wedge \neg p]_{\equiv} \rangle$ constitui uma álgebra booleana. Mais ainda, essa álgebra booleana é isomorfa à álgebra booleana $\langle \mathbb{B}, \sim, +, \cdot, 0, 1 \rangle$. Isto justifica interpretar $\neg$ como $\sim$, $\vee$ como $+$ e $\wedge$ como $\cdot$.

Observação: Tendo em vista que $\alpha \wedge \beta \equiv \neg(\neg\alpha \vee \neg\beta)$ e $\alpha \vee \beta \equiv \neg(\neg\alpha \wedge \neg\beta)$ é possível usar $\{\neg, \vee\}$ ou $\{\neg, \wedge\}$ como conectivos básicos, tendo os demais como conectivos derivados.

3.6 Exercícios

1. Mostre que a lista selecionada de proposições a seguir são tautologias. Pense no símbolo $\equiv$ como $\leftrightarrow$.

   (a) Leis Distributivas:

   i. $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
   ii. $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

   (b) Leis da Negação:

   i. $\neg(\neg\alpha) \equiv \alpha$
   ii. $\neg(\alpha \rightarrow \beta) \equiv \alpha \wedge \neg\beta$
   iii. $\neg(\alpha \leftrightarrow \beta) \equiv (\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \beta)$

   (c) Leis de Morgan:

[2]Seja $\equiv$ uma relação de equivalência sobre um conjunto A e $a \in A$. A classe de equivalência de a é definida por $[a]_{\equiv} = \{b \in A \ / \ a \equiv b\}$. E $A/\equiv$ é o conjunto de todas as classes de equivalência em A, isto é: $A_{/\equiv} = \{[a]_{\equiv} \ / \ a \in A\}$.

i. $\alpha \vee \beta \equiv \beta \vee \alpha$
ii. $\alpha \wedge \beta \equiv \beta \wedge \alpha$
iii. $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
iv. $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$
v. $\neg(\alpha \wedge \beta) \equiv (\neg\alpha) \vee (\neg\beta)$
vi. $\neg(\alpha \vee \beta) \equiv (\neg\alpha) \wedge (\neg\beta)$
vii. $(\alpha \vee \beta) \wedge \beta \equiv \beta$
viii. $(\alpha \wedge \beta) \vee \beta \equiv \beta$

(d) Outras:

i. Lei do Terceiro Excluído: $\alpha \vee \neg\alpha$
ii. Lei da Contradição: $\neg(\alpha \wedge \neg\alpha)$
iii. Lei da Contraposição: $(\alpha \rightarrow \beta) \equiv ((\neg\beta) \rightarrow (\neg\alpha))$
iv. Lei da Exportação: $((\alpha \wedge \beta) \rightarrow \gamma) \equiv (\alpha \rightarrow (\beta \rightarrow \gamma))$

2. Identifique as proposições atômicas presentes nas seguintes afirmações, atribua um símbolo proposicional a elas, reescreva a frase agora como uma fórmula proposicional e diga em que circunstâncias (interpretações) elas seriam verdadeiras.

(a) João não joga futebol nem vôlei, mas joga tênis.

(b) Se Rosa é mãe de João e irmã de Pedro, então Ivan, que é filho de Pedro, é primo de João.

(c) Maria e Ana gostam de João, mas João não gosta delas.

(d) Letícia e Natália são irmãs de Rafael e de Pablo, mas Rafael não é irmão de Pablo.

(e) Sempre que João vá à praia, encontra com amigos, e sempre que João se encontra com amigos bebe umas cervejas.

3. Suponha que Rosa, Margarida e Violeta são irmãs e nenhuma delas tem outras irmãs. Rosa diz que só irá na festa se uma de suas irmãs for. Margarida diz que iria à festa se nenhuma de suas irmãs for, já Violeta diz que não irá à festa se suas duas irmãs forem. Se no dia da festa:

(a) Margarida não foi, enquanto Rosa e Violeta foram, podemos concluir que as três falaram a verdade?

(b) Rosa não foi enquanto as outras duas foram. Quem não fez o que falou?

(c) Se nenhuma foi, quem falou a verdade?

4. Seja $\langle A, +, \cdot, \sim, 0, 1\rangle$ uma álgebra booleana. Mostre que para todo $x, y, z \in A$

(a) $x + (y + z) = (z + x) + y$.

(b) $\neg x \cdot (x + y) = 0$.

(c) Se $x + y = x$ então $x \cdot y = y$.

(d) Se $x + y = 1$ então $y = \sim x$.

(e) Se $x \cdots y = 0$ então $y = \sim x$.

(f) Se $x + y = 0$ então $x = 0$ e $y = 0$.

(g) Se $x \cdot y = 1$ então $x = 1$ e $y = 1$.

5. Seja $\langle A, +, \cdot, \sim, 0, 1\rangle$ uma álgebra booleana. Mostre que para todo $x, y \in A$

   (a) *Idempotência:* $x + x = x$ e $x \cdot x = x$

   (b) *Elemento neutro:* $x + 0 = x$ e $x \cdot 1 = x$

   (c) *Involução:* $\sim\sim x = x$

   (d) *Lei De Morgan:* $\sim (x + y) = \sim x \cdot \sim y$ e $\sim (x \cdot y) = \sim x + \sim y$

6. Mostre que $(\alpha \to \beta) \to \alpha$ é conseqüência lógica de $\neg(\neg\alpha \vee \beta)$.

7. Mostre que $(\alpha \to \beta) \to \alpha$ não é conseqüência lógica de $\neg\alpha \vee \neg\beta$.

8. Mostre que nenhuma das duas proposições abaixo é conseqüência lógica da outra.

   - $\alpha \leftrightarrow (\beta \leftrightarrow \gamma)$
   - $(\alpha \wedge (\beta \wedge \gamma)) \vee ((\neg\alpha) \wedge ((\neg\beta) \wedge (\neg\gamma)))$

9. Mostre que toda tautologia é conseqüência lógica de qualquer fórmula.

10. Mostre que toda fórmula é conseqüência lógica de uma contradição.

11. Mostre que uma contradição só é conseqüência lógica de uma contradição.

12. Quais das seguintes proposições são tautologias? Justifique sua resposta.

    (a) $((\alpha \to \beta) \to \gamma) \to \gamma$

    (b) $\alpha \to (\alpha \to \beta)$

    (c) $\alpha \to (\beta \to \alpha)$

    (d) $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

    (e) $(\alpha \to \alpha) \to \alpha$

    (f) $((\neg\beta) \to (\neg\alpha)) \to \neg(\alpha \to \beta)$

13. Mostre que

    (a) Se $\Gamma \subseteq \Theta$ e $\Gamma \models \alpha$ então $\Theta \models \alpha$.

(b) Se $\Gamma, \alpha, \beta \models \gamma$ se, e somente se, $\Gamma, \alpha \wedge \beta \models \gamma$.

(c) Se $\Gamma \models \alpha$ então $\Gamma \models \alpha \vee \beta$

(d) $\alpha \models \beta$ e $\beta \models \alpha$ se, e somente se, $\models \alpha \leftrightarrow \beta$.

(e) Se $\Gamma \models \alpha$ para cada $\alpha \in \Theta$ e $\Theta \models \beta$ então $\Gamma \models \beta$.

14. Prove ou refute cada uma das seguintes asserções

(a) Se ou $\Gamma \models \alpha$ ou $\Gamma \models \beta$, então $\Gamma \models \alpha \vee \beta$.

(b) Se $\Gamma \models (\alpha \vee \beta)$, então ou $\Gamma \models \alpha$ ou $\Gamma \models \beta$.

(c) Se $\Gamma \models \alpha$ e $\Gamma \models \beta$, então $\Gamma \models \alpha \wedge \beta$.

(d) Se $\Gamma \models (\alpha \wedge \beta)$, então $\Gamma \models \alpha$ e $\Gamma \models \beta$.

15. Verifique se as seguintes proposições são satisfatíveis ou insatisfatíveis:

(a) $\neg((\alpha \rightarrow \beta) \rightarrow \alpha)$

(b) $(\alpha \rightarrow (\beta \rightarrow \gamma)) \leftrightarrow (\alpha \wedge (\beta \wedge \gamma))$

(c) $((\alpha \vee \neg\beta) \wedge (\beta \vee \gamma) \wedge (\neg\alpha \vee \gamma)) \rightarrow \psi$

(d) $\neg(((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha)$

(e) $(\alpha \rightarrow (\beta \rightarrow \gamma)) \leftrightarrow ((\alpha \wedge \beta) \rightarrow \gamma)$

(f) $(\alpha \leftrightarrow \beta) \rightarrow ((\alpha \wedge \beta) \vee (\alpha \vee \beta))$

(g) $((\alpha \rightarrow \beta) \wedge (\gamma \vee \neg\beta)) \rightarrow (\alpha \wedge \neg\gamma)$

16. Dar uma interpretação que satisfaça cada um dos seguintes conjuntos de proposições. Em outras palavras, dê uma atribuição de valores verdade que tornem todas as proposições do conjunto verdadeira.

(a) $\Gamma_1 = \{\alpha \rightarrow \neg\beta, \beta \wedge \gamma, \neg\gamma \vee \beta\}$

(b) $\Gamma_2 = \{\alpha \vee (\beta \rightarrow \gamma), \beta \wedge \neg\phi, \alpha \rightarrow \phi, \neg(\psi \rightarrow (\gamma \wedge \phi))\}$

(c) $\Gamma_3 = \{\alpha \wedge (\phi \vee \neg\psi), \beta \rightarrow \neg\phi, \neg(\gamma \wedge \alpha), \gamma \vee (\neg\alpha \vee \beta)\}$

17. Mostre que a proposição $(\alpha \wedge \gamma) \rightarrow \phi$ não é uma conseqüência lógica do conjunto $\Gamma = \{\alpha \rightarrow \beta, \gamma \rightarrow \beta, \phi \rightarrow \beta\}$.

18. Mostre que a proposição$(\alpha \vee \gamma) \rightarrow \phi$ é uma conseqüência lógica do conjunto $\Gamma = \{\alpha \rightarrow \beta, \gamma \rightarrow \beta, \beta \rightarrow \phi\}$.

19. Sejam as seguintes fbf da linguagem proposicional:

$\alpha = (p \wedge (q \vee \neg r)) \rightarrow \neg(p \wedge r)$

$\beta = \neg((p \wedge r) \rightarrow \neg p)$

$\gamma = (\neg q \wedge r)$

Diga, justificando sua resposta, se

(a) $\alpha \models \beta$
(b) $\alpha \models \gamma$
(c) $\beta \models \alpha$
(d) $\beta \models \gamma$
(e) $\gamma \models \alpha$
(f) $\gamma \models \beta$
(g) $\alpha, \beta \models \gamma$
(h) $\alpha, \gamma \models \beta$
(i) $\beta, \gamma \models \alpha$

ou não.

20. Considere a seguinte afirmação

"Suponha que Maria não gosta de João e que Maria só gosta de Pedro se Pedro não for amigo de João. Por outro lado, Pedro não é amigo de João se Maria não gostar de João. E Pedro só gosta de Maria se Maria gostar dele."

Responda as seguintes pergunta e justifique usando a noção de conseqüência lógica.

(a) Maria gosta de Pedro?
(b) Pedro gosta de Maria?
(c) Pedro é amigo de João?

Capítulo 4

A Teoria Formal da Lógica Proposicional

Nos capítulos anteriores caracterizamos a lógica proposicional como o conjunto das tautologias sobre a linguagem proposicional. Em seguida desenvolvemos um aparato lógico introduzindo o conceito de conseqüência lógica. A definição de conseqüência lógica pressupõe o conceito de tautologia, o qual, por sua vez é definida a partir da noção de interpretação, um conceito extrínseco à linguagem. A pesar da importância do conceito semântico de interpretação, principalmente como base para contra-exemplos de argumentos não "corretos", ele não provê uma ferramenta muito eficiente para achar conseqüências lógicas de premissas ou provar que uma dada proposição é uma conseqüência (lógica) de um conjunto de premissas. Por conseguinte, usualmente, é mais conveniente construir **provas** (ou derivações) da própria lógica proposicional (também conhecida por cálculo proposicional ou cálculo sentencial). A investigação geral de tais provas é o campo de interesse da **teoria da prova** da lógica proposicional, em contrapartida a sua teoria semântica, através da função de interpretação. A teoria da prova de uma linguagem formal começa com a caracterização da forma geral de uma prova e as regras (e às vezes axiomas) para construir provas (também chamadas derivações).

4.1 Teorias Formais

Tabelas verdade nos fornecem algoritmos para responder questões relativa aos conectivos vistos como funções booleanas, tais como se uma dada proposição é uma tautologia, contradição ou contingente (proposição cujo valor verdade depende dos valores verdade das proposições atômicas que a constitui), se uma proposição é uma conseqüência lógica de uma outra ou se é equivalente a uma outra. A parte mais complexa da lógica que veremos a seguir não pode ser manuseada com tabelas verdade, a abordagem será através de uma **teoria formal**, ou método formal. Embora as tabelas verdade dê conta inteiramente da teoria matemática da lógica proposicional, este método não se aplica a outras lógicas, como por exemplo, a lógica de predicados vista mais adiante. Por conseguinte o método

formal é, de fato, aquele possível para estudar teorias formais. Na teoria formal da lógica proposicional já é possível perceber a utilidade e abrangência desse método. No entanto, para a lógica proposicional a teoria é bastante simples.

Uma **Teoria formal** T é definida quando:

1. É especificada uma linguagem formal $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ para T, denominada **a linguagem de** T.
2. Da linguagem $Ling(\mathcal{L})$ (a linguagem especificada por $\mathcal{L}$) é destacada um conjunto Δ de elementos, chamados **axiomas** de T.
3. É especificado um conjunto $\Re$ de regras, chamadas **regras de inferência**, da forma

$$\frac{\alpha_1, \ldots, \alpha_n}{\alpha},$$

tal que a fórmula α é obtida das fórmulas $\alpha_1, \ldots, \alpha_n$. Por isso dizemos que α é uma **conseqüência direta** de $\alpha_1, \ldots, \alpha_n$.

Observe que um axioma, também pode ser entendido como uma regra de inferência sem antecedente. Por exemplo se α é um axioma, então ele representa a regra de inferência $\frac{-}{\alpha}$. Portanto poderíamos perfeitamente excluir da noção de teoria formal esta componente. No entanto, como axiomas têm um papel fundamental na teoria de provas, optamos por destacá-los.

Definição 4.1.1 *Uma* **prova** *de $\alpha \in Ling(\mathcal{L})$ na teoria formal $T = \langle \mathcal{L}, \Delta, \Re \rangle$, é uma seqüência $\alpha_1, \ldots, \alpha_n$ de elementos de $Ling(\mathcal{L})$ tal que $\alpha_n = \alpha$ e para cada $1 \leq i \leq n$, α_i ou é um axioma de T, isto é, $\alpha_i \in \Delta$, ou é obtida como conseqüência direta de alguns elementos anteriores na seqüência usando uma das regras de inferência.*

Definição 4.1.2 *Um* **teorema** *de uma teoria formal $T = \langle \mathcal{L}, \Delta, \Re \rangle$ é um elemento $\alpha \in Ling(\mathcal{L})$ tal que existe uma prova de α em T.*

O conjunto de teoremas de uma teoria formal T, denotado por $\mathcal{T}(T)$, é denominado **lógica apresentada pela teoria formal** T. Denominamos a esta forma de apresentar lógicas de "axiomática". No capítulo 10 veremos outras maneiras de apresentar uma lógica.

Definição 4.1.3 *Um elemento $\alpha \in Ling(\mathcal{L})$ diz-se uma* **conseqüência** *na teoria formal $T = \langle \mathcal{L}, \Delta, \Re \rangle$, de um conjunto Γ de elementos de $Ling(\mathcal{L})$, denotado por $\Gamma \vdash \alpha$, se existe uma seqüência $\alpha_1, \ldots, \alpha_n$ de fórmulas de $Ling(\mathcal{L})$ tal que $\alpha_n = \alpha$ e cada α_i , $1 \leq i < n$, ou é um axioma de T, ou $\alpha_i \in \Gamma$, ou é uma conseqüência direta de α_j anteriores, usando uma das regras de inferência. Neste caso a seqüência $\alpha_1, \ldots, \alpha_n$ é dita uma* **prova de** α **a partir das premissas** Γ.

Observações:

1. Se $\Gamma \vdash \alpha$ dizemos, também, que existe, na teoria T, uma prova de α a partir do conjunto de **premissas**, ou hipóteses, Γ. α é também chamado de **tese** ou **conclusão**.

2. Se $\alpha_1, \ldots, \alpha_n$ é uma prova de α a partir das premissas em Γ, então, neste texto, diremos simplesmente que $\alpha_1, \ldots, \alpha_n$ é uma prova de $\Gamma \vdash \alpha$.

3. Se Γ é finito, isto é, $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ escrevemos $\alpha_1, \ldots, \alpha_n \vdash \alpha$, em vez de $\{\alpha_1, \ldots, \alpha_n\} \vdash \alpha$.

4. Segundo esta definição, um teorema, α, de T é uma conseqüência de um conjunto vazio de premissas. Por isso um teorema α em T será denotado por $\emptyset \vdash \alpha$, ou simplesmente, $\vdash \alpha$.

5. Observe que a rigor a notação deveria ser $\Gamma \vdash_{\mathrm{T}} \alpha$, para destacar a teoria formal à qual estamos nos referindo. Como não vamos fazer um estudo de teorias formais, mas, apenas, desenvolver algumas dessas teorias, não explicitaremos o T na notação, pois ele estará, sempre, claro do contexto.

6. Da definição de conseqüência sai diretamente as seguintes propriedades: Sejam Γ, Θ subconjuntos de $Ling(\mathcal{L})$.

   (a) Monotonicidade: Se $\Gamma \subseteq \Theta$ e $\Gamma \vdash \alpha$, então $\Theta \vdash \alpha$.

   (b) Compacidade: $\Gamma \vdash \alpha$ se, e somente se, existe um subconjunto finito Γ_0 de Γ tal que $\Gamma_0 \vdash \alpha$.

   (c) Lei do corte: Se $\Theta \vdash \alpha$ e para cada $\beta \in \Theta$, $\Gamma \vdash \beta$, então $\Gamma \vdash \alpha$.

Definição 4.1.4 *Seja $T = \langle \mathcal{L}, \Delta, \Re \rangle$ uma teoria formal. As fórmulas $\alpha \in Ling(\mathcal{L})$ e $\beta \in Ling(\mathcal{L})$ são equivalentes na teoria formal T, denotado por $\alpha \equiv_T \beta$ se, $\alpha \vdash \beta$ e $\beta \vdash \alpha$.*

Duas teorias formais são ditas equivalente se há uma maneira de "traduzir´´ fórmulas de uma linguagem na outra, de tal modo que se uma fórmula for traduzida para a outra linguagem e depois retraduzida para a linguagem original, a fórmula original e a resultante deste processo são equivalentes e teoremas de uma teoria quando traduzidos para outra teoria resulta também num teorema. Formalmente,

Definição 4.1.5 *Sejam as teorias formais $T_1 = \langle \mathcal{L}_1, \Delta_1, \Re_1 \rangle$ e $T_2 = \langle \mathcal{L}_2, \Delta_2, \Re_2 \rangle$. T_1 é equivalente a T_2, denotado por $T_1 \cong T_2$, se, e somente se, existem funções $\varphi_1 : Ling(\mathcal{L}_1) \longrightarrow Ling(\mathcal{L}_2)$ e $\varphi_2 : Ling(\mathcal{L}_2) \longrightarrow Ling(\mathcal{L}_1)$ tais que $\varphi_2 \circ \varphi_1 = Id_{Ling(\mathcal{L}_1)}$, $\varphi_1 \circ \varphi_2 = Id_{Ling(\mathcal{L}_2)}$, se $\alpha \in \mathcal{T}(T_1)$ then $\varphi_1(\alpha) \in \mathcal{T}(T_2)$ e se $\alpha \in \mathcal{T}(T_2)$ then $\varphi_2(\alpha) \in \mathcal{T}(T_1)$.*

4.2 Teoria Formal da Lógica Proposicional

Estamos em condições, agora, de especificar uma teoria formal para a lógica proposicional.

Seguindo a metodologia acima, de especificação de uma teoria formal, definimos a teoria formal para a lógica proposicional como $T_P = \langle \mathcal{L}_{P_\rightarrow}, \Delta_\rightarrow, \Re_\rightarrow \rangle$, onde:

1. $\mathcal{L}_{P_\rightarrow}$: A linguagem proposicional, é a linguagem formal obtida de L_P, especificada anteriormente (seção 2.2.), usando como conectivos lógicos somente $\rightarrow$ e $\neg$. Isto é, $L_{P_\rightarrow} = Ling(\mathcal{L}_{P_\rightarrow})$, onde $\mathcal{L}_{P_\rightarrow} = \langle \Sigma_\rightarrow, \mathcal{G}_\rightarrow \rangle$, para $\Sigma_\rightarrow = \Sigma_V \cup \Sigma_P \cup \{\rightarrow, \neg\} \cup \{(,)\}$ e $\mathcal{G}_\rightarrow = \{g_1, g_2, g_5\}$.

2. $\Delta_\rightarrow$: Destacaremos o seguinte conjunto de axiomas, ou regras de inferência sem antecedentes.

   $A_1)$ $\alpha \rightarrow (\beta \rightarrow \alpha)$
   $A_2)$ $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$
   $A_3)$ $(\neg\beta \rightarrow \neg\alpha) \rightarrow ((\neg\beta \rightarrow \alpha) \rightarrow \beta)$

3. $\Re_\rightarrow$: A única regra de inferência é o **modus ponens**, denotado por MP, qual seja,

$$\mathsf{MP} : \frac{\alpha, \quad \alpha \rightarrow \beta}{\beta}$$

Observações:

1. Cada α, β e γ usada na especificação da teoria formal da lógica proposicional é uma meta-variável proposicional, isto é, ela pode ser substituída por qualquer proposição em $L_{P_\rightarrow} = Ling(\mathcal{L}_{P_\rightarrow})$.

2. Expressões envolvendo meta-variáveis são chamadas de **meta-fórmulas** ou **esquema de fórmula** se ao serem substituídas por símbolos proposicionais resultam numa fbf. As meta-variáveis dentro de uma meta-fórmula podem ser substituídas por fórmulas ou por meta-fórmulas e chamamos a elas de **instâncias** ou **meta-instâncias** da meta-fórmula, respectivamente. Por exemplo, as fórmulas $p_1 \rightarrow (p_2 \rightarrow \neg p_1)$ e $(p_1 \rightarrow p_2) \rightarrow (p_3 \rightarrow \neg(p_1 \rightarrow p_2))$ são ambas instâncias da meta-fórmula $\alpha \rightarrow (\beta \rightarrow \neg\alpha)$. Já a meta-fórmula $\neg\alpha \rightarrow ((\alpha \rightarrow \beta) \rightarrow \neg\neg\alpha)$ é uma meta-instância.

3. Assim, cada "axioma" é na verdade um **esquema de axiomas** ou um **meta-axioma**, pois descreve um conjunto infinito de axiomas. No restante deste texto usaremos o nome de axiomas tanto para esquemas de axiomas como para instâncias e meta-instâncias de esquemas de axiomas.

4. A rigor é possível especificar a linguagem da lógica proposicional usando apenas os conectivos $\neg$ e $\rightarrow$, e definindo os demais como segue:

$$\begin{aligned} \alpha \wedge \beta &\stackrel{\text{def}}{=} \neg(\alpha \rightarrow \neg\beta) \\ \alpha \vee \beta &\stackrel{\text{def}}{=} \neg\alpha \rightarrow \beta \\ \alpha \leftrightarrow \beta &\stackrel{\text{def}}{=} (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \end{aligned}$$

   Assim, o fato de usar a linguagem formal $\mathcal{L}_{P_\rightarrow}$ em vez de $\mathcal{L}_P$ não prejudica a generalidade da teoria formal da lógica proposicional aqui proposta.

Proposição 4.2.1 *Para toda fórmula $\alpha \in L_{P_\rightarrow}$, $\vdash \alpha \rightarrow \alpha$. Ou seja, $\alpha \rightarrow \alpha$ é um teorema da teoria formal da lógica proposicional T_P.*

Demonstração: Para demonstrar esta proposição, deveremos exibir uma prova na teoria formal T_P da lógica proposicional. A seqüência $\alpha_1, \ldots, \alpha_5$, onde

1. α_1 é a instância $(\alpha \rightarrow ((\alpha \rightarrow \alpha) \rightarrow \alpha)) \rightarrow ((\alpha \rightarrow (\alpha \rightarrow \alpha)) \rightarrow (\alpha \rightarrow \alpha))$ de A_2. Para ver isto, basta substituir β por $\alpha \rightarrow \alpha$ e γ por α em A_2,

2. α_2 é a instância $\alpha \rightarrow ((\alpha \rightarrow \alpha) \rightarrow \alpha)$ de A_1. Para ver isto basta substituir β por $\alpha \rightarrow \alpha$ em A_1,

3. α_3 é $(\alpha \rightarrow (\alpha \rightarrow \alpha)) \rightarrow (\alpha \rightarrow \alpha)$, a qual resulta da aplicação da regra de inferência modus ponens (MP) a α_1 e α_2,

4. α_4 é a instância $\alpha \rightarrow (\alpha \rightarrow \alpha)$ de A_1, a qual resulta de substituir β por α em A_1,

5. α_5 é $\alpha \rightarrow \alpha$, a qual resulta da aplicação da regra de inferência modus ponens (MP) a α_3 e α_4,

é uma prova de $\alpha \rightarrow \alpha$. ■

Quando é provado que uma certa fórmula é um teorema, esse conhecimento pode ser usado em outras provas da mesma forma como são utilizados os axiomas da teoria.

Exemplo 4.2.2 *A seguinte seqüência de fórmulas*

α_1.	$(\neg\alpha \rightarrow \neg\alpha) \rightarrow ((\neg\alpha \rightarrow \alpha) \rightarrow \alpha)$	*instância de A_3*
α_2.	$\neg\alpha \rightarrow \neg\alpha$	*proposição 4.2.1*
α_3.	$(\neg\alpha \rightarrow \alpha) \rightarrow \alpha$	MP *de α_2 e α_1*
α_4.	$\neg\alpha \rightarrow \alpha$	*hipóteses*
α_5.	α	MP *de α_4 e α_3*

poderia ser vista como uma abreviação de

α_1.	$(\neg\alpha \to \neg\alpha) \to ((\neg\alpha \to \alpha) \to \alpha)$	*instância de* A_3
α_{2a}.	$(\neg\alpha \to ((\neg\alpha \to \neg\alpha) \to \neg\alpha)) \to ((\neg\alpha \to (\neg\alpha \to \neg\alpha)) \to (\neg\alpha \to \neg\alpha))$	*instância de* A_2
α_{2b}.	$\neg\alpha \to ((\neg\alpha \to \neg\alpha) \to \neg\alpha)$	*instância de* A_1
α_{2c}.	$(\neg\alpha \to (\neg\alpha \to \neg\alpha)) \to (\neg\alpha \to \neg\alpha)$	MP *de* α_{2b} *e* α_{2a}
α_{2d}.	$\neg\alpha \to (\neg\alpha \to \neg\alpha)$	*instância de* A_1
α_{2e}.	$\neg\alpha \to \neg\alpha$	MP *de* α_{2d} *e* α_{2c}
α_3.	$(\neg\alpha \to \alpha) \to \alpha$	MP *de* α_{2e} *e* α_1
α_4.	$\neg\alpha \to \alpha$	*hipóteses*
α_5.	α	MP *de* α_4 *e* α_3

que é uma prova de $\neg\alpha \to \alpha \vdash \alpha$. *Observe que o único realizado foi substituir na prova de* $\vdash \alpha \to \alpha$ *todas as ocorrências de* α *por* $\neg\alpha$, *para assim obter uma prova de* $\vdash \neg\alpha \to \neg\alpha$ *e em seguida substituir* α_2 *por esta prova.*

4.3 Teorema da Dedução (Sintático)

Apesar de ser simples e intuitivo que a proposição $\alpha \to \alpha$ é um teorema de T_P, a sua demonstração, isto é, a exibição de uma prova em T_P, não foi tão simples assim. É de se prever que na medida em que as proposições em $L_{P_\to}$ se tornem mais complexas, demonstrar que elas são teoremas de T_P fique cada vez mais difícil. Por isso, a seguir demonstraremos um teorema na nossa meta-teoria (a matemática usada para desenvolver a teoria formal T_P) que facilitará a exibição de provas na teoria T_P. Queremos alertar o leitor que estaremos usando o nome **teorema** tanto para um resultado da teoria objeto, no caso a teoria formal T_P, como para um resultado da matemática (meta-teoria) que estamos usando para fundamentar essa teoria formal. Esperamos que o leitor não confunda uma linguagem com uma meta-linguagem.

Teorema 4.3.1 (da Dedução sintático) *Sejam* $\alpha, \beta \in L_{P_\to}$ *e* $\Gamma \subseteq L_{P_\to}$. $\Gamma, \alpha \vdash \beta$ *se e somente se* $\Gamma \vdash \alpha \to \beta$. *Em particular, para* $\Gamma = \emptyset$, $\alpha \vdash \beta$ *se e somente se* $\vdash \alpha \to \beta$.

Demonstração: Suponha que $\Gamma, \alpha \vdash \beta$. Isto é, suponha que existe uma prova $\alpha_1, \ldots, \alpha_n$ de β a partir do conjunto de premissas $\Gamma \cup \{\alpha\}$. Então $\beta = \alpha_n$. Usaremos indução sobre i para mostrar que se for o caso de $\Gamma, \alpha \vdash \alpha_i$, então $\Gamma \vdash \alpha \to \alpha_i$, para todo i, $1 \leq i \leq n$.

Caso base: $i = 1$. Então α_1 deve ser ou um axioma ou uma premissa.

- Se α_1 é um axioma:

  1. α_1. Por definição
  2. $\alpha_1 \to (\alpha \to \alpha_1)$. Uma instância de A_1, substituindo α por α_1 e β por α.
  3. $\alpha \to \alpha_1$. De 1) e 2) por MP.

Logo, α_1, $\alpha_1 \to (\alpha \to \alpha_1)$, $\alpha \to \alpha_1$ é uma prova de $\alpha \to \alpha_1$ a partir de Γ, isto é, $\Gamma \vdash \alpha \to \alpha_1$.

- Se α_1 é uma premissa:

  a) Se $\alpha_1 \in \Gamma$:

  1. α_1. Por definição de prova
  2. $\alpha_1 \to (\alpha \to \alpha_1)$. Uma instância de A_1
  3. $\alpha \to \alpha_1$. De 1) e 2) por MP.

  Logo, α_1, $\alpha_1 \to (\alpha \to \alpha_1)$, $\alpha \to \alpha_1$ é uma prova de $\alpha \to \alpha_1$ a partir de Γ, isto é, $\Gamma \vdash \alpha \to \alpha_1$.

  b) Se α_1 é α então $\alpha \to \alpha_1$ é $\alpha \to \alpha$ que foi provado na proposição 4.2.1. Logo, $\vdash \alpha \to \alpha_1$ e pelo teorema da monotonicidade (observação 6 da seção 1 deste capítulo) inerente a qualquer teoria formal, temos que $\Gamma \vdash \alpha \to \alpha_1$.

Etapa indutiva: Suponha que o teorema é verdadeiro para $i < k$. Então, por definição de prova.

- α_k é um axioma, ou
- α_k é uma premissa

  a) $\alpha_k \in \Gamma$, ou

  b) α_k é α, ou

- α_k segue por MP de α_i e α_j, onde $i, j < k$, e α_j é $\alpha_i \to \alpha_k$.

Os dois primeiros casos são tratados exatamente como no caso base acima.

Consideremos o terceiro caso.

1. $\alpha \to (\alpha_i \to \alpha_k)$. Hipótese da indução
2. $\alpha \to \alpha_i$. Hipótese da indução
3. $(\alpha \to (\alpha_i \to \alpha_k)) \to ((\alpha \to \alpha_i) \to (\alpha \to \alpha_k))$ instância de A_2.
4. $(\alpha \to \alpha_i) \to (\alpha \to \alpha_k)$. De 1) e 3) por MP.
5. $\alpha \to \alpha_k$. De 2) e 4) por MP.

Logo, $\alpha \to (\alpha_i \to \alpha_k)$, $\alpha \to \alpha_i$, $(\alpha \to (\alpha_i \to \alpha_k)) \to ((\alpha \to \alpha_i) \to (\alpha \to \alpha_k))$, $(\alpha \to \alpha_i) \to (\alpha \to \alpha_k)$, $\alpha \to \alpha_k$ é uma prova de $\alpha \to \alpha_k$ a partir de Γ, isto é $\Gamma \vdash \alpha \to \alpha_k$.

Portanto, $\Gamma \vdash \alpha \to \beta$.

Por outro lado se $\Gamma \vdash \alpha \to \beta$, então existe uma prova $\alpha_1, \ldots, \alpha_n$ (com $\alpha_n = \alpha \to \beta$) de $\alpha \to \beta$ a partir de Γ. Claramente a seqüência $\alpha_1, \ldots, \alpha_n, \alpha$ (hip.), β (MP, α, α_n), é uma prova de $\Gamma, \alpha \vdash \beta$. ■

Observe que a demonstração da primeira parte do teorema da dedução nos dá um algoritmo para transformar qualquer prova de $\Gamma, \alpha \vdash \beta$ numa prova de $\Gamma \vdash \alpha \to \beta$.

Para mostrar o poder de simplificação do teorema da dedução provaremos, usando esse teorema, a proposição 4.2.1, ou seja provaremos que $\vdash \alpha \to \alpha$, para cada $\alpha \in L_{P_\to}$:

α_1. α. hipóteses.

Logo, α_1 é uma prova de α a partir de α, e por tanto $\alpha \vdash \alpha$. A demonstração do teorema da dedução nos garante que existe uma prova de $\vdash \alpha \to \alpha$ (neste caso a própria prova da proposição 4.2.1).

Proposição 4.3.2

i) $\alpha \to \beta, \beta \to \gamma \vdash \alpha \to \gamma$

ii) $\alpha \to (\beta \to \gamma), \beta \vdash \alpha \to \gamma$

Demonstração:

i)

α_1.	$\alpha \to \beta$	hipóteses
α_2.	$\beta \to \gamma$	hipóteses
α_3.	α	hipóteses
α_4.	β	MP de α_1 e α_3.
α_5.	γ	MP de α_2 e α_4.

Mostramos, então que $\alpha \to \beta, \beta \to \gamma, \alpha \vdash \gamma$. Usando o teorema da dedução, temos que esta prova pode ser transformada numa prova de $\alpha \to \beta, \beta \to \gamma \vdash \alpha \to \gamma$, como desejávamos.

ii)

α_1.	$\alpha \to (\beta \to \gamma)$	hipóteses
α_2.	β	hipóteses
α_3.	α	hipóteses
α_4.	$\beta \to \gamma$	MP de α_1 e α_3
α_5.	γ	MP de α_2 e α_4

Assim, provamos que $\alpha \to (\beta \to \gamma), \beta, \alpha \vdash \gamma$. Logo, aplicando o teorema da dedução teremos uma prova para $\alpha \to (\beta \to \gamma), \beta \vdash \alpha \to \gamma$. ■

Observe que na demonstração da proposição 4.3.2.i e ii, usamos o teorema da dedução. Ou seja apresentamos uma prova de $\alpha \to \beta, \beta \to \gamma, \alpha \vdash \gamma$ e argumentamos que por causa do teorema da dedução é possível transformar essa prova numa prova de $\alpha \to \beta, \beta \to \gamma \vdash \alpha \to \gamma$. Mas como podemos fazer isso? utilizando o processo indutivo da demonstração deste teorema.

A seguir mostraremos qual seria o resultado de utilizar esta indução à prova de $\alpha \to \beta, \beta \to \gamma \vdash \alpha \to \gamma$.

α_{1a}.	$\alpha \to \beta$	hipóteses
α_{1b}.	$(\alpha \to \beta) \to (\alpha \to (\alpha \to \beta))$	instância de A_1
α_{1c}.	$\alpha \to (\alpha \to \beta)$	MP de α_{1a} e α_{1b}
α_{2a}.	$\beta \to \gamma$	hipóteses
α_{2b}.	$(\beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$	instância de A_1
α_{2c}.	$\alpha \to (\beta \to \gamma)$MP de α_{2a} e α_{2b}	
α_{3a}.	$(\alpha \to ((\alpha \to \alpha) \to \alpha)) \to ((\alpha \to (\alpha \to \alpha)) \to (\alpha \to \alpha))$	instância de A_2
α_{3b}.	$\alpha \to ((\alpha \to \alpha) \to \alpha)$	instância de A_1
α_{3c}.	$(\alpha \to (\alpha \to \alpha)) \to (\alpha \to \alpha)$	MP de α_{3b} e α_{3a}
α_{3d}.	$\alpha \to (\alpha \to \alpha)$	instância de A_1
α_{3e}.	$\alpha \to \alpha$	MP de α_{3d} e α_{3c}
α_{4a}.	$(\alpha \to (\alpha \to \beta)) \to ((\alpha \to \alpha) \to (\alpha \to \beta))$	instância de A_2
α_{4b}.	$(\alpha \to \alpha) \to (\alpha \to \beta)$	MP de α_{1c} e α_{4a}
α_{4c}.	$\alpha \to \beta$	MP de α_{3e} e α_{4b}
α_{5a}.	$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	instância de A_2
α_{5b}.	$(\alpha \to \beta) \to (\alpha \to \gamma)$	MP de α_{2c} e α_{5a}
α_{5c}.	$\alpha \to \gamma$	MP de α_{4c} e α_{5b}

Proposição 4.3.3 *Para todo $\alpha, \beta, \gamma \in L_{P\to}$ os seguintes são teoremas de T_P.*

a)	$\neg\neg\alpha \to \alpha$	*e)*	$(\alpha \to \neg\beta) \to (\beta \to \neg\alpha)$
b)	$\alpha \to \neg\neg\alpha$	*f)*	$\alpha \to (\neg\beta \to \neg(\alpha \to \beta))$
c)	$\neg\alpha \to (\alpha \to \beta)$	*g)*	$(\alpha \to \beta) \to ((\neg\alpha \to \beta) \to \beta)$
d)	$(\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$	*h)*	$(\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$

Demonstração:

a) $\vdash \neg\neg\alpha \to \alpha$

α_1.	$(\neg\alpha \to \neg\neg\alpha) \to ((\neg\alpha \to \neg\alpha) \to \alpha)$	instância de A_3
α_2.	$\neg\alpha \to \neg\alpha$	proposição 4.2.1
α_3.	$(\neg\alpha \to \neg\neg\alpha) \to \alpha$	proposição 4.3.2.ii de α_1 e α_2
α_4.	$\neg\neg\alpha \to (\neg\alpha \to \neg\neg\alpha)$	instância de A_1
α_5.	$\neg\neg\alpha \to \alpha$	proposição 4.3.2.i de α_3 e α_4

b) $\vdash \alpha \to \neg\neg\alpha$

α_1.	$(\neg\neg\neg\alpha \to \neg\alpha) \to ((\neg\neg\neg\alpha \to \alpha) \to \neg\neg\alpha)$	instância de A_3
α_2.	$\neg\neg\neg\alpha \to \neg\alpha$	item a)
α_3.	$(\neg\neg\neg\alpha \to \alpha) \to \neg\neg\alpha$	MP de α_1 e α_2
α_4.	$\alpha \to (\neg\neg\neg\alpha \to \alpha)$	instância de A_1
α_5.	$\alpha \to \neg\neg\alpha$	proposição 4.3.2.i de α_3 e α_4

c) $\vdash \neg\alpha \to (\alpha \to \beta)$

α_1.	$\neg\alpha$	hipótese
α_2.	α	hipótese
α_3.	$\alpha \to (\neg\beta \to \alpha)$	instância de A_1
α_4.	$\neg\alpha \to (\neg\beta \to \neg\alpha)$	instância A_1
α_5.	$\neg\beta \to \alpha$	MP de α_2 e α_3
α_6.	$\neg\beta \to \neg\alpha$	MP de α_1 e α_4
α_7.	$(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$	instância de A_3
α_8.	$(\neg\beta \to \alpha) \to \beta$	MP de α_6 e α_7
α_9.	β	MP de α_5 e α_8

Assim, $\neg\alpha, \alpha \vdash \beta$. Portanto, aplicando o teorema da dedução duas vezes obteremos uma prova de $\vdash \neg\alpha \to (\alpha \to \beta)$.

d) $\vdash (\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$

α_1.	$\neg\beta \to \neg\alpha$	hipótese
α_2.	$(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$	instância A_3
α_3.	$\alpha \to (\neg\beta \to \alpha)$	instância de A_1
α_4.	$(\neg\beta \to \alpha) \to \beta$	MP de α_1 e α_2
α_5.	$\alpha \to \beta$	Proposição 4.3.2.i de α_3 e α_4

Assim, $\neg\beta \to \neg\alpha \vdash \alpha \to \beta$. Portanto, aplicando o teorema da dedução obtemos uma prova de $\vdash (\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$

e) $\vdash (\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$

α_1.	$\alpha \to \beta$	hipótese
α_2.	$\neg\neg\alpha \to \alpha$	item a)
α_3.	$\neg\neg\alpha \to \beta$	4.3.2.i de α_1 e α_2
α_4.	$\beta \to \neg\neg\beta$	item b)
α_5.	$\neg\neg\alpha \to \neg\neg\beta$	4.3.2.i de α_3 e α_4
α_6.	$(\neg\neg\alpha \to \neg\neg\beta) \to (\neg\beta \to \neg\alpha)$	item d)
α_7.	$\neg\beta \to \neg\alpha$	MP de α_5 e α_6

Assim, $(\alpha \to \beta) \vdash \neg\beta \to \neg\alpha$ e pelo teorema da dedução segue que $\vdash (\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$.

f) $\vdash \alpha \to (\neg\beta \to \neg(\alpha \to \beta))$

Claramente, por MP, $\alpha, \alpha \to \beta \vdash \beta$. Assim, usando duas vezes o teorema da dedução, temos $\vdash \alpha \to ((\alpha \to \beta) \to \beta)$. Pelo item e), temos $\vdash ((\alpha \to \beta) \to \beta) \to (\neg\beta \to \neg(\alpha \to \beta))$. Portanto, usando a proposição 4.3.2.i, obtemos

$$\vdash \alpha \to (\neg\beta \to \neg(\alpha \to \beta)) \quad \blacksquare$$

g) $\vdash (\alpha \to \beta) \to ((\neg\alpha \to \beta) \to \beta)$

α_1.	$\alpha \to \beta$	hipótese
α_2.	$\neg\alpha \to \beta$	hipótese
α_3.	$(\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$	item e)
α_4.	$\neg\beta \to \neg\alpha$	MP de α_1 e α_3
α_5.	$(\neg\alpha \to \beta) \to (\neg\beta \to \neg\neg\alpha)$	item e)
α_6.	$\neg\beta \to \neg\neg\alpha$	MP de α_2 e α_5
α_7.	$(\neg\beta \to \neg\neg\alpha) \to ((\neg\beta \to \neg\alpha) \to \beta)$	instância A_3
α_8.	$(\neg\beta \to \neg\alpha) \to \beta$	MP de α_6 e α_7
α_9.	β	MP de α_4 e α_8

Assim, $\alpha \to \beta, \neg\alpha \to \beta \vdash \beta$. Duas aplicações do teorema da dedução levam a uma prova de $\vdash (\alpha \to \beta) \to ((\neg\alpha \to \beta) \to \beta)$.

h) $\vdash (\alpha \to \neg\beta) \to (\beta \to \neg\alpha)$

α_1.	$\alpha \to \neg\beta$	hipótese
α_2.	$\neg\neg\alpha \to \alpha$	item a)
α_3.	$\neg\neg\alpha \to \neg\beta$	4.3.2.i de α_1 e α_2
α_4.	$(\neg\neg\alpha \to \neg\beta) \to (\beta \to \neg\alpha)$	item d)
α_5.	$\beta \to \neg\alpha$	MP de α_3 e α_4

Assim, $(\alpha \to \neg\beta) \vdash \beta \to \neg\alpha$. Portanto, aplicando o teorema da dedução obtemos uma prova de $\vdash (\alpha \to \neg\beta) \to (\beta \to \neg\alpha)$.

As vezes é interessante demonstrar meta-teoremas da teoria, tipo o teorema da dedução, isto é onde a partir de uma ou mais conseqüências sintáticas seja possível provar uma outra conseqüência sintática.

Exemplo 4.3.4 *Seja o seguinte meta-teorema: se* $\Gamma, \alpha \vdash \beta$ *e* $\Gamma \vdash \alpha$ *então* $\Gamma \vdash \beta$.

Uma demonstração para este meta-teorema seria a seguinte:

Se $\Gamma, \alpha \vdash \beta$ *e* $\Gamma \vdash \alpha$ *então existem provas* $\alpha_1, \ldots, \alpha_m$ *e* $\beta_1, \ldots, \beta_n$ *de* β *a partir de* Γ *e* α *e de* α *a partir de* Γ*, respectivamente. Se a hipóteses* α*, não foi usada na prova* $\alpha_1, \ldots, \alpha_m$*, então, trivialmente,* $\alpha_1, \ldots, \alpha_m$ *também é uma prova para* $\Gamma \vdash \alpha$*. Se a hipóteses* α *foi usada na prova, suponha que em* α_i*, então, trivialmente, a seqüência* $\alpha_1, \ldots, \alpha_{i-1}, \beta_1, \ldots, \beta_n, \alpha_{i+1}, \ldots, \alpha_m$ *seria uma prova de* $\Gamma \vdash \beta$.

Exemplo 4.3.5 *A seguir demonstraremos que se* $\vdash \alpha \to \beta$ *e* $\vdash \neg\beta$ *então* $\vdash \neg\alpha$.

Suponha que $\alpha_1, \ldots, \alpha_m$ *é uma prova de* $\vdash \alpha \to \beta$ *(portanto* α_m *é* $\alpha \to \beta$*) e que* $\beta_1, \ldots, \beta_n$ *é uma prova de* $\vdash \neg\beta$ *e portanto* β_n *é* $\neg\beta$*. Então a seguinte seqüência de fórmulas:*

α_1		
$\vdots$		
α_m	$\alpha \to \beta$	
α_{m+1}	β_1	
$\vdots$		
α_{m+n-1}	β_{n-1}	
α_{m+n}	$\neg\beta$	
α_{m+n+1}	$(\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$	*Proposição 4.3.3.h*
α_{m+n+2}	$\neg\beta \to \neg\alpha$	MP *de* α_m *e* α_{m+n+1}
α_{m+n+3}	$\neg\alpha$	MP *de* α_{m+n} *e* α_{m+n+2}

é uma prova de $\vdash \neg\alpha$.

Pode-se dizer que a lógica clássica é a ciência do argumento. Os sofistas, na Grécia antigua, eram uma espécie de professores, que ensinavam a arte de argumentar a favor ou em contra qualquer afirmação, mesmo não sendo necessariamente correta, daí que hoje em dia se usa o termo "sofismo" de uma forma pejorativa. Essa arte que os sofistas ensinavam era de suma importância na época, pois se alguém fosse acusado de algum delito, ele mesmo deveria apresentar sua defesa perante um tribunal, mesmo que outra pessoa fizesse essa defesa. Considerando que os juices eram pessoas leigas escolhidos aleatoriamente, eram facilmente influenciáveis por argumentos bem construídos, a arte do sofismo era vital.

O mais famoso dos sofistas era Protágoras, quem nasceu em Abdera ao redor do ano 420 a.C. e morreu aproximadamente no ano 480 a.C. Um dos discípulos mais brilhante de Protágoras nesta arte foi Euathlus. Conta a história que Protágoras fez o seguinte trato com Euathlus: ele lhe ensinaria a arte do sofismo por uma certa quantia, metade da qual seria paga no inicio e a outra metade quando Euathlus ganhasse seu primeiro caso. No entanto, Euathlus demorou a pagar a segunda metade a Protágoras, motivo pelo qual Protágoras processou seu discípulo. Protágoras fez perante o tribunal o seguinte argumento:

"Se Euathlus ganhasse o caso, ganharia então seu primeiro caso, logo deveria pagá-lo. Mas, se Euathlus perdesse o caso, também deveria pagá-lo, pois era esse o motivo do processo"

Vejamos se o argumento de Protágoras está correto. Primeiro vejamos quais as afirmações (proposições) atômicas:

- $p =$ "Euathlus ganha o caso"

- $q =$ "Euathlus ganha seu primeiro caso"
- $r =$ "Euathlus deve pagar a Protágoras"

E o argumento de Protágoras propriamente foi o seguinte

$$\frac{\begin{array}{c} p \to q \\ q \to r \\ \neg p \to r \end{array}}{r}$$

que pode ser entendido como: se $p \to q$ e $q \to r$ e $\neg p \to r$ então r.

Assim, se o argumento de Protágoras está correto então é possível provar que

$$p \to q, q \to r, \neg p \to r \vdash r$$

Uma tal prova é a seguinte:

α_1.	$p \to q$	hipóteses
α_2.	$q \to r$	hipóteses
α_3.	$\neg p \to r$	hipóteses
α_4.	$p \to r$	proposição 4.3.2.i de α_1 e α_2
α_5.	$(p \to r) \to ((\neg p \to r) \to r)$	proposição 4.3.3.g
α_6.	$(\neg p \to r) \to r$	MP de α_4 e α_5
α_7.	r	MP de α_3 e α_6

Portanto o argumento de Protágoras está correto. Porém, mesmo sendo correto o argumento de Protágoras, Euathlus conseguiu também dar um outro argumento correto:

"Se ele ganhasse o caso, não deveria pagar, pois Protágoras perderia a causa. Mas se ele perdesse o caso, então ainda não teria ganho o seu primeiro caso e, portanto, não deveria pagar a Protágoras".

Numa linguagem lógica usando as proposições atômicas acima, seria :

$$\frac{\begin{array}{c} p \to \neg r \\ \neg p \to \neg q \\ \neg q \to \neg r \end{array}}{\neg r}$$

Se o argumento de Euathlus está correto então é possível provar que

$$p \to \neg r, \neg p \to \neg q, \neg p \to \neg r \vdash \neg r$$

Uma prova disto é a seguinte:

α_1.	$p \to \neg r$	hipóteses
α_2.	$\neg p \to \neg q$	hipóteses
α_3.	$\neg q \to \neg r$	hipóteses
α_4.	$\neg p \to \neg r$	proposição 4.3.2.i de α_2 e α_3
α_5.	$(p \to \neg r) \to ((\neg p \to \neg r) \to \neg r)$	proposição 4.3.3.g
α_6.	$(\neg p \to \neg r) \to \neg r$	MP de α_1 e α_5
α_7.	r	MP de α_4 e α_6

Portanto, o argumento de Euathlus também está correto. Logo, a lógica proposicional é inconclusiva, porém não errada pois o que ela mostra é que o argumento dado esteja logicamente correto, más não verifica por exemplo a validade das premissas utilizadas. Assim a pergunta sobre quem deveria ter ganho o caso permanece tão em suspenso quanto naquela época.

Uma possível maneira de sair deste impasse seria que Protágoras processasse Euathlus por um motivo qualquer (calunia, por exemplo) e se deixasse ganhar para depois entrar novamente contra Euathlus, está vez por não pagamento e argumentando que Euathlus já tinha ganho um caso.

4.4 Outras Teorias Formais para a Lógica Proposicional

A teoria formal proposicional T_P, vista anteriormente, é só uma entre muitas possíveis teorias formais para a lógica proposicional. A seguir destacaremos outras teorias formais para L_P.

4.4.1 Teoria Formal 1

Nesta teoria formal, denotada por T_{P_T}, a linguagem proposicional é a mesma (baseada nos conectivos $\neg$ e $\to$), os esquemas de axiomas os mesmos (A_1, A_2 e A_3), mas a regra de inferência é o *modus tollens* denotado por MT, qual seja,

$$\mathsf{MT}: \frac{\alpha \to \beta, \quad \neg\beta}{\neg\alpha}$$

A justificativa desta regra é o seguinte raciocínio: Se conhecemos que quando α é verdadeiro β também é verdadeiro e se sabemos, também que β é falso, então α não pode ser verdadeiro, e portanto podemos concluir que α é falso. Observe que a regra modus tollens pode ser obtida a partir do próprio modus ponens e do teorema $(\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$ (proposição 4.3.3.h).

4.4.2 Teoria Formal 2

Meridith em 1953 [Mer53], usou uma teoria formal alternativa para a teoria formal estudada aqui. Esta teoria formal tem a mesma linguagem formal (baseada somente nos conectivos primitivos $\neg$ e $\rightarrow$) e a mesma regra de inferência (modus ponens), mas somente um único esquema de axiomas:

$$A_U) \quad (\alpha \rightarrow \beta \rightarrow (\neg\gamma \rightarrow \neg\psi)) \rightarrow \gamma \rightarrow \sigma \rightarrow (\sigma \rightarrow \alpha \rightarrow (\psi \rightarrow \alpha))$$

4.4.3 Teoria Formal 3

No livro [HA50], os matemáticos David Hilbert (1862-1943) e Wilhelm Ackermann (1896-1962) desenvolveram a teoria formal $T_{P\vee}$ a qual usa como conectivos primitivos os conectivos lógicos $\vee$ e $\neg$. Assim, a linguagem formal, esquemas de axiomas e regras de inferências da teoria formal $T_{P\vee}$ são:

1. Linguagem Proposicional: $Ling(\langle \Sigma_\vee, \mathcal{G}_\vee \rangle) \subseteq L_P$ onde $\Sigma_\vee = \Sigma_V \cup \{\vee, \neg\} \cup \Sigma_P$ e $\mathcal{G}_\vee = \{g_1, g_2, g_4\}$.

2. Axiomas:

   $A_{\vee_1}) \ \neg(\alpha \vee \alpha) \vee \alpha$

   $A_{\vee_2}) \ \neg\alpha \vee (\alpha \vee \beta)$

   $A_{\vee_3}) \ \neg(\alpha \vee \beta) \vee (\beta \vee \alpha)$

   $A_{\vee_4}) \ \neg(\neg\beta \vee \gamma) \vee (\neg(\alpha \vee \beta) \vee (\alpha \vee \gamma))$

3. Regras de Inferência: A única regra de inferência é um tipo de modus ponens, denotado por $\mathsf{MP}_\vee$, qual seja,

$$\mathsf{MP}_\vee : \frac{\alpha, \quad \neg\alpha \vee \beta}{\beta}$$

Observe que usando a $\alpha \rightarrow \beta$ como uma abreviação de $\neg\alpha \vee \beta$ os esquemas de axiomas podem ser re-escritos como $\alpha \vee \alpha \rightarrow \alpha$, $\alpha \rightarrow \alpha \vee \beta$, $\alpha \vee \beta \rightarrow \beta \vee \alpha$, $(\beta \rightarrow \gamma) \rightarrow (\alpha \vee \beta \rightarrow \alpha \vee \gamma)$, respectivamente, e a regra de inferência $\mathsf{MP}_\vee$ seria o próprio modus ponens.

4.4.4 Teoria Formal 4

No livro [Ros53], John Barkley Rosser (1907-1989) introduz uma teoria formal, $T_{P\wedge}$, para a lógica proposicional tendo como conectivos primitivos os conectivos lógicos $\wedge$ e $\neg$. A linguagem formal, o conjunto de esquemas de axiomas e o conjunto de regras de inferência da teoria formal $T_{P\wedge}$ é a seguinte :

1. Linguagem Proposicional: $Ling(\langle \Sigma_\wedge, \mathcal{G}_\wedge \rangle) \subseteq L_P$ onde $\Sigma_\wedge = V \cup \{\wedge, \neg\} \cup P$ e $\mathcal{G}_\wedge = \{g_1, g_2, g_3\}$.
2. Axiomas:

   $A_{\wedge_1}$) $\neg(\alpha \wedge \neg(\alpha \wedge \alpha))$
   $A_{\wedge_2}$) $\neg((\alpha \wedge \beta) \wedge \neg\alpha)$
   $A_{\wedge_3}$) $\neg(\neg(\alpha \wedge \neg\beta) \wedge \neg(\neg(\neg(\beta \wedge \gamma) \wedge \neg\neg(\gamma \wedge \alpha))))$

3. Regras de Inferência: A única regra de inferência é um tipo de modus ponens, denotado por $\mathsf{MP}_\wedge$, qual seja,

$$\mathsf{MP}_\wedge : \frac{\alpha, \quad \neg(\alpha \wedge \neg\beta)}{\beta}$$

Observe que usando a $\alpha \to \beta$ como uma abreviação de $\neg(\alpha \wedge \neg\beta)$ os esquemas de axiomas podem ser re-escritos como $\alpha \to \beta \wedge \alpha$, $\alpha \wedge \beta \to \alpha$, $(\alpha \to \beta) \to (\neg(\beta \wedge \gamma) \to \neg(\gamma \wedge \alpha))$, respectivamente, e a regra de inferência $\mathsf{MP}_\wedge$ seria o próprio modus ponens.

4.4.5 Teoria Formal 5

Em [Kle52], Stephen Kleene (1909-1994), baseado nas teorias formais T, $T_{P\vee}$ e $T_{P\wedge}$, discute uma teoria formal, $\widehat{T_P}$, para a lógica de predicados que considera todos os conectivos como primitivos. A linguagem formal, os esquemas de axiomas e o conjunto de regras de inferências da teoria formal $\widehat{T_P}$ são:

1. Linguagem Proposicional: L_P
2. Axiomas:

   $\widehat{A}_1$) $\alpha \to (\beta \to \alpha)$
   $\widehat{A}_2$) $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
   $\widehat{A}_3$) $(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$
   $\widehat{A}_4$) $\alpha \wedge \beta \to \beta$

$\widehat{A}_5$) $\alpha \to (\beta \to (\alpha \wedge \beta))$
$\widehat{A}_6$) $\alpha \to (\alpha \vee \beta)$
$\widehat{A}_7$) $\beta \to (\alpha \vee \beta)$
$\widehat{A}_8$) $(\alpha \to \gamma) \to ((\beta \to \gamma) \to (\alpha \vee \beta \to \gamma))$
$\widehat{A}_9$) $(\alpha \to \beta) \to ((\alpha \to \neg\beta) \to \neg\alpha)$
$\widehat{A}_{10}$) $\neg\neg\alpha \to \alpha$
$\widehat{A}_{11}$) $(\alpha \leftrightarrow \beta) \to ((\alpha \to \beta) \wedge (\beta \to \alpha))$
$\widehat{A}_{12}$) $((\alpha \to \beta) \wedge (\beta \to \alpha)) \to (\alpha \leftrightarrow \beta)$

3. Regras de Inferência: modus ponens

4.5 Exercícios

1. Provar os seguintes resultados na teoria formal T_P.

   (a) $\alpha \vdash \beta \to (\neg\alpha \to \gamma)$
   (b) $\vdash (\beta \to \alpha) \to ((\neg\alpha \to \beta) \to \alpha)$
   (c) $\alpha, \beta \vdash \neg(\alpha \to \neg\beta)$ (isto é, $\alpha, \beta \vdash \alpha \wedge \beta$)
   (d) $\neg(\alpha \to \neg\beta) \vdash \beta$ (isto é, $\alpha \wedge \beta \vdash \beta$)
   (e) $\neg(\alpha \to \neg\beta) \vdash \alpha$ (isto é, $\alpha \wedge \beta \vdash \alpha$)
   (f) $\vdash (\neg\alpha \to \beta) \to (\neg\beta \to \alpha)$

2. Mostre que:

   (a) Se $\alpha, \beta \vdash \gamma$, então $\neg(\alpha \to \neg\beta) \vdash \gamma$ (isto é, $\alpha \wedge \beta \vdash \gamma$)
   (b) $\alpha \vdash \neg\neg(\alpha \to \neg\beta) \to \neg\beta$ (isto é, $\alpha \vdash \neg(\alpha \wedge \beta) \to \neg\beta$)
   (c) Se $\vdash \alpha \to \beta$ e $\vdash \alpha \to \gamma$, então $\vdash \alpha \to \neg(\beta \to \neg\gamma)$ (isto é, $\vdash \alpha \to (\beta \wedge \gamma)$)
   (d) Se $\Gamma \vdash \alpha$ então $\Gamma \vdash \neg\alpha \to \beta$ (isto é, $\Gamma \vdash \neg\neg\alpha \vee \beta$)
   (e) $\Gamma \vdash \alpha$ e $\Theta \vdash \alpha \to \beta$ então $\Gamma \cup \Theta \vdash \beta$.
   (f) Se $\Gamma \vdash \alpha \to \beta$ e $\Gamma \vdash \neg\alpha \to \gamma$ então $\Gamma \vdash \beta$ ou $\Gamma \vdash \gamma$
   (g) Se $\Gamma \vdash \beta$ ou $\Gamma \vdash \gamma$ então $\Gamma \vdash \neg\beta \to \gamma$

3. Enuncie os respectivos teoremas da dedução para as teorias formais alternativas $T_{P\vee}$ e $T_{P\wedge}$.

4. Considere $\alpha \to \beta$ como uma abreviação da fórmula $\neg\alpha \vee \beta$ e prove os seguintes resultados na teoria formal $T_{P\vee}$:

(a) $\alpha \to \beta \vdash_{T_{P\vee}} \gamma \vee \alpha \to \gamma \vee \beta$

(b) $\vdash_{T_{P\vee}} (\alpha \to \beta) \to ((\gamma \to \alpha) \to (\gamma \to \beta))$

(c) $\alpha \to \beta, \beta \to \gamma \vdash_{T_{P\vee}} \alpha \to \gamma$

(d) $\vdash_{T_{P\vee}} \alpha \to \alpha$ ou equivalentemente $\vdash_{T_{P\vee}} \neg\alpha \vee \alpha$

(e) $\vdash_{T_{P\vee}} \alpha \vee \neg\alpha$

(f) $\vdash_{T_{P\vee}} \alpha \to \neg\neg\alpha$

(g) $\vdash_{T_{P\vee}} \neg\alpha \to (\alpha \to \beta)$

(h) $\vdash_{T_{P\vee}} \alpha \vee (\beta \vee \gamma) \to (\beta \vee (\alpha \vee \gamma)) \vee \alpha$

(i) $\vdash_{T_{P\vee}} (\beta \vee (\alpha \vee \gamma)) \vee \alpha \to \beta \vee (\alpha \vee \gamma)$

(j) $\vdash_{T_{P\vee}} \alpha \vee (\beta \vee \gamma) \to \beta \vee (\alpha \vee \gamma)$

5. Considere $\alpha \to \beta$ como uma abreviação da fórmula $\neg(\alpha \wedge \neg\beta)$ e prove os seguintes resultados na teoria formal $T_{P\wedge}$:

(a) $\alpha \to \beta, \beta \to \gamma \vdash_{T_{P\wedge}} \alpha \to \gamma$

(b) $\vdash_{T_{P\wedge}} \neg(\neg\alpha \wedge \alpha)$

(c) $\vdash_{T_{P\wedge}} \neg\neg\alpha \to \alpha$

(d) $\vdash_{T_{P\wedge}} \neg(\alpha \wedge \beta) \to (\beta \to \neg\alpha)$

(e) $\vdash_{T_{P\wedge}} \alpha \to \neg\neg\alpha$

(f) $\vdash_{T_{P\wedge}} \alpha \to \beta \to (\neg\beta \to \neg\alpha)$

(g) $\neg\alpha \to \neg\beta \vdash_{T_{P\wedge}} \beta \to \alpha$

(h) $\alpha \to \beta \vdash_{T_{P\wedge}} \gamma \wedge \alpha \to \beta \wedge \gamma$

(i) $\vdash_{T_{P\wedge}} \alpha \wedge \beta \to \beta \wedge \alpha$

6. Prove os seguintes resultados na teoria formal T_{P_T}:

(a) $\alpha, \alpha \to \beta \vdash_{T_{P_T}} \beta$

(b) $\vdash_{T_{P_T}} \alpha \to \alpha$

(c) $\alpha \to \beta, \beta \to \gamma \vdash_{T_{P_T}} \alpha \to \gamma$

(d) $\alpha \to (\beta \to \gamma), \beta \vdash_{T_{P_T}} \alpha \to \gamma$

(e) $\vdash_{T_{P_T}} \neg\neg\alpha \to \alpha$

(f) $\vdash_{T_{P_T}} \alpha \to \neg\neg\alpha$

(g) $\vdash_{T_{P_T}} \neg\alpha \to (\alpha \to \beta)$

(h) $\vdash_{T_{P_T}} (\neg\beta \to \neg\alpha) \to (\alpha \to \beta)$

(i) $\vdash_{T_{P_T}} (\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$

(j) $\vdash_{T_{P_T}} \alpha \to (\neg\beta \to \neg(\alpha \to \beta))$

(k) $\vdash_{T_{P_T}} (\alpha \to \beta) \to ((\neg\alpha \to \beta) \to \beta)$

7. Demonstre que o axioma A_3 da teoria formal T_P pode ser substituído pelo axioma:

$$A_3') \quad (\neg\alpha \to \neg\beta) \to (\beta \to \alpha)$$

sem alterar a classe de teoremas. Observe que o teorema da dedução nesta nova teoria formal, teria de ser provado para poder ser utilizado na demonstração.

8. Considere a teoria formal $T_{P_\to}$ alternativa para a lógica proposicional consistindo da linguagem $L_{P_\to}$, da regra de inferência modus ponens e dos seguintes axiomas:

(a) $(\neg\alpha \to \alpha) \to \alpha$

(b) $\alpha \to (\neg\alpha \to \beta)$

(c) $(\alpha \to \beta) \to ((\beta \to \gamma) \to (\alpha \to \gamma))$

Demonstre que T_P e $T_{P_\to}$ são teorias equivalentes, no sentido que descrevem o mesmo conjunto de teoremas. Observe que o teorema da dedução não está disponível em $T_{P_\to}$ até que seja demonstrado.

Capítulo 5

Resolução na Lógica Proposicional

Tendo visto os conceitos de verdade e dedutibilidade (provabilidade), precisamos saber como computar na lógica proposicional. É claro que as tabelas de verdade poderiam ser usadas como um algoritmo para provar se uma determinada sentença da lógica proposicional é verdadeira ou não. No entanto, tabelas de verdade são computacionalmente intratáveis, isto é, para sentenças compostas de n símbolos proposicionais, elas requerem 2^n verificações. Assim, o algoritmo tabelas de verdade tem uma complexidade computacional não polinomial, o qual é muito dispendioso para n relativamente grande. Além disso, tabelas de verdade é um método pouco geral em lógica, isto é, na maioria das lógicas computacionalmente interessantes tabelas de verdade não são aplicáveis.

Neste capítulo, desenvolveremos um método computacional, o método de **Eliminação Literal Complementar**, **ELC** para simplificar, também conhecido como **resolução básica**, o qual nos permitirá decidir se uma determinada sentença é ou não um teorema, de uma maneira mais eficiente que tabelas de verdade. Usaremos o conceito de computação, desenvolvidos aqui, para relacionar as noções de verdade e provabilidade.

Para desenvolver essas noções introduziremos dois novos símbolos, τ e $\Box$ (lido como "a cláusula vazia" ou "caixa"). A intenção ao usarmos esses símbolos é representar afirmações que são sempre verdadeiras (tautologias) ou falsas (contradições), respectivamente. Portanto, para relacionar nossos símbolos com verdade e dedutibilidade, como visto anteriormente, podemos considerar τ e $\Box$ como abreviação de $\alpha \vee \neg\alpha$ e $\alpha \wedge \neg\alpha$, respectivamente.

5.1 Forma Normal Conjuntiva

Definição 5.1.1 *Uma proposição $\alpha \in L_P$, está na* **forma normal conjuntiva**, *fnc, para simplificar, se as seguintes condições são satisfeitas:*

1. *α não possui qualquer ocorrência dos conectivos $\rightarrow$ e $\leftrightarrow$.*

2. *Nenhuma subfórmula de α cujo conectivo principal seja $\neg$ contém uma outra ocorrência de $\neg$, ou uma ocorrência de $\vee$, $\wedge$, τ ou $\Box$. Por exemplo, $\neg(\neg\beta)$, $\neg(\alpha \vee \beta)$, $\neg(\alpha \wedge \beta)$, $\neg\tau$ e $\neg\Box$ não estão na fnc.*

3. *Nenhuma subfórmula de α cujo principal conectivo seja $\vee$ contém $\wedge$, τ ou $\Box$. Ou seja, nem $\wedge$, τ ou $\Box$ ocorrem dentro do escopo de qualquer $\vee$.*

4. *Nem τ nem $\Box$ ocorre dentro do escopo de qualquer $\wedge$.*

A seguir daremos um exemplo de uma típica fórmula na fnc.

$$(p_1 \vee \neg p_2 \vee \neg p_1) \wedge (\neg p_3 \vee \neg p_2) \wedge (p_3 \vee \neg p_1 \vee p_3 \vee p_2)$$

Intuitivamente, podemos dizer que uma fórmula está na fnc se ela é uma conjunção de subfórmulas tais que cada uma das subfórmulas é uma disjunção de literais, onde um **literal** é uma proposição atômica ou a negação de uma proposição atômica. Para qualquer proposição atômica p, os literais p e $\neg p$ são chamados de **literais complementares**. Se $\mathtt{l}$ é um literal então denotaremos por $\overline{\mathtt{l}}$ seu literal complementar.

Definição 5.1.2 *Uma fórmula $\alpha \in L_P$ está na* **forma normal conjuntiva reduzida**, *fncr, para simplificar, se além de está na fnc, satisfaz, ainda, a condição abaixo*

5. *Nenhum literal aparece mais de uma vez no escopo de qualquer $\vee$.*

Exemplo 5.1.3 *Uma fncr equivalente à fnc anterior é $(\neg p_3 \vee \neg p_2) \wedge (p_3 \vee \neg p_1 \vee p_2)$.*

Proposição 5.1.4 *Qualquer proposição, $\alpha \in L_P$, pode ser reduzida a uma fórmula equivalente $\beta \in L_P$ que está na fncr.*

Demonstração: Usaremos a notação $\alpha \rightarrowtail\!\!\!\!\!\rightarrow \beta$, para dizer que α **se reduz a** β. De fato, estamos usando $\alpha \rightarrowtail\!\!\!\!\!\rightarrow \beta$ se, e somente se, $\alpha \equiv \beta$. A idéia que está por traz a noção de redução é transformar fórmulas em fórmulas equivalentes, mas que estejam mais "próximas" da sua fncr.

A seguir exibiremos um algoritmo para reduzir uma proposição α na sua fncr equivalente. Se α_1 é uma subfórmula de α e $\alpha_1 \rightarrowtail\!\!\!\!\!\rightarrow \alpha_2$, por alguma das seguintes reduções, então substitua em α as ocorrências de α_1 por α_2. O algoritmo parará quando não se puder aplicar mais nenhuma das reduções, descritas abaixo, a qualquer subfórmula de α.

1. Elimine na fórmula os conectivos $\rightarrow$ e $\leftrightarrow$.

   (a) $\alpha \rightarrow \beta \rightarrowtail\!\!\!\!\!\rightarrow \neg\alpha \vee \beta$

   (b) $\alpha \leftrightarrow \beta \rightarrowtail\!\!\!\!\!\rightarrow (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$

2. Reduções para obter uma fórmula satisfazendo o critério 1. da definição de fncr, isto é, negações tendo como escopo somente proposições atômicas.

   (a) $\neg\neg\alpha \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

   (b) $\neg(\alpha \vee \beta) \rightarrowtail\!\!\!\!\!\rightarrow \neg\alpha \wedge \neg\beta$

   (c) $\neg(\alpha \wedge \beta) \rightarrowtail\!\!\!\!\!\rightarrow \neg\alpha \vee \neg\beta$

   (d) $\neg\tau \rightarrowtail\!\!\!\!\!\rightarrow \square$

   (e) $\neg\square \rightarrowtail\!\!\!\!\!\rightarrow \tau$

3. As seguintes regras eliminam ocorrências de $\wedge$, τ ou $\square$ no escopo de qualquer $\vee$.

   (a) $\alpha \vee (\beta \wedge \gamma) \rightarrowtail\!\!\!\!\!\rightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

   (b) $(\beta \wedge \gamma) \vee \alpha \rightarrowtail\!\!\!\!\!\rightarrow (\beta \vee \alpha) \wedge (\gamma \vee \alpha)$

   (c) $\alpha \vee \tau \rightarrowtail\!\!\!\!\!\rightarrow \tau$

   (d) $\tau \vee \alpha \rightarrowtail\!\!\!\!\!\rightarrow \tau$

   (e) $\alpha \vee \square \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

   (f) $\square \vee \alpha \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

4. Garante nenhuma ocorrência de τ ou $\square$ no escopo de qualquer $\wedge$ por:

   (a) $\alpha \wedge \tau \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

   (b) $\tau \wedge \alpha \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

   (c) $\alpha \wedge \square \rightarrowtail\!\!\!\!\!\rightarrow \square$

   (d) $\square \wedge \alpha \rightarrowtail\!\!\!\!\!\rightarrow \square$

   Como cada redução acima transforma uma fórmula numa outra equivalente, então usando-as de modo exaustivo (isto é, até não se puder aplicar mais nenhuma) e independente da ordem, necessariamente devemos obter uma fórmula na forma normal conjuntiva equivalente à original,

5. Para reduzir uma fórmula na fnc a uma equivalente na fncr devemos eliminar repetições dos mesmos símbolos proposicionais no escopo de qualquer $\vee$ como segue:

   Suponha que α está na fnc. Seja $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$, onde cada α_i é um literal, uma subfórmula de α.

   (a) Se $\alpha_1 = \alpha_n$ então $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n \rightarrowtail\!\!\!\!\!\rightarrow \alpha_2 \vee \ldots \vee \alpha_n$

   (b) Se $\alpha_1 = \neg\alpha_n$ ou $\alpha_n = \neg\alpha_1$ então $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n \rightarrowtail\!\!\!\!\!\rightarrow \tau$

   Após isto devemos observar se não foram geradas subfórmulas do tipo $\alpha \vee \tau$, $\tau \vee \alpha$, $\alpha \wedge \tau$ ou $\tau \wedge \alpha$. Se isso acontecer, então devemos novamente aplicar as reduções do item 3 e/ou 4, dependendo do caso. ■

Exemplo 5.1.5 *Reduzir* $\alpha \equiv \neg(p_1 \to (p_2 \to p_3)) \to ((p_1 \to p_2) \to (p_1 \to p_3))$ *na sua fncr.*

Primeiro eliminamos as implicações, através da redução 1.a.

$$\begin{aligned}\alpha &\twoheadrightarrow \neg(\neg(p_1 \to (p_2 \to p_3))) \vee ((p_1 \to p_2) \to (p_1 \to p_3))\\ &\twoheadrightarrow \neg(\neg(\neg p_1 \vee (p_2 \to p_3))) \vee (\neg(p_1 \to p_2) \vee (p_1 \to p_3))\\ &\twoheadrightarrow \neg(\neg(\neg p_1 \vee (\neg p_2 \vee p_3))) \vee (\neg(\neg p_1 \vee p_2) \vee (\neg p_1 \vee p_3))\end{aligned}$$

Agora colocamos a negação dentro do parênteses através da redução 2.b.

$$\begin{aligned}&\twoheadrightarrow \neg(\neg\neg p_1 \wedge \neg(\neg p_2 \vee p_3)) \vee ((\neg\neg p_1 \wedge \neg p_2) \vee (\neg p_1 \vee p_3))\\ &\twoheadrightarrow \neg(\neg\neg p_1 \wedge (\neg\neg p_2 \wedge \neg p_3)) \vee ((\neg\neg p_1 \wedge \neg p_2) \vee (\neg p_1 \vee p_3))\\ &\twoheadrightarrow (\neg\neg\neg p_1 \vee \neg(\neg\neg p_2 \wedge \neg p_3)) \vee ((\neg\neg p_1 \wedge \neg p_2) \vee (\neg p_1 \vee p_3))\\ &\twoheadrightarrow (\neg\neg\neg p_1 \vee (\neg\neg\neg p_2 \vee \neg\neg p_3)) \vee ((\neg\neg p_1 \wedge \neg p_2) \vee (\neg p_1 \vee p_3))\end{aligned}$$

Eliminamos a dupla negação via a redução 2.a.

$$\twoheadrightarrow (\neg p_1 \vee (\neg p_2 \vee p_3)) \vee ((p_1 \wedge \neg p_2) \vee (\neg p_1 \vee p_3))$$

Realizamos distributividade do $\vee$ *sobre o* $\wedge$ *(redução 3.a com comutatividade implícita).*

$$\begin{aligned}&\twoheadrightarrow (\neg p_1 \vee (\neg p_2 \vee p_3)) \vee ((p_1 \vee (\neg p_1 \vee p_3)) \wedge (\neg p_2 \vee (\neg p_1 \vee p_3)))\\ &\twoheadrightarrow ((\neg p_1 \vee (\neg p_2 \vee p_3)) \vee (p_1 \vee (\neg p_1 \vee p_3))) \wedge ((\neg p_1 \vee (\neg p_2 \vee p_3)) \vee (\neg p_2 \vee (\neg p_1 \vee p_3)))\end{aligned}$$

Assim, obtivemos uma fnc equivalente à fórmula α*. Para reduzir esta nova fórmula à uma na fncr devemos usar associatividade e comutatividade de modo a rearranjar a fórmula e deixar as ocorrências dos mesmos* p_i*'s juntas. Desse modo, reduzimos a fórmula anterior à seguinte fórmula:*

$$\twoheadrightarrow ((\neg p_1 \vee (p_1 \vee \neg p_1)) \vee \neg p_2 \vee (p_3 \vee p_3)) \wedge ((\neg p_1 \vee \neg p_1) \vee (\neg p_2 \vee \neg p_2) \vee (p_3 \vee p_3))$$

Agora, eliminamos as ocorrências dos mesmos p_i*'s aplicando as reduções em 5, acima, obtemos*

$$\twoheadrightarrow (((\neg p_1 \vee \tau) \vee \neg p_2) \vee p_3) \wedge ((\neg p_1 \vee \neg p_2) \vee p_3)$$

Agora eliminamos τ*, aplicando as reduções em 3 e 4.*

$$\begin{aligned}&\twoheadrightarrow ((\tau \vee \neg p_2) \vee p_3) \wedge ((\neg p_1 \vee \neg p_2) \vee p_3)\\ &\twoheadrightarrow (\tau \vee p_3) \wedge ((\neg p_1 \vee \neg p_2) \vee p_3)\\ &\twoheadrightarrow (\tau) \wedge (\neg p_1 \vee \neg p_2 \vee p_3)\\ &\twoheadrightarrow ((\neg p_1 \vee \neg p_2) \vee p_3)\end{aligned}$$

As reduções apresentadas na demonstração da proposição 5.1.4 são suficientes para transformar qualquer fórmula de L_P numa fncr equivalente. No entanto, as vezes, a

fórmula resultante deste processo é extensa, mas poderia ser simplificada usando outras reduções auxiliares. Por exemplo, vejamos o caso simples da fórmula $(p_1 \wedge p_2) \vee (p_1 \wedge p_3)$. Usando somente as reduções da proposição 5.1.4, teríamos que

$$\begin{array}{ll} (p_1 \wedge p_2) \vee (p_1 \wedge p_3) & \rightarrowtail\!\!\!\!\!\rightarrow ((p_1 \wedge p_2) \vee p_1) \wedge ((p_1 \wedge p_2) \vee p_3) \\ & \rightarrowtail\!\!\!\!\!\rightarrow (p_1 \vee p_1) \wedge (p_2 \vee p_1) \wedge (p_1 \vee p_3) \wedge (p_2 \vee p_3) \\ & \rightarrowtail\!\!\!\!\!\rightarrow p_1 \wedge (p_2 \vee p_1) \wedge (p_1 \vee p_3) \wedge (p_2 \vee p_3) \end{array}$$

Mas, observe que $(p_1 \wedge p_2) \vee (p_1 \wedge p_3) \equiv p_1 \wedge (p_2 \vee p_3)$ e $p_1 \wedge (p_2 \vee p_3)$ está na fncr. Logo, poderíamos usar esta fórmula como redução.

Já na fórmula $(p_1 \wedge p_2) \vee (p_1 \wedge p_2)$. Usando somente as reduções da proposição 5.1.4, teríamos que

$$\begin{array}{ll} (p_1 \wedge p_2) \vee (p_1 \wedge P_2) & \rightarrowtail\!\!\!\!\!\rightarrow ((p_1 \wedge p_2) \vee p_1) \wedge ((p_1 \wedge p_2) \vee p_2) \\ & \rightarrowtail\!\!\!\!\!\rightarrow (p_1 \vee p_1) \wedge (p_2 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_2) \\ & \rightarrowtail\!\!\!\!\!\rightarrow p_1 \wedge (p_2 \vee p_1) \wedge (p_1 \vee p_2) \wedge p_2 \end{array}$$

Mas, trivialmente, $(p_1 \wedge p_2) \vee (p_1 \wedge P_2) \equiv p_1 \wedge p_2$. Assim poderíamos definir uma redução que nos permita atingir esta fórmula diretamente.

Reduções Auxiliares:

1. $\alpha \vee \beta \rightarrowtail\!\!\!\!\!\rightarrow \beta \vee \alpha$
2. $\alpha \wedge \beta \rightarrowtail\!\!\!\!\!\rightarrow \beta \wedge \alpha$
3. $(\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \rightarrowtail\!\!\!\!\!\rightarrow \alpha \wedge (\beta \vee \gamma)$
4. $\alpha \vee \alpha \rightarrowtail\!\!\!\!\!\rightarrow \alpha$
5. $\alpha \wedge \alpha \rightarrowtail\!\!\!\!\!\rightarrow \alpha$
6. $\alpha \wedge (\alpha \vee \beta) \rightarrowtail\!\!\!\!\!\rightarrow \alpha$

5.2 Notação Clausal: sintaxe

Para descrever o processo de eliminação de literais complementares **ELC** sobre fórmulas na forma normal conjuntiva reduzida (fncr) definiremos mais alguns termos. O nosso estilo de especificação da sintaxe é chamado **gramática livre de contexto**. Cada linha representa uma regra para transformar um símbolo não terminal (símbolos auxiliares que não figurarão na cadeia resultado) numa cadeia de símbolos. Para isso usaremos as seguintes convenções:

- Um símbolo terminal será colocado entre aspas duplas.
- O símbolo "::=" será lido como "é um".
- O símbolo "|" será lido como "ou" (representando uma alternativa).
- Justaposição de x com y será denotado por xy.

Exemplo 5.2.1

1. *proposição-atômica* ::= "p_1'' | ... | "p_n'' | ...
2. *literal* ::=*+literal* | *-literal*

   será lido como "um literal é um literal positivo ou um literal negativo".
3. *+literal* ::=*proposição-atômica.*
4. *-literal* ::= " $\neg$ " *proposição-atômica.*
5. *cláusula-não-vazia* ::=*literal* | *literal* " $\vee$ ' *cláusula-não-vazia*
6. *cláusula-básica*::= "$\square$ " | *cláusula-não-vazia*
7. *proposição-básica* ::=*cláusula-básica* | *cláusula-básica* " $\wedge$ " *proposição-básica*

Uma fórmula na fncr é uma proposição-básica. O adjetivo "básico" se refere ao fato de que a cláusula ou proposição que ela procede não contém variáveis. Como não existem variáveis na linguagem proposicional, todas as cláusulas e proposições são básicas. Essa distinção somente será apreciada quando estudarmos lógica de predicados.

Às vezes é conveniente pensar uma proposição básica como um conjunto de cláusulas básicas, e uma cláusula básica como um conjunto de literais. Isto não causará confusão uma vez que $\alpha \vee \alpha \equiv \alpha \wedge \alpha \equiv \alpha$ e sabemos que uma proposição básica é sempre uma conjunção ($\wedge$) de seus elementos, uma cláusula básica é sempre uma disjunção ($\vee$) de seus elementos (literais). Assim, podemos dar as seguintes definições na forma de gramática livre de contexto.

cláusula-não-vazia::=literal | literal $\cup$ cláusula-não-vazia.

proposição-básica ::=cláusula-básica | cláusula-básica $\cup$ proposição-básica.

O símbolo "$\cup$" indica união.

5.3 Eliminação de Literais Complementares

Após essas convenções podemos definir a **eliminação de literais complementares** (**ELC**) de uma fórmula na fncr.

Definição 5.3.1 *Seja* $\mathtt{l}$ *um literal. Se* α *e* β *são cláusulas-básicas, tais que* $l \in \alpha$ *e* $\overline{\mathtt{l}} \in \beta$*, então o* **resolvente básico** *de* α *e* β *com respeito a* $\mathtt{l}$*, é* $(\alpha - \{l\}) \cup (\beta - \{\overline{\mathtt{l}}\})$[1]*. O qual é ilustrado pela figura 5.1.*

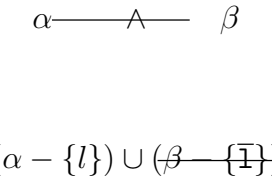


Figura 5.1: Resolvente básico de duas cláusulas

Dessa definição podemos dizer que o resolvente básico é a união das cláusulas originais com exceção dos literais complementares, que foram eliminados. Na convenção de que $\square$ é a cláusula vazia, o resolvente básico de $\mathtt{l}$ e $\overline{\mathtt{l}}$ é $\square$.

Observação: A resolução básica ou **ELC**, pode ser considerada como uma variante do modus ponens (MP), que estabelece que de α e $\alpha \rightarrow \beta$, podemos deduzir β. Isto poderia ser escrito por "de α e $\neg\alpha \vee \beta$ podemos deduzir β". Assim, quando α é um literal e β uma cláusula, podemos deduzir β eliminando α e $\overline{\alpha}$, como mostrado na figura 5.2.

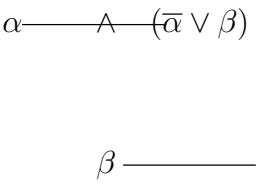


Figura 5.2: ELC como uma variante do modus ponem

Exemplo 5.3.2 *A seguir veremos algumas deduções usando ELC.*

1. *O resolvente básico de* $p_1 \vee p_2$ *e* $p_3 \vee \neg p_1$ *é* $p_2 \vee p_3$ *como ilustrado na figura 5.3.*

$$(p_1 \vee \cancel{p_2}) \cancel{\wedge (p_3 \vee \neg p_1)}$$

$$(p_2 \vee \cancel{p_3})$$

Figura 5.3: Resolvente básico de $p_1 \vee p_2$ e $p_3 \vee \neg p_1$

$$(p_1 \vee \cancel{p_2}) \cancel{\wedge (\neg p_1 \vee \neg p_2)}$$

$$(\neg p_1 \vee \cancel{p_1})$$

Figura 5.4: Um resolvente básico de $p_1 \vee p_2$ e $\neg p_1 \vee \neg p_2$

2. *Uma resolvente básico de $p_1 \vee p_2$ e $\neg p_1 \vee \neg p_2$ é $\neg p_1 \vee p_1$, como ilustrado na figura 5.4.*

3. *O resolvente básico de $p_1 \vee p_2$ e $\neg p_1 \vee \neg p_2$ é a fórmula $\neg p_2 \vee p_2$, como ilustrado na figura 5.11.*

$$(p_1 \vee \cancel{p_2}) \cancel{\wedge (\neg p_1 \vee \neg p_2)}$$

$$(\neg p_2 \vee \cancel{p_2})$$

Figura 5.5: Outro resolvente básico de $p_1 \vee p_2$ e $\neg p_1 \vee \neg p_2$

Observação: *Os ítens 2. e 3. do exemplo 5.3.2 são ambos legítimos. No entanto não seria possível eliminar ambos os pares de literais complementares de uma vez.*

Uma dedução de $\neg p_1$ a partir de $\neg p_1 \vee p_2 \vee \neg p_3$, p_3 e $\neg p_2$ usando o método de ELC, pode ser vista na figura 5.6.

Observação: *É claro que poderíamos ter optado por eliminar primeiro p_2 e $\neg p_2$. O importante é que o resultado será sempre o mesmo, independente da ordem de eliminação de literais.*

[1] $\alpha - \beta$ indica o conjunto diferença de α e β, isto é, $\alpha - \beta = \{l \ / \ l \in \alpha \text{ e } l \notin \beta\}$.

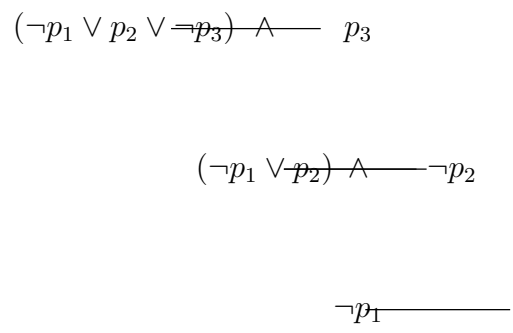


Figura 5.6: Dedução de $\neg p_1$ a partir de $\neg p_1 \vee p_2 \vee \neg p_3, p_3$ e $\neg p_2$ usando ELC

Neste último exemplo realizamos duas resoluções básicas consecutivas para deduzir $\neg p_1$. Esta idéia pode ser generalizada e formalizada.

Definição 5.3.3 *Uma* **dedução por resolução** *de uma cláusula α a partir de um conjunto de cláusulas Γ, denotado por $\Gamma \xrightarrow{*} \alpha$, é uma seqüência finita de cláusulas $\beta_1, \ldots, \beta_n$ tal que*

1. *β_n é α*
2. *para cada i, $1 \leq i \leq n$, ou β_i*
   (a) *é um elemento de Γ, ou*
   (b) *$p, \neg p \in \beta_i$ para algum símbolo proposicional p, ou*
   (c) *é um resolvente básico de β_j e β_k, onde $j, k < i$.*

Note que a seqüência finita de cláusulas $\beta_1, \ldots, \beta_n$ de uma dedução por resolução, pode ser disposta na forma de uma árvore invertida, como na figura 5.6 que ilustra a seqüência:

$$\neg p_1 \vee p_2 \vee \neg p_3, p_3, \neg p_1 \vee p_2, \neg p_2, \neg p_1.$$

Exemplo 5.3.4 *Do exemplo 5.3.2, podemos concluir que:*

1. $p_1 \vee p_2, p_3 \vee \neg p_1 \xrightarrow{*} p_2 \vee p_3$,
2. $p_1 \vee p_2, \neg p_1 \vee \neg p_2 \xrightarrow{*} \neg p_1 \vee p_1$,
3. $p_1 \vee p_2, \neg p_1 \vee \neg p_2 \xrightarrow{*} \neg p_2 \vee p_2$,
4. $\neg p_1 \vee p_2 \vee \neg p_3, p_3, \neg p_2 \xrightarrow{*} \neg p_1$,

Numa dedução por resolução $\beta_1, \ldots, \beta_n$ dizemos que β_i **depende** de β_j, com $j < i$, se β_i é o resolvente básico de β_k e β_p, e uma delas é β_j ou depende de β_i. Denotaremos por $Dep(\alpha)$ o conjunto de todas as cláusulas que dependem de α numa dedução por resolução. Por exemplo, na dedução por resolução descrito na árvore 5.6, $Dedp_3 = \{\neg p_1 \vee p_2, \neg p_1\}$. Assim, na árvore que descreve uma dedução por resolução, os dependentes de uma cláusula são os descendentes dela.

Analogamente como em provas em teorias formais, numa dedução por resolução $\Gamma \xrightarrow{*} \alpha$, uma cláusula de Γ pode ser usada mais de uma vez ou inclusive nenhuma. A partir de agora omitiremos o símbolo $\wedge$ das árvores.

Exemplo 5.3.5 *Seja Γ o seguinte conjunto de cláusulas: $\{\neg p_3, p_1 \vee \neg p_2, p_1 \vee p_3, \neg p_1 \vee p_2 \vee p_3\}$. Uma dedução por resolução de $\Gamma \xrightarrow{*} p_2 \vee p_4$ é ilustrado na figura 5.7.*

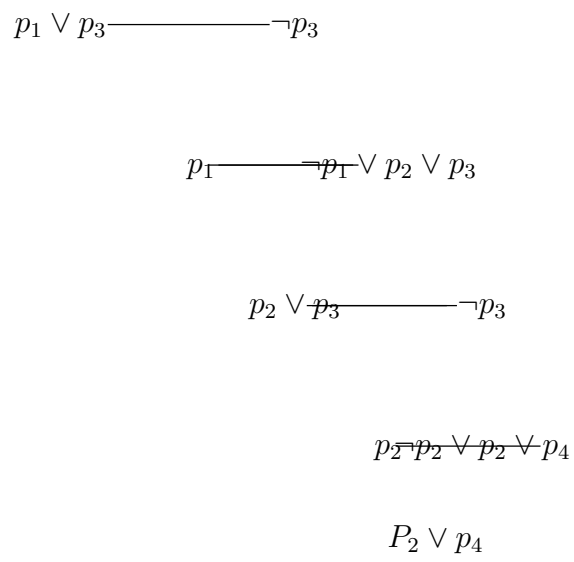


Figura 5.7: Dedução por resolução de $\{\neg p_3, p_1 \vee \neg p_2, p_1 \vee p_3, \neg p_1 \vee p_2 \vee p_3\} \xrightarrow{*} p_2 \vee p_4$

Nesta dedução a cláusula $\neg p_3$ é usada duas vezes, enquanto a cláusula $p_1 \vee \neg p_2$ nenhuma. Note também que a cláusula $\neg p_2 \vee p_2 \vee p_4 \notin \Gamma$. Isto é possível porque esta cláusula contem dois literais complementares e portanto satisfaz o item 2.(b) da definição 5.3.3.

Exemplo 5.3.6 *Seja $\Gamma = \{\neg p_1, p_1 \vee P_3, p_2 \vee \neg p_3 \vee p_4, \neg p_1 \vee p_3 \vee \neg p_4, \neg p_1 \vee \neg p_2\}$ e $\alpha = p_2 \vee p_3$ então, como ilustrado na figura 5.8, $\Gamma \xrightarrow{*} \alpha$. Observe que nesta dedução a cláusula $p_1 \vee p_3$ foi usada duas vezes, já a cláusula $\neg p_1 \vee \neg p_2$ nenhuma.*

Proposição 5.3.7 *Seja Γ um conjunto de cláusulas básicas e α e β cláusulas básicas, e* `l` *um literal.*

1. *Monotonicidade à esquerda: Se $\Gamma \xrightarrow{*} \alpha$ e $\Gamma \subseteq \Theta$ então $\Theta \xrightarrow{*} \alpha$*

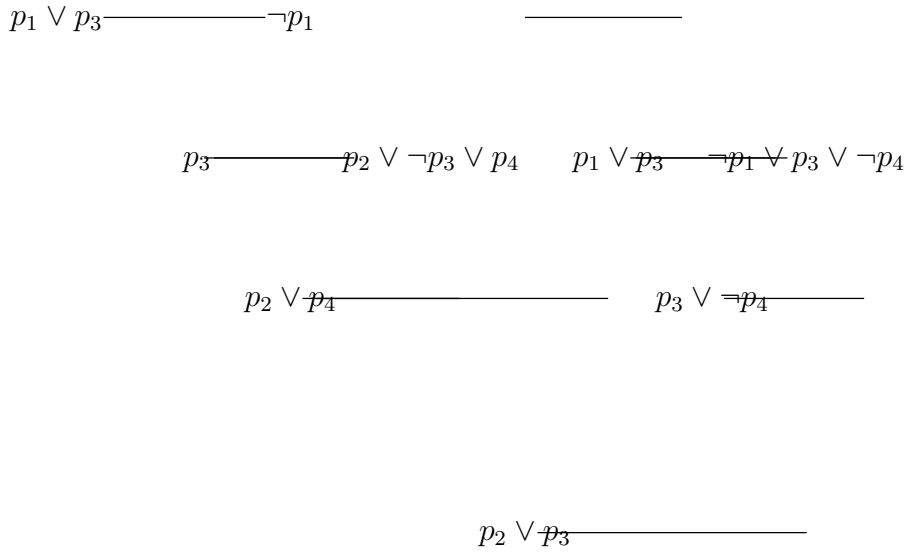


Figura 5.8: Dedução por resolução de $\neg p_1, p_1 \vee P_3, p_2 \vee \neg p_3 \vee p_4, \neg p_1 \vee p_3 \vee \neg p_4, \neg p_1 \vee \neg p_2 \xrightarrow{*} p_2 \vee p_3$.

2. *Monotonicidade à direita: Se* $\alpha \subseteq \beta$ *e* $\Gamma \xrightarrow{*} \alpha$ *então* $\Gamma \xrightarrow{*} \beta$

3. *Compacidade: Se* $\Gamma \xrightarrow{*} \alpha$ *então existe um conjunto finito* $\Gamma_0 \subseteq \Gamma$ *tal que* $\Gamma_0 \xrightarrow{*} \alpha$

4. *Teorema da dedução simples:* $\Gamma, \mathtt{l} \xrightarrow{*} \beta$ *se, e somente se,* $\Gamma \xrightarrow{*} \overline{\mathtt{l}} \vee \beta$

Demonstração: Monotonicidade à esquerda e compacidade são resultados diretos da definição de dedução por resolução.

Monotonicidade à direita: Se $\Gamma \xrightarrow{*} \alpha$, então existe uma dedução por resolução $\alpha_1, \ldots, \alpha_n$ de α a partir de Γ. Assim, $\alpha_n = \alpha$. Se $\alpha \neq \square$ então existe um literal $\gamma \in \alpha$. Então, neste caso, $\alpha_1, \ldots, \alpha_n, \gamma \vee \neg\gamma \vee (\beta - \alpha), \beta$ é uma dedução por resolução de β a partir de Γ. Se $\alpha = \square$ então α_{n-2} e α_{n-1} são literais complementares. Logo, a seqüência $\alpha_1, \ldots, \alpha_{n-2}, \alpha_{n-1} \vee \alpha_{n-2} \vee \beta, \alpha_{n-2} \vee \beta, \alpha_{n-1}, \beta$ é uma dedução por resolução de β a partir de Γ

Teorema da dedução simples: Se $\Gamma, \mathtt{l} \xrightarrow{*} \beta$ então existe uma dedução por resolução $\beta_1, \ldots, \beta_n$ de β a partir de $\Gamma \cup \{\mathtt{l}\}$. Se $\mathtt{l} \notin \{\beta_1, \ldots, \beta_n\}$ então $\beta_1, \ldots, \beta_n$ é uma dedução por resolução de β a partir de Γ, e portanto $\Gamma \xrightarrow{*} \beta$. Logo, por monotonicidade à direita $\Gamma \xrightarrow{*} \overline{\mathtt{l}} \vee \beta$.

Se $\mathtt{l} =\in \{\beta_1, \ldots, \beta_n\}$ então $\mathtt{l} = \beta_i$ para algum $i \in \{1, \ldots, n\}$. Sem perda de generalidade podemos considerar que a seqüência satisfaz $Dep(\mathtt{l}) = \{\beta_{i+1}, \ldots, \beta_n\}$. Logo, a

seqüência $\beta_1, \ldots, \beta_{i-1}, \beta_{i+1} \vee \overline{\mathtt{l}}, \ldots, \beta_n \vee \overline{\mathtt{l}}$ é uma dedução por resolução de $\beta \vee \overline{\mathtt{l}}$ a partir de Γ e portanto $\Gamma \xrightarrow{*} \overline{\mathtt{l}} \vee \beta$.

Por outro lado, se $\Gamma \xrightarrow{*} \overline{\mathtt{l}} \vee \beta$, então existe uma dedução por resolução $\beta_1, \ldots, \beta_n$ de $\overline{\mathtt{l}} \vee \beta$ a partir de Γ. Como $\beta_n = \overline{\mathtt{l}} \vee \beta$, a seqüência $\beta_1, \ldots, \beta_n, \mathtt{l}, \beta$ é uma dedução por resolução de β a partir de $\Gamma \cup \{\mathtt{l}\}$, e portanto $\Gamma, \mathtt{l} \xrightarrow{*} \beta$. ■

O teorema da dedução simples pode ser generalizado para cláusulas em vez de literais.

Teorema 5.3.8 (Teorema da Dedução) *Seja* $\alpha = \mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k$ *e* $\overline{\alpha} = \overline{\mathtt{l}_1} \vee \ldots \vee \overline{\mathtt{l}_k}$.

$$\Gamma, \alpha \xrightarrow{*} \beta \text{ se, somente se, } \Gamma \xrightarrow{*} \overline{\alpha} \vee \beta. \tag{5.1}$$

Demonstração: Demonstraremos por indução no tamanho de $\mid \alpha \mid$, isto é k, que a equação 5.1 é válida.

Caso base: Se $k = 1$ então a equação 5.1 sai direto do teorema da dedução simples.

Hipóteses indutiva: Suponha que a equação 5.1 é válida para todo conjunto de cláusulas Γ e cláusula α de tamanho menor ou igual a k. Seja $\alpha = \mathtt{l}_1 \vee \ldots \vee \mathtt{l}_{k+1}$. Se

Etapa indutiva: $\Gamma, \alpha \xrightarrow{*} \beta$ então existe uma dedução por resolução $\beta_1, \ldots, \beta_n$ de β a partir de $\Gamma \cup \alpha$. Se $\alpha \notin \{\beta_1, \ldots, \beta_{n-1}\}$ então $\beta_1, \ldots, \beta_n$ é uma dedução por resolução de β a partir de Γ e portanto $\Gamma \xrightarrow{*} \beta$. Logo por monotonicidade à direita $\Gamma \xrightarrow{*} \overline{\alpha} \vee \beta$. Se $\alpha = \beta_i$, para algum $i \in \{1, \ldots, n-1\}$ e $\alpha \in Dep(\beta_j)$ para algum $j \in \{1, \ldots, n-1\}$, então $eta_1, \ldots, \beta_n$ é uma dedução por resolução de β a partir de Γ e portanto $\Gamma \xrightarrow{*} \beta$. Logo por monotonicidade à direita $\Gamma \xrightarrow{*} \overline{\alpha} \vee \beta$. Se $\alpha = \beta_i$, para algum $i \in \{1, \ldots, n-1\}$ e $\alpha \notin Dep(\beta_j)$ para todo $j \in \{1, \ldots, n-1\}$, então existe β_j tal que β_j é o resolvente básico de α e outra cláusula. Seja $\mathtt{l}$ o literal em α que foi eliminado nessa resolução básica. Por simplicidade, considera que $j = i + 1$ e $Dep(\beta_j) = \{\beta_{j+1}, \ldots, \beta_n\}$. Observe que sempre é possível reorganizar a seqüência $\beta_1, \ldots, \beta_n$ de tal modo que isto ocorra. Assim, a seqüência $\beta_1, \ldots, \beta_{i-1}, \beta_{i+2} \vee \overline{\mathtt{l}}, \ldots, \beta_n \vee \overline{\mathtt{l}}$ é uma dedução por resolução de $\overline{\mathtt{l}} \vee \beta$ a partir de $\Gamma \cup (\alpha - \{\mathtt{l}\})$. Assim, $\Gamma, \alpha - \{\mathtt{l}\} \xrightarrow{*} \overline{\mathtt{l}} \vee \beta$. Como $\mid \alpha - \mathtt{l} = k$ então pela hipóteses indutiva, $\Gamma \xrightarrow{*} \overline{\alpha - \mathtt{l}} \vee \overline{\mathtt{l}} \vee \beta$ e portanto $\Gamma \xrightarrow{*} \overline{\alpha} \vee \beta$.

Inversamente, se $\Gamma \xrightarrow{*} \overline{\alpha} \vee \beta$ então existe uma dedução por resolução $\beta_1, \ldots, \beta_n$ de $\overline{\alpha} \vee \beta$ a partir de Γ. Então a seqüência $\beta_1, \ldots, \beta_n, \alpha, \overline{\alpha - \{\mathtt{l}_1\}} \vee \beta, \overline{(\alpha - \{\mathtt{l}_1), \mathtt{l}_2\}} \vee \beta, \ldots, \overline{\alpha - \{\mathtt{l}_1, \ldots, \mathtt{l}_{k+1}\}} \vee \beta$ é uma dedução por resolução de β a partir de $\Gamma \cup \{\alpha\}$, ou seja $\Gamma, \alpha \xrightarrow{*} \beta$. ■

Definição 5.3.9 *Uma dedução por resolução de* $\square$ *a partir de um conjunto de cláusulas básicas* Γ*, denotado por* $\Gamma \xrightarrow{*} \square$*, é chamada uma* **refutação de resolução** *ou simplesmente uma* **refutação**, *de* Γ.

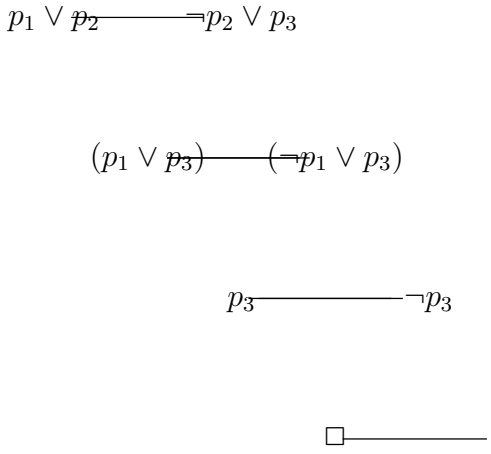


Figura 5.9: $p_1 \vee p_2, \neg p_2 \vee p_3, \neg p_1 \vee p_3, \neg p_3 \xrightarrow{*} \square$

Exemplo 5.3.10 *Uma refutação de* $\Gamma = \{p_1 \vee p_2, \neg p_2 \vee p_3, \neg p_1 \vee p_3, \neg p_3\}$ *é ilustrada na figura 5.9*

Corolário 5.3.11 *Se* $\Gamma, \neg\alpha \xrightarrow{*} \square$ *então* $\Gamma \xrightarrow{*} \alpha$

DEMONSTRAÇÃO: Se $\alpha = \mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k$ então

$$\begin{aligned} \neg\alpha &= \neg(\mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k) \\ &= \neg\mathtt{l}_1 \wedge \ldots \wedge \neg\mathtt{l}_k \\ &= \overline{\mathtt{l}_1} \wedge \ldots \wedge \overline{\mathtt{l}_k} \end{aligned}$$

Assim, se $\Gamma, \neg\alpha \xrightarrow{*} \square$ então $\Gamma, \overline{\mathtt{l}_1}, \ldots, \overline{\mathtt{l}_k} \xrightarrow{*} \square$. Logo, pelo teorema da dedução 5.3.8, $\Gamma, \overline{\mathtt{l}_1}, \ldots, \overline{\mathtt{l}_{k-1}} \xrightarrow{*} \overline{\overline{\mathtt{l}_k}} \vee \square$. Isto é, $\Gamma, \mathtt{l}_1^{op}, \ldots, \mathtt{l}_{k-1}^{op} \xrightarrow{*} \mathtt{l}_k$. Assim, aplicando $k-1$ vezes o teorema da dedução teremos que

$$\Gamma \xrightarrow{*} \mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k$$

Ou seja, $\Gamma \xrightarrow{*} \alpha$. ■

Observe que este corolário pode simplificar bastante uma prova, pois a negação de uma cláusula é uma conjunção de literais.

Por exemplo se considerarmos o conjunto de cláusulas Γ e a cláusula α do exemplo 5.3.6, podemos provar que $\Gamma, \neg\alpha \xrightarrow{*} \square$ de forma bem mais simples, como mostra a figura 5.10.

Nosso método de computação, o qual estamos chamando de **ELC** ou **resolução básica**, foi descrita como um meio de produzir **refutações** de conjuntos de cláusulas. O objetivo, aqui, em usando essa terminologia, é, ao refutar um conjunto de cláusulas,

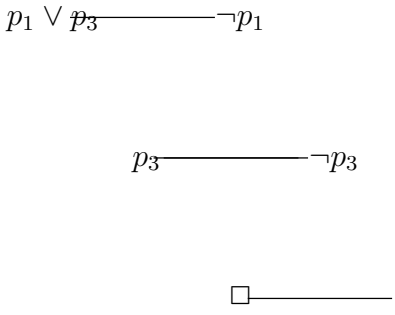


Figura 5.10: $\{negp_1, p_1 \vee P_3, p_2 \vee \neg p_3 \vee p_4, \neg p_1 \vee p_3 \vee \neg p_4, \neg p_1 \vee \neg p_2, \neg p_2, \neg p_3\} \xrightarrow{*} \square$

mostrar que a conjunção dessas cláusulas é contraditória ou impossível de satisfazer. Assim, quando relacionamos esse método de computação com nosso entendimento de verdade e prova, estamos tentando refutar a **negação** do que sabemos ser verdadeiro e/ou provável. Se uma fórmula é sempre verdadeira, então sua negação é sempre falsa e portanto esperamos exibir sua refutação. Igualmente, se podemos refutar a negação de uma fórmula α, então esperamos poder provar α em nossa teoria formal.

Exemplo 5.3.12 *Para mostrar que a fórmula* $\alpha \equiv (p_1 \wedge (p_1 \rightarrow \neg(p_1 \vee p_2))) \rightarrow (\neg p_1 \wedge \neg p_2)$ *é uma tautologia usando o método ELC, primeiro negamos a fórmula, depois reduzimos a fórmula na sua fncr e, finalmente, mostramos uma refutação da fórmula na fncr. Isto é, realizamos os seguintes passos.*

Passo 1: Negação da fórmula α. $\neg\alpha \equiv \neg((p_1 \wedge (p_1 \rightarrow \neg(p_1 \vee p_2))) \rightarrow (\neg p_1 \wedge \neg p_2))$

Passo 2: Transformação à fncr de $\neg\alpha$.

$$\begin{aligned}
\neg\alpha \quad &\rightarrowtail\!\!\!\!\!\rightarrow \neg((p_1 \wedge (\neg p_1 \vee \neg(p_1 \vee p_2))) \rightarrow (\neg p_1 \wedge \neg p_2)) \\
&\rightarrowtail\!\!\!\!\!\rightarrow \neg(\neg(p_1 \wedge (\neg p_1 \vee \neg(p_1 \vee p_2))) \vee (\neg p_1 \wedge \neg p_2)) \\
&\rightarrowtail\!\!\!\!\!\rightarrow \neg\neg(p_1 \wedge (\neg p_1 \vee (\neg p_1 \wedge \neg p_2))) \wedge \neg(\neg p_1 \wedge \neg p_2) \\
&\rightarrowtail\!\!\!\!\!\rightarrow (p_1 \wedge (\neg p_1 \vee (\neg p_1 \wedge \neg p_2))) \wedge (\neg\neg p_1 \vee \neg\neg p_2) \\
&\rightarrowtail\!\!\!\!\!\rightarrow (p_1 \wedge (\neg p_1 \vee (\neg p_1 \wedge \neg p_2))) \wedge (p_1 \vee p_2) \\
&\rightarrowtail\!\!\!\!\!\rightarrow (p_1 \wedge ((\neg p_1 \vee \neg p_1) \wedge (\neg p_1 \vee \neg p_2))) \wedge (p_1 \vee p_2) \\
&\rightarrowtail\!\!\!\!\!\rightarrow p_1 \wedge \neg p_1 \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_2)
\end{aligned}$$

Passo 3: Aplicar método ELC às cláusulas p_1, $\neg p_1$, $\neg p_1 \vee \neg p_2$ *e* $p_1 \vee p_2$. *O qual se simplifica ao resolvente básico das cláusulas* p_1 *e* $\neg p_1$, *como ilustrado na figura 5.11. Assim,* $p_1, \neg p_1, \neg p_1 \vee \neg p_2, p_1 \vee p_2 \xrightarrow{*} \square$

É claro que o método de resolução, a cada passo, pode ter muitos caminhos possíveis a serem seguidos. Uma maneira de automatizar o processo é considerar todos os caminhos possíveis. Ou seja, em cada etapa gerar todos os possíveis resolventes do conjunto de cláusulas correntes.

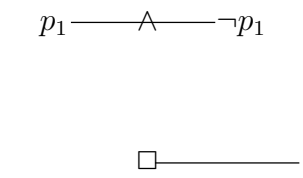


Figura 5.11: Resolvente básico de p_1 e $\neg p_1$

Definição 5.3.13 *Dado um conjunto de cláusulas* Γ*, definimos* $\mathcal{R}(\Gamma)$ *como sendo a união de* Γ *com o conjunto de todos os possíveis resolventes básicos de cláusulas em* Γ*. Definimos* $\mathcal{R}^n$ *como segue*

$\mathcal{R}^0(\Gamma) = \Gamma$
$\mathcal{R}^{n+1}(\Gamma) = \mathcal{R}(\mathcal{R}^n(\Gamma))$

Exemplo 5.3.14 *Tomando a resolução do exemplo* **??** *temos que* $\Gamma = \{p_1, \neg p_1, \neg p_1 \vee \neg p_2, p_1 \vee p_2\}$.

Por definição. $\mathcal{R}^0(\Gamma) = \Gamma$*. Assim, como* $\mathcal{R}^1(\Gamma) = \mathcal{R}(\mathcal{R}^0(\Gamma))$ *então* $\mathcal{R}^1(\Gamma) = \mathcal{R}(\Gamma)$*. E* $\mathcal{R}(\Gamma)$ *é a união de* Γ *com todos os resolventes básicos envolvendo cláusulas em* Γ*. Ou seja,* $\mathcal{R}(\Gamma) = \{\square, \neg p_2, p_2, p_2 \vee \neg p_2, p_1 \vee \neg p_1\} \cup \Gamma$*. Por tanto,*

$$\mathcal{R}^1(\Gamma) = \{\square, \neg p_2, p_2, p_2 \vee \neg p_2, p_1 \vee \neg p_1, p_1, \neg p_1, \neg p_1 \vee \neg p_2, p_1 \vee p_2\}$$

$\mathcal{R}^2(\Gamma) = \mathcal{R}(\mathcal{R}^1(\Gamma))$ *e* $\mathcal{R}(\mathcal{R}^1(\Gamma)) = \{\square, \neg p_2, p_2, p_2 \vee \neg p_2, p_1 \vee \neg p_1, p_1, \neg p_1, \neg p_1 \vee \neg p_2, p_1 \vee p_2\} \cup \mathcal{R}^1(\Gamma)$*. Por tanto, como não acrescentou novas cláusulas,* $\mathcal{R}^2(\Gamma) = \mathcal{R}^1(\Gamma)$*. Claramente, o mesmo vai ocorrer para os próximos* $\mathcal{R}^n$ *e portanto para todo* $n \geq 1$*,* $\mathcal{R}^n(\Gamma) = \mathcal{R}^1(\Gamma)$*.*

Proposição 5.3.15 *Seja* Γ *um conjunto de cláusulas.* $\alpha \in \mathcal{R}^n(\Gamma)$ *para algum* n *se, e somente se,* $\Gamma \xrightarrow{*} \alpha$*.*

Demonstração: Mostraremos, por indução em n que se $\alpha \in \mathcal{R}^n(\Gamma)$ então $\Gamma \xrightarrow{*} \alpha$.

Caso base: Se $n = 1$ então $\mathcal{R}^n(\Gamma) = \mathcal{R}(\Gamma)$, ou seja a união de Γ com todos seus resolventes básicos. Se $\alpha \in \mathcal{R}(\Gamma)$ então ou $\alpha \in \Gamma$, em cujo caso, a seqüência α é uma dedução por resolução de α a partir de Γ, isto é, $\Gamma \xrightarrow{*} \alpha$. Se α é um resolvente básico de de cláusulas em Γ, digamos α_i e α_j, então a seqüência, $\alpha_i, \alpha_j, \alpha$ é uma dedução por resolução de α a partir de Γ, isto é, $\Gamma \xrightarrow{*} \alpha$.

Etapa indutiva: Assuma que para cada $\alpha' \in \mathcal{R}^k(\Gamma)$ temos que $\Gamma \xrightarrow{*} \alpha'$. Se $\alpha \in \mathcal{R}^{k+1}(\Gamma)$ então $\alpha \in \mathcal{R}(\mathcal{R}^k(\Gamma))$. Logo, por definição de $\mathcal{R}$, ou $\alpha \in \mathcal{R}^k(\Gamma)$ ou é resolvente básico de algum $\alpha_i, \alpha_j \in \mathcal{R}^k(\Gamma)$. Em ambos os casos procedemos como no caso base.

Por outro lado, se $\Gamma \xrightarrow{*} \alpha$, então existe uma dedução por resolução, $\alpha_1, \ldots, \alpha_m$, de α a partir de Γ. Mostraremos por indução em i, com $1 \leq i \leq m$, que cada $\alpha_i \in \mathcal{R}^n(\Gamma)$ para algum n.

Caso base: Se $i = 1$, então necessariamente $\alpha_i \in \Gamma$. Logo, trivialmente, $\alpha_i \in \mathcal{R}^0\Gamma$.

Etapa indutiva: Assuma que para cada $i \leq k < m$, $\alpha_i \in \mathcal{R}^n(\Gamma)$ para algum n. Como α_{k+1} ou pertence a Γ ou é o resolvente básico de α_j e α_l, com $j, l \leq k$. No primeiro caso, trivialmente, $\alpha_{k+1} \in \mathcal{R}^0(\Gamma)$. No segundo caso, pela hipótese indutiva temos que $\alpha_j \in \mathcal{R}^{n1}(\Gamma)$ e $\alpha_l \in \mathcal{R}^{n2}(\Gamma)$. Como, por definição de $\mathcal{R}^n$, $\mathcal{R}^p(\Gamma) \subseteq \mathcal{R}^q(\Gamma)$ sempre que $p \leq q$, então $\mathcal{R}^{n1}(\Gamma) \subseteq \mathcal{R}^{max(n1,n2)}(\Gamma)$ e $\mathcal{R}^{n2}(\Gamma) \subseteq \mathcal{R}^{max(n1,n2)}(\Gamma)$. Logo, $\alpha_j, \alpha_l \in \mathcal{R}^{max(n1,n2)}(\Gamma)$. Portanto, $\alpha_{k+1} \in \mathcal{R}(\mathcal{R}^{max(n1,n2)}(\Gamma))$, ou seja $\alpha_{k+1} \in \mathcal{R}^{max(n1,n2)+1}(\Gamma)$. ■

Corolário 5.3.16 *Seja Γ um conjunto de cláusulas. $\square \in \mathcal{R}^n(\Gamma)$ para algum n se, e somente se, $\Gamma \xrightarrow{*} \square$.*

Demonstração: Direto do teorema anterior fazendo α a cláusula vazia ($\square$). ■

Por definição, $\Gamma \subseteq \mathcal{R}(\Gamma) \subseteq \mathcal{R}^2(\Gamma) \subseteq \cdots$. Além disso, cada $\mathcal{R}^n(\Gamma) \subseteq \Gamma^*$, o conjunto de todas as possíveis cláusulas usando literais contidos em Γ, pois nenhum literal novo pode ser gerado numa resolução básica. Logo, como Γ^* é finito, então para algum inteiro m, $\mathcal{R}^m(\Gamma) = \mathcal{R}^k(\Gamma)$ para todo $k > m$. Assim, a operação de tomar resolventes encerra depois de alguma quantidade finita de passos. Se $\square \in \mathcal{R}^m(\Gamma)$, então existe uma refutação de Γ (corolário 5.3.16), caso contrário nenhuma refutação de Γ existe. Assim, dado qualquer conjunto Γ de cláusulas, quando ou não $\Gamma \xrightarrow{*} \square$ é decidível.

5.4 Resultados de Completude

Nesta seção investigaremos as relações entre as três maneiras de abordar a lógica proposicional vistas até o momento: conseqüência lógica, representado pelo símbolo $\models$; provabilidade, representado por $\vdash$; e refutabilidade através do método **ELC**, representado pelo símbolo $\xrightarrow{*}$. O primeiro é o método matemático ou semântico, o segundo é o método formal ou dedutivo e o terceiro é o método computacional. Primeiramente, mostraremos que todo teorema de T_P é **logicamente válido**, isto é, uma tautologia. Para isso mostraremos, preliminarmente, o seguinte lema.

Lema 5.4.1 *Se α é um axioma de T_P, então α é uma tautologia.*

Demonstração: Para mostrar isto usaremos as tabelas verdade.

1. O axioma A_1, isto é:

$$\alpha_1 \to (\alpha_2 \to \alpha_1),$$

é uma tautologia, pois basta observar a sua tabela verdade

α_1	$\to$	$(\alpha_2$	$\to$	$\alpha_1)$
1	1	1	1	1
1	1	0	1	1
0	1	1	0	0
0	1	1	1	0

2. O axioma A_2, isto é,

$$(\alpha_1 \to (\alpha_2 \to \alpha_3)) \to ((\alpha_1 \to \alpha_2) \to (\alpha_1 \to \alpha_3))$$

é uma tautologia, pois basta observar a sua tabela verdade.

$(\alpha_1$	$\to$	$(\alpha_2$	$\to$	$\alpha_3))$	$\to$	$((\alpha_1$	$\to$	$\alpha_2)$	$\to$	$(\alpha_1$	$\to$	$\alpha_3))$
1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	0	0
1	1	0	1	1	1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	0	0	1	1	0	0
0	1	1	1	1	1	0	1	1	1	0	1	1
0	1	1	0	0	1	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1	0	1	0	1	1
0	1	0	1	1	1	0	1	0	1	0	1	0

3. O axioma A_3, isto é,

$$(\neg\alpha_2 \to \neg\alpha_1) \to ((\neg\alpha_2 \to \alpha_1) \to \alpha_2),$$

é uma tautologia, pois basta observar a sua tabela verdade.

$(\neg$	α_2	$\to$	$\neg$	$\alpha_1)$	$\to$	$((\neg$	α_2	$\to$	$\alpha_1)$	$\to$	$\alpha_2)$
0	1	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	0	1	1	0	0
0	1	1	1	0	1	0	1	1	0	1	1
1	0	1	1	0	1	1	0	0	0	1	0

Teorema 5.4.2 (T_P é segura) *Se $\vdash \alpha$ então $\models \alpha$. Isto é, se α é provável, então α é uma tautologia.*

DEMONSTRAÇÃO: Se $\vdash \alpha$, então existe uma derivação ou prova de α, digamos $\alpha_1, \alpha_2, \ldots, \alpha_n$, onde $\alpha_n = \alpha$ e tal que para todo i, $1 \leq i \leq n$, α_i é um axioma ou segue por MP de α_j e α_k, onde $\alpha_k = \alpha_j \rightarrow \alpha_i$ e $j, k < i$.

Mostraremos, por indução sobre i, que cada α_i na derivação é uma tautologia.

Caso base: $i = 1$. Por definição de prova, α_1 deve ser um axioma, e portanto pelo lema 5.4.1, uma tautologia.

Etapa indutiva: Suponha que α_i é uma tautologia para todo i, $1 \leq i < k$. Devemos mostrar que α_k é uma tautologia.

1. Se α_k é um axioma, pelo lema 5.4.1, é uma tautologia.
2. Se α_k segue por modus ponens de α_i e α_j, onde $i, j < k$ e $\alpha_j = \alpha_i \rightarrow \alpha_k$ então, como α_i e α_j são tautologias, pela hipótese indutiva, α_k é uma tautologia pela definição de $\rightarrow$.

Portanto, α_i é uma tautologia para todo i, $1 \leq i \leq n$. Como $\alpha = \alpha_n$, segue que α é uma tautologia. ■

Corolário 5.4.3 *Se* $\Gamma \vdash \alpha$ *então* $\Gamma \models \alpha$.

DEMONSTRAÇÃO: Se $\Gamma \vdash \alpha$ então pela observação 6.c do capítulo 4 (teorema da compacidade) temos que $\Gamma_0 \vdash \alpha$ para algum $\Gamma_0 \subseteq \Gamma$, finito. Suponha que $\Gamma_0 = \{\alpha_1, \ldots, \alpha_n\}$, então, pelo teorema da dedução, temos que $\vdash \alpha_1 \rightarrow (\alpha_2 \rightarrow (\ldots(\alpha_n \rightarrow \alpha)\ldots)$. Logo, pelo teorema seguro temos que $\models \alpha_1 \rightarrow (\alpha_2 \rightarrow (\ldots(\alpha_n \rightarrow \alpha)\ldots)$. Aplicando o teorema da dedução (repetidas vezes) temos que $\Gamma_0 \models \alpha$. Como $\Gamma_0 \subseteq \Gamma$, então pelo exercício 9.a do capítulo 3, temos que $\Gamma \models \alpha$. ■

Assim, dedutibilidade implica verdade, significando que a teoria formal T_P é segura e o corolário nos diz que a noção de conseqüência na teoria formal T_P se corresponde com a noção de conseqüência semântica ou lógica. Vejamos, agora, verdade e computação. Pretendemos mostrar que se α é uma tautologia, então podemos achar uma refutação para $\neg\alpha$. Se α é uma tautologia, então $\neg\alpha$ é insatisfatível. Queremos mostrar que se $\neg\alpha$ é insatisfatível, então $\neg\alpha \xrightarrow{*} \square$.

Assumiremos que a fórmula $\neg\alpha$ está na forma normal conjuntiva reduzida. Portanto, $\neg\alpha$ é um conjunto de cláusulas, o qual entendemos como a conjunção dessas cláusulas.

Definição 5.4.4 *Se* Γ *é um conjunto de cláusulas, a frase "*Γ **com** α **substituído por** β*" denota o conjunto* $(\Gamma - \{\alpha\}) \cup \{\beta\}$.

Ou seja, a cláusula α é removida do conjunto de cláusulas Γ e então a cláusula β é adicionada ao resultado.

Lema 5.4.5 *Se* Γ *é um conjunto insatisfatível de cláusulas básicas com* $\alpha \in \Gamma$, $\beta \subseteq \alpha$[2] *e* $\beta \neq \emptyset$, *então* Γ, *com* α *substituído por* β, *é insatisfatível.*

DEMONSTRAÇÃO: Por definição de cláusula, α é uma disjunção de literais. Como $\beta \subseteq \alpha$, β é uma disjunção de literais. Se Γ, com α substituído por β, é satisfatível, então existe uma interpretação sob a qual Γ com α substituído por β é verdadeira. Como Γ é uma conjunção de cláusulas, sabemos que β deve ser verdadeira sob esta interpretação. Logo, como β é uma disjunção de literais, ela é verdadeira, somente se um dos literais em β é verdadeiro. Como $\beta \subseteq \alpha$, a mesma interpretação torna α verdadeira e portanto Γ é satisfatível, contradizendo nossa suposição inicial. Portanto, Γ com α substituído por β é insatisfatível. ■

Este lema nos assegura que um conjunto de cláusulas insatisfatível não pode ser satisfatível eliminando alguns literais em suas cláusulas. E eliminando a cláusula toda? Como uma cláusula é uma disjunção, eliminando elementos (literais) torna-a mais distante de ser satisfatível. Mas, como uma fórmula é uma conjunção de cláusulas, eliminar alguma de suas cláusulas, torna-a mais provável de ser satisfatível. Por este motivo que colocamos a restrição no lema anterior, de β ser diferente de vazio.

Definição 5.4.6 *Um conjunto* Γ *de cláusulas é um* **conjunto minimal insatisfatível** *se* Γ *é insatisfativel, mas qualquer subconjunto próprio de* Γ *não é insatisfatível.*

Observação: Pelo teorema da compacidade o conjunto minimal insatisfatível é finito.

Todo conjunto insatisfatível contém um subconjunto minimal insatisfatível. Mas, esse conjunto não é necessariamente único.

Exemplo 5.4.7 $\{p_1, p_2, \neg p_1, \neg p_2\}$ *tem dois subconjuntos minimal insatisfatível, quais sejam* $\{p_1, \neg p_1\}$ *e* $\{p_2, \neg p_2\}$.

No sentido de simplificar a prova do próximo lema, introduziremos as noções de **cláusula unitária, multiunitária e não unitária**, assim como o conceito de **excesso de literais**.

Definição 5.4.8 *Uma cláusula que contém exatamente um literal é uma* **cláusula unitária**. *Uma* **cláusula multiunitária** *é aquela que contém dois ou mais literais. Uma cláusula* **não unitária** *é ou a cláusula vazia ou uma cláusula multiunitária.*

Definição 5.4.9 *Seja* Γ *um conjunto finito de cláusulas. Denote por* $\#\Gamma$ *o* **número de cláusulas** *no conjunto* Γ, *e* $\#\#\Gamma$ *o* **número de ocorrência de literais** *em* Γ. *O número de* **excesso de literais** *em* Γ *é* $\#\#\Gamma - \#\Gamma$, *que é não negativo se* $\square \notin \Gamma$.

[2]Lembre-se que α é uma cláusula e portanto pode ser visto como um conjunto.

Exemplo 5.4.10

- $\Gamma = \{p_1 \vee p_2, \neg p_1 \vee p_3, \neg p_2 \vee p_3, \neg p_3\}$
- $\#\Gamma = 4$ *(o número de cláusulas em* Γ*)*
- $\#\#\Gamma = 7$ *(o número de literais em* Γ*)*
- $\#\#\Gamma - \#\Gamma = 3$ *(o número de excesso de literais em* Γ*)*

$\Box$ é a única cláusula sem literais. Logo se $\Box \notin \Gamma$, Γ tem no mínimo um literal em cada cláusula, tornando $\#\#\Gamma \geq \#\Gamma$ e o número de excesso de literais não negativo.

Lema 5.4.11 *Se* Γ *é um conjunto, não vazio, minimal insatisfatível de cláusulas básicas, então* $\Gamma \xrightarrow{*} \Box$.

Demonstração: Se $\Box \in \Gamma$, então $\Gamma = \{\Box\}$ uma vez que Γ é minimal, e $\{\Box\} \xrightarrow{*} \Box$. Suponha, agora, que $\Box \notin \Gamma$ e seja n o número de excesso de literais em Γ, isto é, $n = \#\#\Gamma - \#\Gamma$. Mostraremos, por indução sobre n, que $\Gamma \xrightarrow{*} \Box$.

Caso base: $n = 0$. Se o número n de excesso de literais é 0, então, necessariamente, todas as cláusulas em Γ são não unitárias. Logo, Γ é da forma $\{p_{i_1}, \neg p_{i_1}, p_{i_2}, \neg p_{i_2}, \ldots, p_{i_k}, \neg p_{i_k}\}$ onde cada p_{i_j}, com $1 \leq j \leq k$, é uma proposição atômica. Neste caso, a resolução básica sobre duas cláusulas unitárias complementares, p_{i_1} e $\neg p_{i_1}$ por exemplo, gera $\Box$.

Etapa indutiva: Suponha que $\Gamma \xrightarrow{*} \Box$ para todo conjunto minimal insatisfatível de cláusulas básicas Γ com k excessos de literais, para todo $0 < k < n$. Devemos mostrar que $\Gamma \xrightarrow{*} \Box$, onde Γ é um conjunto minimal insatisfatível de cláusulas básicas, com n excessos de literais.

Como $n > 0$, existe uma cláusula multiunitária α, em Γ. Seja $\alpha_1 \in \alpha$ e Γ_1 igual a Γ com α substituído por $\alpha - \{\alpha_1\}$. Pelo lema 5.4.5, Γ_1 é insatisfatível. Seja Γ' um subconjunto minimal insatisfatível de Γ_1. Agora Γ_1 (e portanto Γ') tem menos que n excessos de literais. Pela hipótese indutiva, $\Gamma' \xrightarrow{*} \Box$ por alguma refutação de solução $\beta_1, \beta_2, \ldots, \beta_{k-1}, \beta_k = \Box$. Algum β_i é $\alpha - \{\alpha_i\}$, caso contrário Γ não seria minimal. Tomamos $\beta_1, \ldots, \beta_k$ numa dedução de Γ substituindo α por $\alpha - \{\alpha_i\}$, para gerar α, e adicionando subseqüentes resolventes para incluir α, se necessário. Seja $\beta'_1, \ldots, \beta'_k$, denotando a dedução de Γ, assim obtida. Se $\beta'_k = \Box$ então tudo bem. Caso contrário, $\beta_k = \{\alpha\}$, e devemos obter uma refutação de $A \cup \{\alpha\}$ para algum subconjunto A de Γ e juntar ele com $\beta'_1, \ldots, \beta'_k$ para obter uma refutação de Γ.

Seja Γ_2 igual a Γ com α substituído por $\{\alpha_1\}$. Γ_2 é insatisfatível novamente pelo lema 5.4.5. Seja Γ'_2 um subconjunto minimal de Γ_2. Agora, Γ'_2 é $\Theta - \{\alpha_1\}$ para algum $\Theta \subset \Gamma$, desde que $\{\alpha_1\}$ deve estar em Γ'_2 (Caso contrário Γ'_2 seria um subconjunto próprio de Γ, contradizendo a hipótese de que Γ é um subconjunto minimal insatisfatível).

$\alpha \cup \{\alpha_1\}$ tem menos elementos que n excessos de literais, portanto pela hipótese de indução existe uma refutação $\gamma_1, \ldots, \gamma_r$ de $\Theta \cup \{\alpha_1\}$. Então $\beta'_1, \ldots, \beta'_k, \gamma_1, \ldots, \gamma_r$ é a pretendida refutação de Γ. ■

Teorema 5.4.12 $\models \alpha$ *implica* $\neg\alpha \xrightarrow{*} \square$. *Ou seja, se* α *é uma tautologia (logicamente válida), então existe uma refutação de* $\neg\alpha$.

DEMONSTRAÇÃO: $\models \alpha$ se e somente se $\neg\alpha$ é insatisfatível. Se $\neg\alpha$ é insatisfatível então existe um subconjunto minimal insatisfatível Γ de $\neg\alpha$. Pelo lema 5.4.11, $\Gamma \xrightarrow{*} \square$. Portanto, pela mesma refutação, $\neg\alpha \xrightarrow{*} \square$. ■

Corolário 5.4.13 *Se* Γ *é um conjunto insatisfatível de cláusulas básicas, então* $\Gamma \xrightarrow{*} \square$.

DEMONSTRAÇÃO: Se Γ é um conjunto insatisfatível de cláusulas básicas, então sua negação, isto é $\neg\Gamma$, é uma tautologia, isto é $\models \neg\Gamma$. Logo, pelo teorema anterior, temos que $\neg\Gamma \xrightarrow{*} \square$. ■

Assim, se uma fórmula α é uma tautologia (válida) podemos computar uma refutação de $\neg\alpha$. Isto significa que nosso esquema de computação é **completo**. Agora generalizemos este resultado, ligando a noção de consequência lógica com dedução por resolução.

Proposição 5.4.14 *Seja* Γ *uma conjunto de cláusulas e* α *uma cláusula. Se* $\Gamma \models \alpha$ *então* $\Gamma \xrightarrow{*} \alpha$. *Ou seja, se* α *é conseqüência lógica de* Γ, *então existe uma dedução por resolução de* α *a partir de* Γ.

DEMONSTRAÇÃO: Se $\Gamma \models \alpha$ então, pelo teorema da compacidade, existe $\Gamma_0 \subseteq \Gamma$ finito tal que $\Gamma_0 \models \alpha$. Seja $\Gamma_0 = \{\alpha_1, \ldots, \alpha_n\}$. Pelo teorema da dedução temos que: $\models \alpha_1 \rightarrow (\ldots(\alpha_n \rightarrow \alpha)\ldots)$, isto é, $\models \neg\alpha_1 \vee \ldots \vee \neg\alpha_n \vee \alpha$. Logo, pelo teorema 5.4.12, $\neg(\neg\alpha_1 \vee \ldots \vee \neg\alpha_n \vee \alpha) \xrightarrow{*} \square$. Suponha ainda que a cláusula α é $\mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k$. Assim, $\alpha_1, \ldots, \alpha_n, \mathtt{l}_1^{op}, \ldots, \mathtt{l}_k^{op} \xrightarrow{*} \square$. Aplicando o teorema da dedução simples (proposição 5.3.7) k-vezes, temos que $\alpha_1, \ldots, \alpha_n \xrightarrow{*} \mathtt{l}_1 \vee \ldots \vee \mathtt{l}_k$, ou seja, $\Gamma_0 \xrightarrow{*} \alpha$. Aplicando o teorema da compacidade (proposição 5.3.7), temos que $\Gamma \xrightarrow{*} \alpha$. ■

Note que o fato desta proposição requer que em $\Gamma \models \alpha$, todos sejam somente cláusulas, não tira a generalidade do mesmo, pois se Γ é um conjunto de fórmula quaisquer, então cada uma das fórmulas em Γ pode ser transformada numa fncr equivalente, e portanto geraria um conjunto de cláusulas. Unindo todos esses conjuntos de cláusulas teríamos um conjunto de cláusulas Γ' equivalente ao conjunto de fórmulas Γ. Analogamente, se α for uma fórmula qualquer ela pode ser transformada à fncr, suponha que a fncr de α seja $\alpha'_1 \wedge \ldots \wedge \alpha'_m$. Então em vez de $\Gamma \models \alpha$ teríamos $\Gamma' \models \alpha'_1$ e $\ldots$ e $\Gamma' \models \alpha'_m$, satisfazendo a condição da proposição acima.

Até aqui sabemos que dedutibilidade implica verdade, e verdade implica computação. Comparemos, agora, computação e dedução.

Começaremos estabelecendo que se β é um resolvente básico de cláusulas num conjunto de cláusulas α, podemos provar que $\vdash \alpha \rightarrow \beta$. A partir daqui generalizamos, considerando o caso no qual existe uma dedução por resolução de β a partir de α (possivelmente envolvendo várias etapas de **ELC**).

Lema 5.4.15 *Seja α uma fórmula na fncr. Se β é um resolvente básico de duas cláusulas em α, então $\vdash \alpha \to \beta$. Isto é, se $\alpha \xrightarrow{*} \beta$ usando uma única resolução básica, então $\vdash \alpha \to \beta$.*

DEMONSTRAÇÃO: Se $\alpha \xrightarrow{*} \beta$ usando uma única resolução básica, então α é da forma $(\alpha_1 \vee p) \wedge (\neg p \vee \alpha_2) \wedge \alpha_3$ onde p é uma proposição atômica, α_1 e α_2 cláusulas, α_3 é uma conjunção de cláusulas, e $\beta = \alpha_1 \vee \alpha_2$. Pretendemos mostrar que

$$(\alpha_1 \vee p) \wedge (\neg p \vee \alpha_2) \wedge \alpha_3 \vdash \beta,$$

o qual, por definição de $\vee$ como uma abreviação, podemos também escrever

$$(\neg\alpha_1 \to p) \wedge (\neg\neg p \to \alpha_2) \wedge \alpha_3 \vdash (\neg\alpha_1 \to \alpha_2)$$

1.	$\neg\alpha_1 \to p$	hipótese
2.	$p \to \neg\neg p$	proposição 4.3.3.b
3.	$\neg\alpha_1 \to \neg\neg p$	1,2, proposição 4.3.2.a
4.	$\neg\neg p \to \alpha_2$	hipótese
5.	$\neg\alpha_1 \to \alpha_2$	3,4, proposição 4.3.2.a

Assim, temos que $\neg\alpha_1 \to p, \neg\neg p \to \alpha_2 \vdash \neg\alpha_1 \to \alpha_2$. Logo,

6.	$(\neg\alpha_1 \to p) \wedge (\neg\neg p \to \alpha_2) \vdash \neg\alpha_1 \to \alpha_2$	Exercício 5 do capítulo 4
7.	$(\neg\alpha_1 \to p) \wedge (\neg\neg p \to \alpha_2), \alpha_3 \vdash \neg\alpha_1 \to \alpha_2$	propriedade 1 de dedutibilidade
8.	$(\neg\alpha_1 \to p) \wedge (\neg\neg p \to \alpha_2) \wedge \alpha_3 \vdash \neg\alpha_1 \to \alpha_2$	

isto é, $\alpha \vdash \beta$, e pelo teorema da dedução para T_P temos $\vdash \alpha \to \beta$. ■

Lema 5.4.16 $\Gamma \xrightarrow{*} \beta$ *implica* $\Gamma \vdash \beta$.

DEMONSTRAÇÃO: $\Gamma \xrightarrow{*} \beta$ implica que uma dedução por resolução de β a partir de Γ existe. (Isto é, existe uma seqüência, $\alpha_1, \ldots, \alpha_n$ tal que $\alpha_n = \beta$ e para todo i, $1 \leq i \leq n$, α_i ou é uma cláusula de Γ ou é um resolvente básico de dois α_i's prévios na seqüência).

Mostremos, por indução sobre i, que para todo i, $\Gamma \vdash \alpha_i$.

Caso base: $i = 1$. α_1 é uma cláusula de Γ. Então, trivialmente, a seqüência α_1 é uma prova de $\Gamma \vdash \alpha_1$.

Etapa indutiva: Assuma que $\Gamma \vdash \alpha_i$, para cada $i < k$. Devemos mostrar que $\Gamma \vdash \alpha_k$.

1. Se α_k é uma cláusula de Γ, então, como no caso para $i = 1$, temos que trivialmente, $\Gamma \vdash \alpha_k$.

2. Se α_k é um resolvente básico de duas cláusulas anteriores na seqüência, suponha que de α_i e α_j, onde $i, j < k$, então $\alpha_k = (\alpha_i - \mathtt{l}) \cup (\alpha_j - \overline{\mathtt{l}})$ para algum literal $\mathtt{l}$[3]. Pela hipóteses indutiva temos que $\Gamma \vdash \alpha_i$ e $\Gamma \vdash \alpha_j$. Observe que $\mathtt{l} \vee \alpha \equiv \overline{\mathtt{l}} \rightarrow \alpha$. Assim, $\alpha_i \equiv \overline{\mathtt{l}} \rightarrow (\alpha_i - \mathtt{l})$ e $\alpha_j \equiv \mathtt{l} \rightarrow (\alpha_j - \overline{\mathtt{l}})$. Portanto $\Gamma \vdash \overline{\mathtt{l}} \rightarrow (\alpha_i - \mathtt{l})$ e $\Gamma \vdash \mathtt{l} \rightarrow (\alpha_j - \overline{\mathtt{l}})$. Pelo exercício 2.f. e 2.g. do capítulo 4, temos que $\Gamma \vdash \neg(\alpha_i - \mathtt{l}) \rightarrow (\alpha_j - \overline{\mathtt{l}})$. Ou seja $\Gamma \vdash (\alpha_i - \mathtt{l}) \cup (\alpha_j - \overline{\mathtt{l}})$. Portanto, $\Gamma \vdash \alpha_k$.

Para $i = n$, temos $\Gamma \vdash \alpha_n$. Como $\alpha_n = \beta$, então $\Gamma \vdash \beta$. Portanto, $\Gamma \xrightarrow{*} \beta$ implica $\Gamma \vdash \beta$. ■

Corolário 5.4.17 *Se $\neg\alpha \xrightarrow{*} \beta$ então $\vdash \neg\alpha \rightarrow \beta$.*

DEMONSTRAÇÃO: Se $\neg\alpha \xrightarrow{*} \beta$, então pelo lema acima, fazendo $\Gamma = \{\neg\alpha\}$, temos que $\neg\alpha \vdash \beta$. Aplicando o teorema da dedução temos que $\vdash \neg\alpha \rightarrow \beta$. ■

Teorema 5.4.18 *Se existe uma refutação de $\neg\alpha$ (por* **ELC***), então podemos provar α em T_P. Isto é, $\neg\alpha \xrightarrow{*} \Box$ implica $\vdash \alpha$.*

DEMONSTRAÇÃO: Se $\neg\alpha \xrightarrow{*} \Box$, então existe uma dedução por resolução $\alpha_1, \ldots, \alpha_n = \Box$ a partir de $\neg\alpha$.

1. Se $\Box \in \neg\alpha$ então $\neg\alpha = \{\Box\}$, uma vez que $\neg\alpha$ está na forma normal conjuntiva reduzida. Mas, então $\alpha = \neg(\neg\alpha) = \neg\Box = \tau$. Lembre-se que τ é uma abreviação para $\alpha \vee \neg\alpha$, o qual, por definição, é equivalente a : $\neg\alpha \rightarrow \neg\alpha$. Portanto, α implica $\neg\alpha \rightarrow \neg\alpha$, o qual é provável pelo lema **??**.

2. $\Box$ deve ser resolvente básico de α_i e α_j, $i, j < n$. Lembre-se que $\Box$ é uma abreviação para $\alpha \wedge \neg\alpha$. Logo pelo corolário 5.4.17, temos $\vdash \neg\alpha \rightarrow (\alpha_1 \wedge \neg\alpha_1)$.

1.	$\vdash \neg\alpha \rightarrow \neg(\alpha_1 \rightarrow \neg\neg\alpha_1)$	definição de $\wedge$
2.	$\vdash \alpha \rightarrow \neg\neg\alpha$	proposição 4.3.3.b
3.	$\vdash (\neg\alpha \rightarrow \neg(\alpha_1 \rightarrow \neg\neg\alpha_1)) \rightarrow ((\alpha_1 \rightarrow \neg\neg\alpha_1) \rightarrow \alpha)$	proposição 4.3.3.d
4.	$\vdash (\alpha_1 \rightarrow \neg\neg\alpha_1) \rightarrow \alpha$	MP 1,3
5.	$\vdash \alpha$	MP 2,4. ■

Teorema 5.4.19 (Teorema da Completude) *Na lógica proposicional*

$$\vdash \alpha \text{ se e somente se } \models \alpha \text{ se e somente se } \neg\alpha \xrightarrow{*} \Box$$

[3]Se $\mathtt{l}$ é a fórmula atômica p então $\overline{\mathtt{l}}$ é a fórmula $\neg p$. Se $\mathtt{l}$ é a fórmula $\neg p$ então $\overline{\mathtt{l}}$ é a fórmula atômica $\neg p$.

DEMONSTRAÇÃO:

$\vdash \alpha$	*implica*	$\models \alpha$	teorema 5.4.2
$\models \alpha$	*implica*	$\neg\alpha \xrightarrow{*} \square$	teorema 5.4.12
$\neg\alpha \xrightarrow{*} \square$	*implica*	$\vdash \alpha$	teorema 5.4.18. ■

Conseqüentemente, na lógica proposicional uma fórmula é provável se e somente se ela é logicamente válida (isto é, é uma tautologia) ou, equivalentemente, se e somente se sua negação se reduz à cláusula vazia pela resolução básica (**ELC**).

Corolário 5.4.20 (Teorema da Completude Forte) *Na lógica proposicional*

$$\Gamma \vdash \alpha \text{ se e somente se } \Gamma \models \alpha \text{ se e somente se } \Gamma \xrightarrow{*} \alpha$$

DEMONSTRAÇÃO: Direto do corolário 5.4.3, da proposição 5.4.14 e do lema 5.4.16. ■

Observe que, entre outras coisas, o teorema da completude nos permite substituir a tediosa e difícil tarefa de provar que uma fórmula é um teorema na teoria formal T_P, por simplesmente, mostrar que a fórmula é uma tautologia, a qual é uma tarefa menos tediosa e principalmente automática, ou melhor ainda, por simplesmente mostrar que existe uma refutação de sua negação, o qual pode também ser realizado automaticamente e com um custo computacional menor que usando tabelas verdade. Assim, o teorema da completude interliga as três dimensões da lógica proposicional vistas aqui, e desta maneira podemos passar, por exemplo, resultados obtidos na dimensão semântica, para a dimensão sintática ou computacional, isto é, todo o que pode ser demonstrado numa dimensão, pode ser demonstrados nas outras. O corolário 5.4.20 estende este resultado para ligar dedução a partir de um conjunto de premissas com conseqüência lógica e com dedução por resolução de uma cláusula a partir de um conjunto de cláusulas.

5.5 Exercícios

1. Exibir uma refutação da **negação** de cada uma das seguintes proposições. (Observe que isto requer que cada fórmula seja negada e então convertida para fncr antes de começar a resolução).

   a. $p_1 \rightarrow p_1$

   b. $p_1 \rightarrow \neg\neg p_1$

   c. $((p_1 \rightarrow p_2) \rightarrow p_3) \rightarrow ((p_1 \rightarrow p_2) \rightarrow (p_1 \rightarrow p_3))$

   d. $(\neg p_1 \rightarrow p_2) \rightarrow (\neg p_2 \rightarrow p_1)$

   e. $(\neg p_2 \rightarrow \neg p_1) \rightarrow ((\neg p_2 \rightarrow p_1) \rightarrow p_2)$

f. $(p_1 \to \neg\neg p_1) \wedge (\neg\neg p_1 \to p_1)$

g. $\neg(p_1 \to \neg p_2) \to p_2$

h. $p_1 \wedge p_2 \to p_2$

i. $\neg(p_1 \to \neg p_2) \to p_1$

j. $p_1 \wedge p_2 \to p_1$

k. $p_1 \to (\neg\neg(p_1 \to \neg p_2) \to \neg p_2)$

l. $(p_1 \to p_2) \wedge (p_1 \to p_3) \to (p_1 \to p_2 \wedge p_3)$

m. $(p_1 \to p_2) \to (p_3 \vee p_1 \to p_3 \vee p_2)$

n. $(p_1 \to p_2) \to ((p_3 \to p_1) \to (p_3 \to p_2))$

o. $(p_1 \to p_2) \wedge (p_2 \to p_3) \to (p_1 \to p_3)$

p. $\neg p_1 \to (p_1 \to p_2)$

q. $p_1 \vee (p_2 \vee p_3) \to (p_2 \vee (p_1 \vee p_3)) \vee p_1$

r. $(p_2 \vee (p_1 \vee p_3)) \vee p_1 \to p_2 \vee (p_1 \vee p_3)$

s. $p_1 \vee (p_2 \vee p_3) \to p_2 \vee (p_1 \vee p_3)$

t. $\neg(\neg p_1 \wedge p_1)$

u. $\neg(p_1 \wedge p_2) \to (p_2 \to \neg p_1)$

v. $(p_1 \to p_2) \to (\neg p_2 \to \neg p_1)$

w. $(\neg p_1 \to \neg p_2) \to (p_2 \to p_1)$

x. $(p_1 \to p_2) \to (p_3 \wedge p_1 \to p_2 \wedge p_3)$

y. $p_1 \wedge p_2 \to p_2 \wedge p_1$

z. $(p_1 \to (p_2 \to p_3)) \wedge p_2 \to (p_1 \to p_3)$

aa. $\neg p_1 \to (p_1 \to p_2)$

ab. $(p_1 \to p_2) \to (\neg p_2 \to \neg p_1)$

ac. $p_1 \to (\neg p_2 \to \neg(p_1 \to p_2))$

ad. $(p_1 \to p_2) \to ((\neg p_1 \to p_2) \to p_2)$

ae. $(\neg p_1 \to \neg p_2) \to (p_2 \to p_1)$

af. $(\neg p_1 \to p_1) \to p_1$

ag. $p_1 \to (\neg p_1 \to p_2)$

ah. $(p_1 \to p_2) \to ((p_2 \to p_3) \to (p_1 \to p_3))$

2. Exibir uma refutação por resolução dos seguintes conjuntos de cláusulas

   (a) $\{p_1 \vee p_2 \vee p_3, \neg p_1 \vee p_2, \neg p_3 \vee p_1, \neg p_2\}$

   (b) $\{\neg p_1 \vee p_2, \neg p_2 \vee p_3, \neg p_3 \vee \neg p_1, p_1\}$

(c) $\{p_1 \vee p_2, \neg p_3, \neg p_2, p_2 \vee p_3 \vee \neg p_1\}$

3. Mostre que $\Gamma \xrightarrow{*} \alpha$,quando:

   (a) $\Gamma = \{p_1 \vee p_2 \vee p_3, \neg p_3 \vee p_4, \neg p_1, \neg p_2 \vee p_3\}$ e $\alpha = p_3 \vee p_4$

   (b) $\Gamma = \{\neg p_1 \vee p_2, \neg p_2 \vee p_3, \neg p_3 \vee p_1 \vee p_4, \neg p_4\}$ e $\alpha = \neg p_2 \vee p_3 \vee p_4$

   (c) $\Gamma = \{p_4 \vee p_3, \neg p_1 \vee \neg p_4 \vee p_3, \neg p_3 \vee p_4\}$ e $\alpha = \neg p_1 \vee p_4$

4. Para os conjuntos de cláusulas Γ do exercício anterior, determine Γ, $\#\Gamma$ e $\#\#\Gamma$.

5. Demonstre que se $\Gamma \xrightarrow{*} \alpha$, para cada $\alpha \in \Theta$ e $\Theta \xrightarrow{*} \beta$ então $\Gamma \xrightarrow{*} \beta$. (Lei do corte)

Capítulo 6

Lógica de Predicados: Linguagem e Semântica

A linguagem lógica vista até aqui não é suficientemente expressiva para analisar asserções como "todo estudante gosta de tirar boas notas" ou "não existem corvos azuis". Para tal precisamos ter um mecanismo que nos permita percorrer o universo dos indivíduos sobre os quais se esta afirmando alguma coisa (nas asserções acima, o universo seria o dos estudantes e o dos corvos, respectivamente). Considere agora a sentença "o gato é magro", ou em notação lógica, $magro(gato)$. Esta sentença é verdadeira ou falsa? Antes de responder essa questão precisamos definir a palavra gato de modo que saibamos exatamente o que gato significa. Agora compare esta sentença com "algo é magro". "Algo" aqui é bastante diferente de "gato". Para testar a verdade dessa sentença não precisamos conhecer de antemão exatamente o que "algo" é. Precisamos saber o conjunto o qual "magro" percorre e então observar cada elemento desse conjunto para verificar se pelo menos um é magro. Caso sejamos bem sucedido então "algo é magro" é verdadeira. Por isso, em lógica não escrevemos $magro(algo)$, mas $\exists x.magro(x)$. Isto pode ser lido literalmente como "para algum x, x é magro", o qual é obviamente equivalente a "algo é magro".

O significado dessa sentença é

> *existe um valor, o qual quando substituído por x em $magro(x)$, gera uma proposição verdadeira.*

Considere, por exemplo, a asserção "o gato é magro e faminto", que na linguagem lógica pode ser escrito por $magro(gato) \wedge faminto(gato)$. Como o significado de gato é fixo de antemão ambas as ocorrências de "o gato" se referem à mesma coisa. Por outro lado, em "algo é magro" e "algo é faminto", que em linguagem lógica é $\exists x.magro(x) \wedge \exists y.faminto(y)$, as duas coisas poderiam ser diferentes. é também possível dizer "algo é ambos magro e faminto", isto é, $\exists x.(magro(x) \wedge faminto(x))$. Neste caso, o que quer que seja o que estamos nos referindo é magro e faminto.

Diferentemente de gato, que é uma **constante**, x tem o potencial para variar e portanto é chamado **variável**. O x em $\exists x$ é uma variável e aplica-se a toda a fórmula que segue o ponto. Para fórmulas não atômicas o uso de parênteses é necessário para delimitar o escopo de x. Por exemplo, $\exists x.(P(x) \vee Q(x))$. A ocorrência de $\exists x$ **liga** as ocorrências de x naquela fórmula.

Os símbolos $\exists$ e $\forall$ são chamados de **quantificadores**, enquanto que $\exists x$ e $\forall x$ são chamados de **quantificações**. O quantificador $\exists$ é chamado de **quantificador existencial**, pois pode ser lido como "existe um" ou "para algum". Já o quantificador $\forall$ é chamado de **quantificador universal**, pois pode ser lido como "para todo" ou "para cada". Por exemplo, em "Pedro gosta de todo mundo", não escrevemos $gosta(Pedro, todomundo)$, mas $\forall x.gosta(Pedro, x)$. Para ver se esta sentença é verdadeira (estamos usando o nome **sentença** como sinônimo de afirmação, embora sejam noções sutilmente diferentes) precisamos checar que Pedro gosta de todos os valores como especificados na abrangência. O significado dessa sentença é

> *Para todo valor substituído por x, $gosta(Pedro, x)$ é uma asserção verdadeira.*

É importante ter consciência de que quando temos duas ocorrências da mesma variável, ligada pela mesma quantificação elas devem denotar o mesmo valor. Por exemplo, os x's em $\exists x.(magro(x) \wedge faminto(x))$ deve denotar o mesmo valor. Para tornar a sentença verdadeira devemos achar um valor para x que seja ambos magro e faminto. Por outro lado em $\exists x.magro(x) \wedge \exists x.faminto(x)$ os dois x's são ligados por diferentes quantificações e precisamos achar algo que é magro e algo que é faminto. O mesmo ou diferente, não importa, para a sentença ser verdadeira. No caso, $\forall x.gosta(x, Pedro) \vee \forall x.gosta(x, Maria)$ é verdadeira se $\forall x.gosta(x, Pedro)$ ou $\forall x.gosta(x, Maria)$ é verdadeira. Aqui os dois x's são ligados por diferentes quantificações. Elas são realmente variáveis diferentes.

Na sentença "algo gosta de todo mundo", o qual é $\exists x.\forall y.gosta(x, y)$ as duas variáveis dos quantificadores aninhados são diferentes.

Quantificadores que são da mesma espécie podem ser colocados em qualquer ordem. Por exemplo,

$$\forall x.\forall y.(mãe(x, y) \rightarrow parente(x, y))$$

não é diferente de

$$\forall y.\forall x.(mãe(x, y) \rightarrow parente(x, y))$$

Eles significam que, para quaisquer x e y, se x é mãe de y então x é um parente de y. Generalizando, $\exists x.\exists y.\alpha$ significa o mesmo que $\exists y.\exists x.\alpha$.

Para quantificadores de espécies diferentes a ordem é importante. Por exemplo,

$$\forall y.\exists x.mãe(x, y)$$

não significa o mesmo que

$$\exists x.\forall y.mãe(x, y)$$

O primeiro diz que todo mundo tem uma mãe, literalmente, para todo y existe algum x tal que x é mãe de y. Sabemos que isso é verdadeiro quando x e y variam entre seres humanos. O segundo diz que existe uma pessoa que é mãe de todo mundo. Literalmente, existe algum x tal que para todo y, x é mãe de y. Isto não é verdadeiro quando y e x percorre o conjunto dos seres humanos.

6.1 Tradução do Português para a Lógica

Traduzir do Português para lógica, no caso da lógica proposicional, basta ligar as sentenças atômicas com os conectivos $\wedge$ (e), $\vee$ (ou), $\rightarrow$ (se ... então), $\neg$ (não) e $\leftrightarrow$ (se e somente se). O mesmo princípio se aplica quando temos quantificadores e variáveis, mas existem alguns problemas específicos que precisam ser considerados.

A seguir veremos algumas regras úteis para nos ajudar no processo de tradução.

Pronomes: Pronomes, palavras tais como "ele", ou "nada" por si próprios não se referem a qualquer coisa específica mas têm significado a partir do contexto. Já tivemos a oportunidade de ver como "algo" e "todo mundo" são traduzidos usando quantificadores. "Nada" é similar. "Nada é magro" torna-se $\neg\exists x.magro(x)$.

Palavras tais como "ele", "ela" são usadas especificamente como referência a alguém ou algo que já foi mencionado, portanto inevitavelmente correspondem a duas ou mais referências ao mesmo valor. Quando encontramos um pronome tal como este devemos proceder como se estivéssemos manuseando com o que ele se refere. Se ele é uma constante, então substituí-lo pela constante. Por exemplo, "José gosta de Maria e ela o adora" torna-se

$$gosta(José,Maria) \wedge gosta(Maria,José)$$

Isso é fácil, mas quando o pronome se refere a uma variável devemos primeiro estabelecer uma quantificação e assegurar que ele se aplica a ambos. Por exemplo, considere "algo está magro e ele está faminto". Não é correto traduzir separadamente as duas frases "algo está magro" e "ele está faminto". Não é correto porque o "algo" e "ele" são ligados pelo conectivo "e". Neste caso devemos estabelecer uma variável para lidar com esta ligação.

$$\exists x.(x \text{ está magro e } x \text{ está faminto})$$

e então lidar com o "e".

$$\exists x.(magro(x) \wedge faminto(x))$$

Podemos estabelecer a seguinte regra: se pronomes estão ligados por um conectivo, trate dos pronomes antes do conectivo.

6.2 Quantificadores e Tipos

Freqüentemente uma frase deve ser traduzida à linguagem lógica usando uma quantificação universal sobre um certo tipo de coisas em vez de acerca de todas as coisas. Assim, nesses casos, devemos qualificar o quantificador. No caso do quantificador universal isso é feito usando uma **implicação**. Por exemplo, "todos os seres racionais odeiam violência" tem como tradução

$$\forall \text{ (racional) } x.(x \text{ odeia violência})$$

onde racional diz-se um **qualificativo**. Isto se traduz para

$$\forall x.(racional(x) \rightarrow odeia\text{-}violência(x))$$

Se a quantificação é existencial, então é usada uma **conjunção** para ligar a parte principal com a parte qualificada. Por exemplo, "Maria gosta de alguém que gosta de lógica". Devemos escrever primeiro

$$\exists \text{ (uma pessoa que gosta de lógica)} y.gosta(Maria, y)$$

e então

$$\exists y.(pessoa(y) \wedge gosta(y, lógica) \wedge gosta(Maria, y))$$

Regra: Torne a estrutura da sentença correta antes de manipular o qualificativo.

Qualificativos podem ser traduzidos usando $\rightarrow$ ou $\wedge$ apropriadamente. Por isso introduziremos uma notação para facilitar os seus usos. Escreva $\forall x : nome - do - tipo.[\cdots]$ ou $\exists x : nome - do - tipo.[\cdots]$ e chamemo-lhes **quantificadores tipados**.

A notação é principalmente usada para qualificativos padrão, as vezes referidos como "**tipos**". Sentenças usando tipos podem sempre ser reescritas com a propriedade do

tipo explicitada. **Qualificativos padrão** incluem pessoas, números (inteiros, reais, etc.) caracteres, tempo, lista, conjuntos enumerados, etc. Por exemplo,

$$\forall x : tempo. \forall y : tempo. (x \geq y \rightarrow após(x, y))$$

seria uma simplificação para

$$\forall x. \forall y. (tempo(x) \wedge tempo(y) \wedge x \geq y \rightarrow após(x, y))$$

Tipos padrão são muito usados quando se escreve especificação de programas, os quais correspondem às várias estruturas de dados, tais como lista, num, etc., usados em programas.

Já vimos que a sentença $\forall x.\alpha(x)$ é verdadeira se e somente se **cada** sentença $\alpha[t/x]$ obtida de $\alpha(x)$ ao substituirmos cada ocorrência de x por um valor t, é verdadeira.

Por exemplo, "todos os programas que funcionam terminam", cuja tradução lógica é

$$\forall x. (programa(x) \rightarrow (funciona(x) \rightarrow termina(x)))$$

é verdadeira se cada sentença obtida substituindo um valor de x é verdadeira. Se o valor t substituído é um programa, então $programa(t)$ é verdadeiro e a sentença resultante

$$programa(t) \rightarrow (funciona(t) \rightarrow termina(t))$$

é verdadeira se $funciona(t) \rightarrow termina(t)$ é verdadeira. Se o valor t torna $programa(t)$ falsa (isto é, t não é um programa) então a sentença resultante também é verdadeira. Na prática, avaliamos o valor de uma sentença numa situação na qual os valores que substituem x são fixados de antemão. Por exemplo, eles poderiam ser {todos os programas escritos por nós}.

Quando são usados quantificadores qualificados eles sugerem a abrangência dos valores que devem substituir de modo que se possa testar a verdade da sentença.

A sentença "todos os programas que funcionam terminam" será escrita

$$\forall x : programas. (funciona(x) \rightarrow termina(x)).$$

Nesse caso os únicos valores que devemos considerar para x são aqueles que nomeiam programas. Esses são exatamente os valores que são úteis para mostrar que a sentença é verdadeira. Analogamente, a sentença "alguns programas que terminam funcionam", o que em lógica é

$$\exists x. (programa(x) \wedge funciona(x) \wedge termina(x))$$

é verdadeira se pelo menos uma das sentenças obtidas substituindo x por t for verdadeira. Não é o caso de tentar valores de t que tornem $programa(t)$ falsa pois eles não tornam a sentença verdadeira. Isto é sugerido pela versão do quantificador tipado

$$\exists x : programa.(funciona(x) \wedge termina(x))$$

Podem aparecer dificuldades como no exemplo: "Todo inteiro é menor que algum número natural", o qual em lógica se torna

$$\forall x : inteiro.\exists u : natural.x < u$$

Agora existe uma infinidade de sentenças para considerar, uma para cada inteiro. Como podemos checar todas? claro que não podemos checar todas individualmente. Em vez disso poderemos considerar diferentes casos.

Por exemplo, podemos ter dois casos. $x < 0$ ou $x \geq 0$. Assim, todos os inteiros negativos são considerados de uma vez, assim como todos os naturais. No primeiro caso a sentença é verdadeira tomando $u = 0$, pois 0 é um número natural e maior do que qualquer inteiro negativo. No segundo caso, $x + 1$ é um número natural que pode ser u. às vezes temos de usar uma prova para justificar a verdade de uma sentença, como veremos mais adiante.

6.3 Quantificadores como Conjunções e Disjunções Infinitas

Agora, dado $\mathbb{N}$, o conjunto dos números naturais, seja $P(x)$ a afirmação "o número natural x tem a propriedade P". Essa propriedade pode ser, por exemplo, "$x > 0$". Desse modo, as proposições $P(0)$, $P(1)$, $P(2)$ e $P(3)$ estabelecem que os números $0, 1, 2$ e 3 tem a propriedade descrita por P. No exemplo "$x > 0$", as proposições $P(1)$, $P(2)$ e $P(3)$ são verdadeiras, enquanto $P(0)$ é falsa. Elas tem a forma, respectivamente, $0 > 0$, $1 > 0$, $2 > 0$ e $3 > 0$. Se quisermos estabelecer numa única proposição que $1 > 0$, $2 > 0$ e $3 > 0$ tem a propriedade P, devemos combinar essas três proposições com a conjunção: $P(1)$ e $P(2)$ e $P(3)$. Usando esse procedimento, para estabelecer que "todo número natural tem a propriedade P", teríamos de escrever a conjunção infinita

$$P(0) \text{ e } P(1) \text{ e } P(2) \text{ e } P(3) \text{ e } \cdots$$

Como não é possível representar uma conjunção infinita temos de apelar para uma representação finita. A saída é usar a proposição "para todo x $P(x)$" ou simbolicamente "$\forall x.P(x)$". A frase "para todo x $\cdots$" é chamado **quantificador universal ligado à variável** x. Assim,

$$\forall x.P(x) \equiv \ P(0) \text{ e } P(1) \text{ e } P(2) \text{ e } P(3) \text{ e } \cdots$$

Um argumento análogo nos leva à conclusão que o quantificador existencial, isto é, a frase "existe um x $\cdots$" é uma disjunção infinita. De fato, suponha que quiséssemos expressar a proposição "existe um número natural menor do que 5 que tem a propriedade P". Um modo de fazer isso é escrever "$P(0)$ ou $P(1)$ ou $P(2)$ ou $P(3)$ ou $P(4)$". De modo similar se quisermos expressar que "existe um número natural com a propriedade P", teríamos de apelar para a disjunção infinita,

$$P(0) \text{ ou } P(1) \text{ ou } P(2) \text{ ou } P(3) \text{ ou } \cdots$$

Portanto,

$$\exists x.P(x) \equiv \ P(0) \text{ ou } P(1) \text{ ou } P(2) \text{ ou } P(3) \text{ ou } \cdots$$

Considere o Σ-domínio $\langle D, f_1, f_2, \cdots, f_n, R_1, \cdots, R_m\rangle$. Como age f_i sobre D? Vejamos o exemplo $\mathcal{N} = \langle \mathbb{N}, +, \leq\rangle$.

A função binária $+ : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ recebe o par $(3, 5)$, por exemplo, e retorna $8 \in \mathbb{N}$, $(3, 5) \mapsto 8$. Ou seja, se f_i é uma função n-ária em D, o efeito de f_i é associar n indivíduos de D a um indivíduo de D. Podemos, então, dizer que a ação de f_i sobre D não altera o ambiente "D", isto é , o resultado da ação de f_i sobre D está em D.

A relação binária "$\leq$" do domínio acima, associa $(3, 5)$, por exemplo, a "$3 \leq 5$". Entretanto, "$3 \leq 5$" é uma propriedade de indivíduos de D, e como tal é verdadeira ou falsa. Essa propriedade afirma que "3 é menor ou igual que 5", que acontece ser verdadeira. Assim, uma relação n-ária sobre D é uma função que associa n-indivíduos de D a um dos valores verdade, verdadeiro ou falso. Isto é, $R_i : D^n \longrightarrow \{0, 1\}$. Portanto, a ação de uma relação sobre D altera o ambiente de D, pois o resultado não é mais um individuo de D.

6.4 Linguagem de 1^a Ordem

A linguagem de 1^a ordem é mais complicada porque expressamos idéias mais complicadas. Queremos ser capazes de considerar a verdade ou falsidade de sentenças compostas construídas a partir de sentenças atômicas que podem, às vezes, ser verdadeiras e às vezes falsas sob uma única interpretação. Por exemplo, a sentença "se x é par então $x + 1$ é ímpar", ou traduzido para a lógica $\forall x.(par(x) \rightarrow impar(x + 1))$, é uma implicação das sentenças atômicas "$par(x)$" e "$impar(x + 1)$". Essa sentença é verdadeira no conjunto

dos inteiros, mas como já vimos anteriormente não podemos dizer que "x é par" é verdadeiro, ou que é falsa. Ela depende do valor de x. Portanto, estamos acrescentando à nossa linguagem uma habilidade para fazer asserções paramétricas. Essas asserções são atômicas, mas em vez de ser simplesmente verdadeiras ou falsas (como acontecia com proposições), elas estabelecem que alguma relação se dá entre seus argumentos. Essas relações são expressas com **predicados**. aplicados a argumentos.

Isso nos leva a considerar **teorias de** 1^a **ordem**. Teorias de 1^a ordem se distinguem de teorias de ordem mais baixas (tais como a teoria da lógica proposicional) no fato de que os argumentos de predicados são termos construídos de constantes, variáveis e aplicações de funções. Somente são permitidos quantificadores sobre variáveis. Em teoria de ordens mais altas, funções e predicados podem tomar funções e predicados como argumentos. Podemos também fazer afirmativas sobre todas as funções e predicados quantificando sobre símbolos de função e de predicados.

As linguagens de 1^a ordem ou linguagens de predicados são extensões das linguagens proposicionais. Na linguagem proposicional o alfabeto Σ não contém variáveis. Os únicos símbolos de funções que aparecem são os símbolos lógicos $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$. Por isso uma interpretação para essa linguagem é uma álgebra de Boole. Nas linguagens de 1^a ordem o alfabeto Σ além de conter variáveis, símbolos de função, e operadores lógicos, contém, também, **símbolos de predicados**. Por isso as Σ-estruturas correspondentes (Σ-domínios), podem ser estruturas com relações que não a igualdade, como por exemplo, relação de ordem. A lógica correspondente à linguagem de predicado, é uma extensão da lógica proposicional, e conhecida como **lógica de predicados** ou lógica de 1^a ordem.

Um típico **alfabeto de** 1^a **ordem**, Σ, tem a seguinte constituição:

$$\Sigma = X \cup \Sigma_C \cup \Sigma_F \cup \Sigma_R \cup \Sigma_L \cup \Sigma_P, \text{ onde}$$

1. $X = \{x, y, z, x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots\}$ é um conjunto contável de variáveis
2. $\Sigma_C = \{a, b, c, a_1, a_2, \ldots, b_1, b_2, \ldots, c_1, c_2, \ldots\}$ é um conjunto contável de símbolos constantes
3. $\Sigma_F = \{f_1, f_2, \ldots\}$ é um conjunto de símbolos de funções não lógicas
4. $\Sigma_R = \{R_1, R_2, \ldots\}$ é um conjunto de símbolos de relação ou predicado
5. $\Sigma_L = \{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \forall, \exists\}$ é o conjunto de símbolos lógicos
6. $\Sigma_P = \{(,), ., , \}$ é o conjunto de símbolos de pontuação.

Embora toda constante pode ser vista como um função sem argumentos, por questões didáticas decidimos separar os símbolos de função que representam funções com pelo menos um argumento dos que representam constantes.

Σ_L é fixo em todas as linguagens de 1^a ordem, o que varia e caracteriza a linguagem, como veremos nos exemplos mais adiante, é $\Sigma_d = \Sigma_C \cup \Sigma_F \cup \Sigma_R$.

O alfabeto Σ_d junto com uma função $arid : \Sigma_d \longrightarrow \mathbb{N}$ que indica a aridade dos símbolos de função (incluindo constantes que têm por definição aridade zero) e de predicados é uma assinatura relacional, pois consiste de símbolos funcionais e símbolos relacionais, cuja interpretação em vez de ser uma Σ-álgebra de Boole é um Σ_d-domínio.

Definiremos, agora, as linguagens de 1^a ordem. Mas, como na linguagem de predicados, existem duas entidades de natureza diferente (os termos e as fórmulas, propriamente ditas), a linguagem de primeira ordem ou de predicados para o alfabeto de primeira ordem será construída em duas fases. Na primeira construímos uma linguagem para os termos sobre os quais se vai predicar, chamamos esta linguagem de **linguagem dos termos de primeira ordem**. Depois usando estes termos como parte do alfabeto, construímos a **linguagem de primeira ordem** ou **linguagem de predicados**, onde seus termos são predicados sobre termos de primeira ordem. Assim, não devemos confundir os termos de primeira ordem com os termos da linguagem de primeira ordem. No primeiro caso estamos nos referindo aos termos que denotam objetos de algum universo (joão, cadeira, gato, 5, π, etc.). já no segundo estamos nos referindo a fórmulas bem formadas da linguagem, isto é, fórmulas que denotam alguma afirmação sobre objetos de algum universo ("João é alto", "todos os corvos são pretos", etc.)

Definição 6.4.1 *Seja $\Sigma = X \cup \Sigma_d \cup \Sigma_L \cup \Sigma_P$ um alfabeto de primeira ordem e $arid : \Sigma_d \longrightarrow \mathbb{N}$ uma função que indica a aridade dos símbolos de função e de predicado. A* **linguagem dos termos de primeira ordem** *é a linguagem gerada pela linguagem formal $\mathcal{L}^T = \langle X \cup \Sigma_C \cup \Sigma_F \cup \{(,),,\}, \mathcal{G}^T\rangle$ onde $\mathcal{G}^T = \{\mathsf{T_1}, \mathsf{T_2}, \mathsf{T_3}\}$ com:*

$\mathsf{T_1} : \dfrac{-}{x}$, $x \in X$

$\mathsf{T_2} : \dfrac{-}{a}$, $a \in \Sigma_C$

$\mathsf{T_3} : \dfrac{t_1, \ldots, t_n}{f(t_1, \ldots, t_n)}$ $arid(f) = n$.

A **linguagem de primeira ordem** *é gerada pela* **linguagem formal de primeira ordem** $\mathcal{L}^P = \langle \Sigma, \mathcal{G}\rangle$ *onde*

1. $\Sigma = Ling(\mathcal{L}^T) \cup \Sigma_R \cup \Sigma_L \cup \Sigma_P$
2. $\mathcal{G} = \{F_1, \ldots, F_8\}$ *onde*

   $\mathsf{F_1} : \dfrac{t_1, \ldots, t_n}{R(t_1, \ldots, t_n)}$ $arid(R) = n$ *e* $t_j \in Ling(\mathcal{L}^T)$ *para todo* $1 \leq j \leq n$

   $\mathsf{F_2} : \dfrac{\alpha}{\neg\alpha}$

   $\mathsf{F_3} : \dfrac{\alpha, \ \beta}{(\alpha \wedge \beta)}$

   $\mathsf{F_4} : \dfrac{\alpha, \ \beta}{(\alpha \vee \beta)}$

$$\mathsf{F}_5 : \frac{\alpha,\ \beta}{(\alpha \rightarrow \beta)}$$

$$\mathsf{F}_6 : \frac{\alpha,\ \beta}{(\alpha \leftrightarrow \beta)}$$

$$\mathsf{F}_7 : \frac{\alpha}{\exists x.\alpha} \quad x \in X$$

$$\mathsf{F}_8 : \frac{\alpha}{\forall x.\alpha} \quad x \in X$$

Observações:

1. $L^P = Ling(\mathcal{L}^P)$.
2. α e β são meta-variáveis, isto é podem ser substituídas, nas regras, por qualquer fórmula em L^P.
3. As fórmulas geradas por F_1 são chamadas de **fórmulas atômicas**, pois elas não contém nenhum operador lógico.
4. O conjunto L^P é chamado de conjunto das fórmulas bem formadas (fbf), ou simplesmente fórmulas de $\mathcal{L}^P$.
5. De aqui em diante chamaremos de linguagem formal de 1^a ordem às linguagens formais que especificam uma linguagem de 1^a ordem. Se não houver confusão às vezes omitiremos o termo "formal" em linguagens formais de 1^a ordem.

Embora tendo colocado "$\exists$" e "$\forall$" entre os operadores lógicos, não é esse exatamente o caso. Eles constituem exceções em relação a esses operadores. De certo modo, como já vimos na introdução deste capítulo, esses dois operadores caracterizam a diferença entre linguagem proposicional e uma linguagem de predicados. No que segue analisaremos, com um pouco mais de profundidade, esses dois quantificadores.

Consideremos, sobre $\mathbb{N}$, a seguinte relação ternária $R(x, y, z) = \{(x, y, z) \in \mathbb{N}^3 / \ x+y = z\}$, a qual indicamos por "$x + y = z$". Nesta relação x, y, e z são "variáveis livres". Isto é, não estão ligados por nenhum quantificador. Podemos dizer, também, que elas são livres de assumir quaisquer valores naturais, daí o nome relação ternária. Apliquemos o "operador" "$\exists y$" à relação "$x + y = z$". O resultado será a relação $R_1 \equiv \exists y.(x + y = z)$. Nesta relação y não é mais livre, ele está ligado ao operador $\exists y$. As variáveis x e z continuam livres em R_1. é fácil ver que R_1 é a relação $x \leq z$ sobre $\mathbb{N}$. Em outras palavras, o efeito do operador $\exists y$ foi transformar a relação ternária $R \equiv$ "$x + y = z''$ na relação binária $R_1 \equiv$ "$x \leq z''$.

Apliquemos, agora, o operador "$\forall z$", sobre a relação R_1. O resultado é a relação unária $R_2 \equiv \forall z.\exists y.(x + y = z)$, ou simplesmente, $R_2 \equiv \forall z.(x \leq z)$. Aqui z não é mais livre, mas ligada ao quantificador $\forall$. x continua livre. A relação $R_2 \equiv \forall z.x \leq z$ é um subconjunto de $\mathbb{N}$, mais precisamente o subconjunto $\{0\}$, que por definição é uma relação unária sobre

$\mathbb{N}$. Finalmente, apliquemos o operador "$\exists x$" à relação R_2. O resultado $\exists x.\forall z.(x \leq z)$ é uma "propriedade" dos elementos de $\mathbb{N}$, qual seja "existe um natural menor que todos os outros", que acontece ser verdadeira. Observe que $\exists x.\forall z.(x \leq z)$ não possui mais variáveis livres, ela é uma asserção ou proposição sobre números naturais e portanto é verdadeira ou falsa.

Podemos tirar várias conclusões relativas à análise das quantificações aplicadas sobre as relações acima. Primeiro, uma relação sem variáveis livres é uma proposição sobre os indivíduos do universo de discurso. No caso acima o universo de discurso é o conjunto dos números naturais. Segundo, os quantificadores "$\forall$" e "$\exists$" quando quantificados por variáveis são transformadores de relações. Essas transformações são efetuadas do seguinte modo. Se $R(x_1, \cdots, x_n)$ é uma relação n-ária, ou seja, uma relação com n variáveis livres, a quantificação "$\forall x_i$" ou "$\exists x_i$" tem como efeito transformar a relação n-ária $R(x_1, \cdots, x_n)$ na relação $(n-1)$-ária $R_1(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$, onde agora x_i não é mais livre, mas ligada ao quantificador correspondente. No caso particular de $R(x)$ ser uma relação unária o operador "$\exists x$" ou "$\forall x$" transforma $R(x)$ na proposição "$\exists x.R(x)$" ou "$\forall x.R(x)$", cuja denotação é verdadeiro ou falsa.

Observe que se x não é livre em R, os operadores $\forall x$ e $\exists x$ não produzem nenhum efeito sobre R. Em outras palavras, os operadores "$\forall$" e "$\exists$", não atuam sobre proposições, mas sobre fórmulas abertas (fórmulas com alguma variável livre, a qual será chamada, simplesmente, de fórmula). Portanto, uma proposição pode ser vista como uma fórmula sem variáveis livres, conhecida, também, como uma **fórmula fechada**. Uma sentença ou proposição pode ser definida como uma fórmulas sem variáveis livres. Se as variáveis $x_1, \ldots, x_n$ são livres na fórmula α indicaremos isso por $\alpha(x_1, \cdots, x_n)$.

Embora de já termos apelado para a intuição para entender o conceito de variável livre e ligada, devido à importância desses conceitos daremos uma definição precisa. Porém, primeiro estudaremos isoladamente cada ocorrência de uma variável numa fórmula, e determinaremos se essa ocorrência em particular é livre ou ligada.

Definição 6.4.2 *Uma* **ocorrência** *de uma variável x numa fbf é* **ligada** *se a ocorrência de x é a variável de uma quantificação ou se está dentro do escopo de uma quantificação cuja variável que está sendo quantificada é x. De outro modo a* **ocorrência** *da variável é dita* **livre** *na fbf.*

Cada ocorrência de uma variável ou é livre ou ligada, nunca ambas.

Exemplo 6.4.3 *A fórmula $\forall x.(P(x, f(a, y)) \wedge \exists y.P(y, x)) \vee \neg\exists y.P(x, y)$ tem quatro ocorrências da variável x e cinco da variável y. A seguir veremos quais dessas ocorrências são livres e quais ligadas:*

$$\forall \underbrace{x}_{1}.(P(\underbrace{x}_{2}, f(a, \underbrace{y}_{3})) \wedge \exists \underbrace{y}_{4}.P(\underbrace{y}_{5}, \underbrace{x}_{6})) \vee \neg\exists \underbrace{y}_{7}.P(\underbrace{x}_{8}, \underbrace{y}_{9})$$

1. *esta ocorrência da variável x é ligada, pois é a variável de uma quantificação.*
2. *esta ocorrência da variável x é ligada, pois esta dentro do escopo de uma quantificação 1. cuja variável que está sendo quantificada é x.*
3. *esta ocorrência da variável y é livre, pois não é a variável de uma quantificação e embora esteja dentro do escopo de uma quantificação 1., a variável que está sendo quantificada não é y.*
4. *esta ocorrência da variável y é ligada, pois é a variável de uma quantificação.*
5. *esta ocorrência da variável y é ligada, pois esta dentro do escopo de duas quantificações (1. e 4.) e, no caso da quantificação 2., a variável que está sendo quantificada é y.*
6. *esta ocorrência da variável x é ligada, pois esta dentro do escopo de duas quantificações (1. e 4.) e, no caso da quantificação 1., a variável que está sendo quantificada é x.*
7. *esta ocorrência da variável y é ligada, pois é a variável de uma quantificação.*
8. *esta ocorrência da variável x é livre, pois não é a variável de uma quantificação e embora esteja dentro do escopo de uma quantificação 7., a variável que está sendo quantificada não é x.*
9. *esta ocorrência da variável y é ligada, pois esta dentro do escopo de uma quantificação 7. cuja variável que está sendo quantificada é y.*

Definição 6.4.4 *x é uma* **variável livre** *numa fórmula α se existe em α uma ocorrência livre de x. Analogamente, x é uma* **variável ligada** *numa fórmula α se existe em α uma ocorrência ligada da variável x.*

No caso do exemplo anterior, tanto x como y têm ocorrências livres e ligadas na fórmula $\forall x.(P(x, f(a, y)) \wedge \exists y.P(y, x)) \vee \neg\exists y.P(x, y)$, e portanto dizemos que essas variáveis são livres e ligadas ao mesmo tempo nessa fórmula. Assim, numa fórmula α, qualquer, uma variável pode ser livre e ligada ao mesmo tempo. Observe que o conceito de variável ligada é um conceito relativo a uma quantificação, pois uma variável x é ligada a uma fórmula α se, e somente se, na fórmula α existe uma quantificação $\forall x$ ou $\exists x$.

Se x é uma variável livre numa fórmula α, o nome x nas ocorrências livres da variável não é importante. Em outras palavras, podemos trocá-la na fórmula por uma nova variável, sem alterar o significado da fórmula. Por exemplo, seja a fórmula $\alpha \stackrel{\text{def}}{=} \exists y.x+1 = y \wedge \forall x.\exists y.x < y$. Trocando as ocorrências livres da variável x por z em α, indicado por $\alpha[z/x]$, a fórmula resultante $\alpha[z/x] \stackrel{\text{def}}{=} \exists y.z + 1 = y \wedge \forall x.\exists y.x < y$ não alterou seu significado. Observe que x não poderia ser trocado por y em α, sem alterar seu significado pois a expressão $\alpha[y/x] \stackrel{\text{def}}{=} \exists y.y+1 = y \wedge \forall x.\exists y.x < y$ é falsa no domínio $\mathcal{N} = \langle \mathbb{N}, +, <, = \rangle$ dos

números naturais, enquanto na expressão original, por ser uma fórmula não fechada (com variáveis livres), o seu valor verdade depende de uma interpretação (é contingente). Ou seja, $\alpha[y/x]$ é fechada, e portanto, só pode tomar um valor verdade numa interpretação. Isto ocorreu porque quando substituímos as ocorrências livres de x por y, essas novas ocorrências da variável y tornaram-se ligada pela quantificação $\exists y$. Neste caso dizemos que x não é substituível por y ou o termo $t \stackrel{\text{def}}{=} y$ não é livre para x. Podemos generalizar esta idéia e trocar todas as ocorrências livres de uma variável por algum termo, o qual quando substituído não ligue nenhuma das variáveis que compõem o termo.

Definição 6.4.5 *Seja α uma fbf. Um* ***termo*** *t é dito* **livre para** *x* **em** *α se nenhuma das ocorrências livres de x em α está dentro do escopo de uma quantificação $\forall y$ ou $\exists y$ onde y é uma variável em t. A substituição de todas as ocorrências livres de x pelo termo t em α, é denotado por $\alpha[t/x]$.*

Exemplo 6.4.6

1. *O termo y é livre para x em $R(x)$, mas não é livre para x em $\forall y.R(x)$.*

2. *O termo x é livre para y em $R(z)$ e em $\forall y.R(y)$.*

3. *$f(g(a,y))$ é livre para x, y, z em $\forall x.R(f(x), y, z) \rightarrow \neg R(z, x, f(y))$, mas ele é livre para y, mas não para x e z em $\forall z.R(x, y, z) \wedge \forall y.S(f(x), z, y) \wedge P(z)$.*

4. *Qualquer termo t que não contém variáveis é livre para qualquer variável em qualquer fórmula. Isto é, $g(f(a), c)$ é livre para qualquer variável em qualquer fórmula.*

5. *Um termo t é livre para qualquer variável em α se nenhuma das variáveis em t é ligada em α. Por exemplo, $h(a, f(g(b, y)))$ é livre para qualquer variável nas fórmulas*

   (a) $R(x, y) \rightarrow \forall x.S(x)$

   (b) $\forall x.R(f(x), y, z) \rightarrow \neg R(z, x, f(y))$

   (c) $\forall z.R(x, y, z) \wedge \forall x.S(f(x), z, y) \wedge P(z))$.

6. *Qualquer variável é livre para ela própria em qualquer fórmula.*

7. *Se a variável x não é livre em α, então qualquer termo t é livre para x em α. Por exemplo, $f(x, y)$ é livre para z em $\forall x.P(x, y)$. Observe que se x não ocorre em α, então ela não é livre em α. Logo, se x não ocorre em α, então qualquer termo t é livre para x em α.*

6.5 Verdade

Na linguagem proposicional toda proposição era composta de proposições atômicas, usando os conectivos lógicos $\neg$, $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$. Para fornecer uma interpretação para uma proposição, tudo que precisávamos saber era se as proposições atômicas eram verdadeiras ou falsas. Podíamos listar todas as possíveis interpretações usando a tabela verdade. Agora a situação é mais complexa. Podemos expressar relações que são verdadeiras para certos indivíduos e não para outros, em vez de simplesmente estabelecer que é verdadeira ou falsa. Para determinar a veracidade ou falsidade de uma fórmula em uma interpretação precisamos saber que valores as variáveis podem tomar, que elementos as constantes interpretam, e que funções e relações irão representar os símbolos de função e de relação, respectivamente.

Definição 6.5.1 *Sejam $\Sigma = X \cup \Sigma_d \cup \Sigma_L \cup \Sigma_P$ um alfabeto de 1^a ordem e L^P a linguagem de 1^a ordem associada a este alfabeto. Uma* **interpretação** *para L^P é um Σ_d-domínio.*

Ou seja, uma interpretação para L^P é uma estrutura matemática, onde as variáveis e constantes de Σ tomam valores e onde são interpretados os símbolos de funções e relações.

Exemplo 6.5.2 *Considere o alfabeto de 1^a ordem*

$$\Sigma = \{x, y, \cdots, f, R_1, R_2, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \exists, \forall, (,)\},$$

onde $arid(f) = 1$, $arid(R_1) = 2$ e $arid(R_2) = 1$. Observe que $\Sigma_d = \{f, R_1, R_2\}$. Seja L^P a linguagem de 1^a ordem adjacente a este alfabeto e $\mathbb{Z}$ o conjunto dos números inteiros. O Σ_d-domínio $\langle \mathbb{Z}, f_{\mathbb{Z}}, R_{1_{\mathbb{Z}}}, R_{2_{\mathbb{Z}}} \rangle$, onde $f_{\mathbb{Z}} : \mathbb{Z} \longrightarrow \mathbb{Z}$ tal que $f_{\mathbb{Z}}(x) = x + 1$, $R_{1_{\mathbb{Z}}}(x, y) \equiv$ "$x = y$" e $R_{2_{\mathbb{Z}}}(x) \equiv$ "$x > 1$" é uma interpretação de L^P. Uma outra interpretação pode ser dada pelo Σ_d-domínio $\langle \mathbb{Z}, f^{\mathbb{Z}}, R_1^{\mathbb{Z}}, R_2^{\mathbb{Z}} \rangle$, onde $f^{\mathbb{Z}} : \mathbb{Z} \longrightarrow \mathbb{Z}$ tal que $f^{\mathbb{Z}}(x) = x \cdot x$, $R_1^{\mathbb{Z}}(x, y) \equiv$ "$x \cdot y = 0$" e $R_2^{\mathbb{Z}}(x) \equiv$ "$x = 0$".

Seja L^P uma linguagem de 1^a ordem, $\mathcal{D} = \langle D, \Sigma_{d_D} \rangle$ uma interpretação para L^P e $\rho : X \longrightarrow D$ uma atribuição de valores para as variáveis na interpretação $\mathcal{D}$. Analogamente, como na definição da linguagem de predicado, para determinar o valor verdade que toma qualquer fórmula α em $Ling(L^P)$ quando interpretada em $\mathcal{D}$ e as variáveis tomando os valores atribuídos por ρ, devemos fazê-lo por parte. Assim, primeiro determinamos os valores que tomam os termos de primeira ordem e depois interpretamos as fórmulas. O valor que tomam os termos de primeira ordem estão em estreita relação com os valores atribuídos às variáveis, e portanto, cada atribuição $\rho : X \longrightarrow D$ determina uma função de valoração $V_\rho : Ling(\mathcal{L}_T) \longrightarrow D$, a qual é definida recursivamente como segue

$V_\rho(x) = \rho(x)$ para cada $x \in X$

$V_\rho(a) = a_D$ para cada $a \in \Sigma_C$

$V_\rho(f(t_1, \ldots, t_n)) = f_D(V_\rho(t_1), \ldots, V_\rho(t_n))$ para cada $f \in \Sigma_F$ tal que $arid(f) = n$.

Exemplo 6.5.3 *Seja a linguagem de 1^a ordem do exemplo 6.5.2 e a primeira interpretação dada nesse mesmo exemplo. Uma valoração para os termos $f(x)$ e $f(f(y))$, na atribuição $\rho : X \longrightarrow \mathbb{Z}$ tal que $\rho(x) = 5$ e $\rho(y) = 7$ é*

$$\begin{aligned} V_\rho(f(x)) &= f_{\mathbb{Z}}(V_\rho(x)) \\ &= f_{\mathbb{Z}}(\rho(x)) \\ &= f_{\mathbb{Z}}(5) \\ &= 5+1 \\ &= 6 \end{aligned}$$

$$\begin{aligned} V_\rho(f(f(y))) &= f_{\mathbb{Z}}(V_\rho(f(y))) \\ &= f_{\mathbb{Z}}(f_{\mathbb{Z}}(V_\rho(y))) \\ &= f_{\mathbb{Z}}(f_{\mathbb{Z}}(\rho(y))) \\ &= f_{\mathbb{Z}}(f_{\mathbb{Z}}(7)) \\ &= f_{\mathbb{Z}}(7+1) \\ &= f_{\mathbb{Z}}(8) \\ &= 8+1 \\ &= 9 \end{aligned}$$

Agora podemos extender esta valoração para fórmulas qualquer na linguagem.

Definição 6.5.4 *Seja $\Sigma = X \cup \Sigma_d \cup \Sigma_L \cup \Sigma_P$ um alfabeto de 1^a ordem, $arid : \Sigma_d \longrightarrow \mathbb{N}$ uma função de aridade e L^P a linguagem de 1^a ordem adjacente a este alfabeto. Uma valoração verdade das fórmulas em L^P numa interpretação $\mathcal{D}$, com respeito a uma atribuição $\rho : X \longrightarrow D$, é uma função $\mathcal{V}_\rho : L^P \longrightarrow \{0,1\}$ definida por*

$\mathcal{V}_\rho(P(t_1,\ldots,t_n)) = P_D(V_\rho(t_1),\ldots,V_\rho(t_n))$ *para cada $P \in \Sigma_R$ com $arid(P) = n$ e $t_1,\ldots,t_n \in Ling(\mathcal{L}^T)$.*

$\mathcal{V}_\rho(\neg\alpha) = \sim \mathcal{V}_\rho(\alpha)$

$\mathcal{V}_\rho(\alpha \wedge \beta) = \mathcal{V}_\rho(\alpha) \cdot \mathcal{V}_\rho(\beta)$

$\mathcal{V}_\rho(\alpha \wedge \beta) = \mathcal{V}_\rho(\alpha) \cdot \mathcal{V}_\rho(\beta)$

$\mathcal{V}_\rho(\alpha \vee \beta) = \mathcal{V}_\rho(\alpha) + \mathcal{V}_\rho(\beta)$

$\mathcal{V}_\rho(\alpha \rightarrow \beta) = \mathcal{V}_\rho(\alpha) \Rightarrow \mathcal{V}_\rho(\beta)$

$\mathcal{V}_\rho(\alpha \leftrightarrow \beta) = \mathcal{V}_\rho(\alpha) \Leftrightarrow \mathcal{V}_\rho(\beta)$

$$\mathcal{V}_\rho(\forall x.\alpha) = \begin{cases} 1 & \text{, se para toda atribuição } \rho' : X \longrightarrow D \text{ tal que} \\ & \rho'(y) = \rho(y) \text{ sempre que } y \neq x,\ \mathcal{V}_{\rho'}(\alpha) = 1 \\ 0 & \text{, senão} \end{cases}$$

$$\mathcal{V}_\rho(\exists x.\alpha) = \begin{cases} 1 & \text{, se para alguma atribuição } \rho' : X \longrightarrow D \text{ tal que} \\ & \rho'(y) = \rho(y) \text{ sempre que } y \neq x,\ \mathcal{V}_{\rho'}(\alpha) = 1 \\ 0 & \text{, senão} \end{cases}$$

onde as operações $\sim$, $+$, $\cdot$, $\Rightarrow$ e $\Leftrightarrow$ são as mesmas definidas para a álgebra de Boole $\mathcal{B}$.

Exemplo 6.5.5 *Considere as seguintes fórmulas na linguagem de primeira ordem:*

1. $\alpha_1 \stackrel{\text{def}}{=} P(x,y) \rightarrow R(x,c)$
2. $\alpha_2 \stackrel{\text{def}}{=} \forall x.(R(x,c) \rightarrow R(f(x),c))$
3. $\alpha_3) \stackrel{\text{def}}{=} \forall x.\exists y.(P(x,y) \rightarrow \exists y.\forall x.(P(x,y)))$

Uma interpretação para estas fórmulas pode ser a seguinte:

- *Conjunto base:* $\mathbb{N}$ *o conjunto dos números naturais*
- *Interpretação de* f*:* $f_{\mathbb{N}} : \mathbb{N} \longrightarrow \mathbb{N}$ *definida por* $f_{\mathbb{N}}(n) = 2n$.
- *Interpretação de* c*:* $c_{\mathbb{N}} : \mathbb{N}^0 \longrightarrow \mathbb{N}$*, ou seja* $c_{\mathbb{N}}$ *é uma constante em* $\mathbb{N}$*, definida por* $c_{\mathbb{N}} = 2$.
- *Interpretação de* P*:* $P_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$ *definido por* $P_{\mathbb{N}} = \{(n,m) \in \mathbb{N} \times \mathbb{N} \ / \ n$ *e* m *têm a mesma paridade*$\}$*. Observe que* $P_{\mathbb{N}}(n,m)$ *denota que* $(n,m) \in P_{\mathbb{N}}$.
- *Interpretação de* R*:* $R_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$ *definido por* $R_{\mathbb{N}} = \{(n,m) \in \mathbb{N} \times \mathbb{N} \ / \ n \geq m\}$*. Observe que* $R_{\mathbb{N}}(n,m)$ *denota que* $(n,m) \in R_{\mathbb{N}}$.

Uma atribuição de valores para as variáveis da linguagem de primeira ordem nesta interpretação é uma função $\rho : X \longrightarrow \mathbb{N}$ *tal que* $\rho(x) = 5$ *e* $\rho(y) = 7$.

Nesta interpretação e para esta atribuição de valores verdade temos que:

α_1 :

$$\begin{aligned}
\mathcal{V}_\rho(P(x,y) \rightarrow R(x,c)) &= \mathcal{V}_\rho(P(x,y)) \Rightarrow \mathcal{V}_\rho(R(x,c)) \\
&= P_{\mathbb{N}}(V_\rho(x), V_\rho(y)) \Rightarrow R_{\mathbb{N}}(V_\rho(x), V_\rho(c)) \\
&= P_{\mathbb{N}}(\rho(x), \rho(y)) \Rightarrow R_{\mathbb{N}}(\rho(x), c_{\mathbb{N}}) \\
&= P_{\mathbb{N}}(5,7) \Rightarrow R_{\mathbb{N}}(5,2) \\
&= 1 \Rightarrow 1 \\
&= 1
\end{aligned}$$

α_2 :

$$\begin{aligned}
\mathcal{V}_\rho(\forall x.(R(x,c) \rightarrow R(f(x),c))) &= \mathcal{V}_{\rho'}(R(x,c) \rightarrow R(f(x),c)) \\
&= \mathcal{V}_{\rho'}(R(x,c)) \Rightarrow \mathcal{V}_{\rho'}(R(f(x),c)) \\
&= R_{\mathbb{N}}(\mathcal{V}_{\rho'}(x), \mathcal{V}_{\rho'}(c)) \Rightarrow R_{\mathbb{N}}(\mathcal{V}_{\rho'}(f(x)), \mathcal{V}_{\rho'}(c)) \\
&= R_{\mathbb{N}}(\rho'(x), c_{\mathbb{N}}) \Rightarrow R_{\mathbb{N}}(f_{\mathbb{N}}(\mathcal{V}_{\rho'}(x)), c_{\mathbb{N}}) \\
&= R_{\mathbb{N}}(\rho'(x), 2) \Rightarrow R_{\mathbb{N}}(f_{\mathbb{N}}(\rho'(x)), 2) \\
&= R_{\mathbb{N}}(\rho'(x), 2) \Rightarrow R_{\mathbb{N}}(2\rho'(x), 2)
\end{aligned}$$

Quando $\rho'(x) < 2$ *então* $R_{\mathbb{N}}(\rho'(x), 2)$ *será falso, e portanto* $R_{\mathbb{N}}(\rho'(x), 2) \Rightarrow R_{\mathbb{N}}(2\rho'(x), 2)$ *será verdadeiro. Se* $\rho'(x) \geq 2$ *então* $2\rho'(x) \geq 2$*, e portanto* $R_{\mathbb{N}}(\rho'(x), 2)$ *e* $R_{\mathbb{N}}(2\rho'(x), 2)$ *serão verdadeiros. Logo,* $R_{\mathbb{N}}(\rho'(x), 2) \Rightarrow R_{\mathbb{N}}(2\rho'(x), 2)$ *será verdadeiro.*

Assim, como para $\mathcal{V}_\rho(\forall x.(R(x,c) \to R(f(x),c)))$ *ser verdadeiro,* $\mathcal{V}_{\rho'}(R(x,c) \to R(f(x),c))$ *deve ser verdadeiro para toda atribuição* $\rho' : X \longrightarrow \mathbb{N}$ *que se diferencie de* ρ*, eventualmente, no valor atribuído a* x*, podemos concluir que* $\mathcal{V}_\rho(\forall x.(R(x,c) \to R(f(x),c)))$ *é verdadeiro.*

α_3 :

O uso de quantificadores torna mais difícil a demonstração de que uma fórmula é verdadeira ou falsa, pois devemos considerar uma quantidade infinita (pois o conjunto base é infinito) de funções de valoração. Como α_3 *tem quatro quantificadores, verificar que ela é verdadeira, será difícil usando o método anterior. Assim, neste caso, apelaremos à intuição e conhecimento matemáticos sobre o conjunto dos números naturais.*

Primeiro, "traduziremos" a fórmula α_3 *nesta interpretação:*

$$\forall x \in \mathbb{N}.\exists y \in \mathbb{N}.(paridade(x) = paridade(y) \to \exists y \in \mathbb{N}.\forall x \in \mathbb{N}.paridade(x) = paridade(y))$$

o qual informalmente seria "para todo número natural existe um número natural tal que, se eles têm a mesma paridade então existe um número natural tal que para todo numero natural eles têm a mesma paridade". o qual é verdadeiro, pois uma fórmula do tipo $\forall x.\exists y(\alpha \to \beta)$ *é verdadeira quando* $\forall x.\exists y.\alpha$ *for falsa. Neste caso em particular temos que, para cada* x *existe um* y *que não têm a mesma paridade. Por exemplo, para* $x = 3$ *existe* $y = 4$*, tal que "se 3 e 4 têm a mesma paridade então ....", como 3 e 4 têm paridade diferente, então esta afirmação é verdadeira. Claramente para qualquer* x *podemos encontrar um tal* y*, que faça que a sentença "se* x *e* y *têm a mesma paridade então ..." seja verdadeira.*

Definição 6.5.6 *Uma fórmula* α *é* ***verdadeira numa interpretação*** $\mathcal{D}$ ***e atribuição para as variáveis*** *(livres)* ρ*, denotado por* $\mathcal{D} \models_\rho \alpha$*, se* $\mathcal{V}_\rho(\alpha) = 1$*. Neste caso o par* $\langle \mathcal{D}, \rho \rangle$ *é dito um* ***modelo*** *de* α*. Uma fórmula* α *é* **satisfatível** *se tem algum modelo. Se* α *não tiver qualquer modelo dizemos que a fórmula* α *é* **insatisfatível**. *Uma fórmula é* ***falsa numa interpretação*** $\mathcal{D}$ ***e atribuição*** ρ*, denotado por* $\mathcal{D} \not\models_\rho \alpha$*, se* $\mathcal{V}_\rho(\alpha) = 0$

Note que o fato de uma fórmula ser falsa numa interpretação e atribuição não significa que ela seja insatisfatível, já se for insatisfatível ela será falsa em qualquer interpretação e atribuição. Note ainda que se uma fórmula α é satisfatível e t é um termo livre para x em α então $\alpha[t/x]$ também é satisfatível. Analogamente, se uma fórmula α é insatisfatível e t é um termo livre para x em α então $\alpha[t/x]$ também é insatisfatível.

A seguinte proposição permite verificar se uma determinada interpretação satisfaz uma fórmula para uma atribuição de modo recursivo, porém sem apelar diretamente à função $\mathcal{V}_\rho$.

Proposição 6.5.7 *Sejam* $\alpha \in L^P$*,* $\mathcal{D} = \langle D, \Sigma_D \rangle$ *uma interpretação para* α *e* $\rho :\longrightarrow D$ *uma atribuição.*

1. *Se* $\alpha = R(t_1, \ldots, t_n)$ *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se* $R_D(V_\rho(t_1), \ldots, V_\rho(t_n))$ *é uma proposição verdadeira, onde* R_D *é a interpretação do símbolo de predicado* R *em* $\mathcal{D}$.
2. *Se* $\alpha = \neg\beta$ *para algum* $\beta \in L^P$, *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se não é o caso* $\mathcal{D} \models_\rho \beta$.
3. *Se* $\alpha = \beta \wedge \gamma$ *para algum* $\beta, \gamma \in L^P$, *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se* $\mathcal{D} \models_\rho \beta$ *e* $\mathcal{D} \models_\rho \gamma$.
4. *Se* $\alpha = \beta \vee \gamma$ *para algum* $\beta, \gamma \in L^P$, *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se* $\mathcal{D} \models_\rho \beta$ *ou* $\mathcal{D} \models_\rho \gamma$.
5. *Se* $\alpha = \beta \rightarrow \gamma$ *para algum* $\beta, \gamma \in L^P$, *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se* $\mathcal{D} \models_\rho \neg\beta$ *ou* $\mathcal{D} \models_\rho \gamma$.
6. *Se* $\alpha = \beta \leftrightarrow \gamma$ *para algum* $\beta, \gamma \in L^P$, *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se* $\mathcal{D} \models_\rho \beta \wedge \gamma$ *ou* $\mathcal{D} \models_\rho \neg\beta \wedge \neg\gamma$.
7. *Se* $\alpha = \forall x.\beta$ *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se para qualquer atribuição* ρ' *tal que* $\rho'(y) = \rho(y)$ *quando* $y \neq x$, $\mathcal{D} \models_{\rho'} \beta$.
8. *Se* $\alpha = \exists x.\beta$ *então* $\mathcal{D} \models_\rho \alpha$ *se e somente se para alguma atribuição* ρ' *tal que* $\rho'(y) = \rho(y)$ *quando* $y \neq x$, $\mathcal{D} \models_{\rho'} \beta$.

Demonstração: Direto da definição de $\mathcal{D} \models_\rho \alpha$. ■

Note que, ítens 7 e 8 nada foi exigido de ρ' quando aplicado a x, e portanto $\rho'(x) = \rho(x)$ ou não, mas se $y \neq x$ então necessariamente $\rho'(y) = \rho(y)$. Assim, ρ' é uma atribuição que difere de ρ no máximo no valor atribuído a x.

Os seguintes exemplos simples esclarecerão melhor as diferenças entre cada uma dos conceitos introduzidos na definição 6.5.6.

Exemplo 6.5.8 *Seja a seguinte interpretação* $\mathcal{Z}$ *ou* Σ*-domínio:*

- *Conjunto base: o conjunto dos números inteiros* $\mathbb{Z}$
- *Interpretação do símbolo de função (constante) c:* $c_\mathbb{Z} = 1$
- *Interpretação do símbolo de predicado P:* $P_\mathbb{Z} \subseteq \mathbb{Z} \times \mathbb{Z}$ *definido por* $P_\mathbb{Z} = \{(x, y) \ / \ x \geq y\}$. *Assim,* $P_\mathbb{Z}(n, m)$ *se, e somente se,* $n \geq m$.

A fórmula $P(x, c)$ *é verdadeira na interpretação* $\mathcal{Z}$ *e atribuição* $\rho(x) = 2$, *isto é* $\mathcal{Z} \models_\rho P(x, c)$. *Assim, o par* $\langle \mathcal{Z}, \rho \rangle$ *seria um modelo da fórmula* $P(x, c)$ *e por tanto* $P(x, c)$ *é satisfatível. De fato toda fórmula atômica é satisfatível. Porém a mesma fórmula é falsa para a mesma interpretação e atribuição* $\rho(x) = 0$.

Exemplo 6.5.9 *A fórmula* $P(x, y) \wedge \neg P(x, y)$ *é insatisfatível, pois para qualquer interpretação e valor que tomem* x *e* y *nessa interpretação, nunca ocorrera que esses valores satisfaçam e não satisfaçam ao mesmo tempo a mesma propriedade.*

Definição 6.5.10 *Uma fórmula α é* **verdadeira numa dada interpretação** $\mathcal{D}$*, denotado por $\mathcal{D} \models \alpha$, se para qualquer atribuição ρ, $\mathcal{D} \models_{\rho} \alpha$. Uma fórmula α é* **falsa numa dada interpretação** $\mathcal{D}$*, se para qualquer atribuição ρ, não é o caso que $\mathcal{D} \models_{\rho} \alpha$. Uma fórmula α é* ***contingente numa dada interpretação*** *se existem atribuições ρ e ρ' tais que $\mathcal{D} \models_{\rho} \alpha$ e não é o caso de que $\mathcal{D} \models_{\rho'} \alpha$. Uma fórmula é* **universalmente válida** *se ela é verdadeira para toda interpretação. Analogamente, uma fórmula é uma* ***contradição*** *se ela é falsa para toda interpretação.*

Note que a definição de contradição coincide com a de insatisfatível, ou seja uma fórmula é insatisfatível se e somente se ela é uma contradição.

Exemplo 6.5.11 *Seja a fórmula $\forall x.(P(x,y) \rightarrow \exists y.\neg P(y,x))$. Então:*

1. *α é verdadeira na interpretação $\mathcal{N}_1 = \langle \mathbb{N}, D \rangle$ onde $D(n,m)$ se e somente se $n \neq n$,*
2. *α é falsa na interpretação $\mathcal{N}_2 = \langle \mathbb{N}, \leq \rangle$, e*
3. *α é contingente na interpretação $\mathcal{N}_3 = \langle \mathbb{N}, < \rangle$.*

No caso de $\mathcal{N}_1$, α ela é verdadeira, pois para todo x natural e qualquer valor que atribua a y, $x = x$, e por tanto a premissa da implicação $P(x,y) \rightarrow \exists y.\neg P(y,x)$ sempre sera falsa. Logo a implicação como um todo é verdadeira (Lembre que $0 \Rightarrow 0 = 0 \Rightarrow 1 = 1$).

No caso de $\mathcal{N}_2$, se ρ é uma atribuição qualquer então quando x na quantificação toma o valor 0, trivialmente a premissa da implicação $P(x,y) \rightarrow \exists y.\neg P(y,x)$ sempre será verdadeira, porém a conclusão da implicação será falsa (sempre existe um valor que é maior ou igual que 0). Logo a implicação seria falsa para um valor de x e por tanto a fórmula como um todo seria falsa (lembre que para uma fórmula $\forall x.\alpha$ se falsa é suficiente que α seja falso para um valor de x).

No caso de $\mathcal{N}_3$, se ρ atribui a y um valor maior que 0 então, analogamente ao caso anterior, quando o x na quantificação toma o valor 0, trivialmente a premissa da implicação $P(x,y) \rightarrow \exists y.\neg P(y,x)$ será verdadeira, porém a conclusão da implicação será falsa (sempre existe um valor que é maior ou igual que 0). Logo a implicação seria falsa para um valor de x e por tanto a fórmula como um todo seria falsa. Mas no caso de uma atribuição 0 para y, todo valor que possa tomar x nunca será menor que 0 e por tanto a premissa da implicação será falsa. Logo, a implicação como um todo é verdadeira. Por tanto, α seria contingente nessa interpretação.

Exemplo 6.5.12 *Seja a interpretação $\mathcal{N}_2$ do exemplo anterior. Então as fórmulas*

1. *$\forall x.(\neg P(x,y) \rightarrow P(y,x))$*
2. *$P(x,y) \rightarrow \forall x.P(x,y)$*

são verdadeiras e contingentes, respectivamente.

Para o primeiro caso, se x tomar uma valor que seja menor ou igual ao valor atribuído a y, então a premissa será falsa e por tanto a implicação como um todo será verdadeira. Já, se x tomar um valor maior que o valor atribuído a y então a premissa será verdadeira, mas a conclusão também pois o valor de y necessariamente será menor ou igual ao de x. Logo, independentemente do valor atribuído a y por um ρ, $\neg P(x,y) \rightarrow P(y,x)$ é verdadeira para qualquer valor de x.

Para o segundo caso, é suficiente ver que para a atribuição $\rho(x) = 2$ e $\rho(y) = 1$ a $P(x,y)$ é falso e portanto a fórmula é verdadeira enquanto para a atribuição $\rho(x) = 2$ e $\rho(y) = 3$ é falsa, pois $P(x,y)$ é verdadeira enquanto $\forall x.P(x,y)$ é falsa.

Exemplo 6.5.13 *A seguir veremos exemplos de conceitos que envolvem todas as interpretações possíveis.*

1. *A fórmula $\forall x.P(x) \rightarrow \exists x.P(x)$ é universalmente válida. Pois em qualquer interpretação $\mathcal{D}$ se $\mathcal{D} \models \forall x.P(x)$, significa que para qualquer atribuição ρ, $\mathcal{D} \models_{\rho} P(x)$, logo pelo menos para um ρ, $\mathcal{D} \models_{\rho} P(x)$, e portanto $\mathcal{D} \models \exists x.P(x)$. Informalmente a fórmula $\forall x.P(x) \rightarrow \exists x.P(x)$ declara que se uma propriedade é satisfeita por todo objeto numa interpretação, então essa propriedade é satisfeita por pelo menos um objeto na interpretação.*

2. *A fórmula $P(x) \wedge \neg P(x)$ é uma contradição, pois independente de como interpretemos P e do valor tomado por x, nunca poderemos ter que esse valor satisfaz e não satisfaz ao mesmo tempo a propriedade P.*

3. *A fórmula $\exists x.(\neg P(x) \vee P(f(x))$ é universalmente válida, pois em qualquer interpretação $\mathcal{D}$, tome um elemento arbitrário do universo de discurso, digamos a. Se $P_D(f_D(a))$ é verdadeiro então a fórmula é verdadeira para essa interpretação, se $P_D(f_D(a))$ for falso, então tome o valor $b = f_D(a)$. Claramente $\sim P_D(b) = \sim P_D(f_D(a)) = \sim 0 = 1$. Logo, sempre existe um x tal que $\sim P_D(x) + P_D(f_D(x)) = 1$. Portanto, a fórmula é verdadeira em qualquer interpretação.*

Observações:

1. $\forall x.\alpha$ é verdadeira numa interpretação se e somente se α é verdadeira na mesma interpretação.

2. $\exists x.\alpha$ é falsa numa interpretação se e somente se α é falsa na mesma interpretação.

3. α é universalmente válida se e somente se $\neg\alpha$ é uma contradição.

4. O fato de uma fórmula α não ser verdadeira numa interpretação $\mathcal{D}$, denotado por $\mathcal{D} \not\models \alpha$, não significa necessariamente que α seja falsa na interpretação $\mathcal{D}$, pois ela ainda pode ser contingente.

Definição 6.5.14 *Seja $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ uma linguagem de 1^a ordem e Γ um conjunto de fórmulas em $Ling(\mathcal{L})$. Dizemos que α é uma* **conseqüência lógica** *ou* **conseqüência semântica** *de Γ, cuja notação é, $\Gamma \models \alpha$, se todo modelo de Γ é um modelo de α.*

Exemplo 6.5.15 *Qualquer fórmula α é conseqüência lógica da fórmula $\forall x.\alpha$, isto é $\forall x.\alpha \models \alpha$, pois se o par $\langle \mathcal{D}, \rho \rangle$ é um modelo de $\forall x.\alpha$, então como a variável x não é livre, o valor atribuído por ρ a x é irrelevante. Portanto, α será verdadeira para essa interpretação e atribuições $\rho's$ que sejam iguais a ρ exceto, eventualmente, no valor atribuído a x. Logo, o par $\langle \mathcal{D}, \rho \rangle$ é um modelo para α.*

Observações:

1. Uma fórmula α é conseqüência lógica do vazio, isto é $\emptyset \models \alpha$ ou simplesmente $\models \alpha$ se e somente se α é universalmente válida.
2. **Teorema da dedução** $\Gamma, \alpha \models \beta$ se e somente se $\Gamma \models \alpha \rightarrow \beta$
3. **Teorema da Compacidade** Se $\Gamma \models \alpha$, então existe um subconjunto finito $\Gamma_0 = \{\alpha_1, \cdots, \alpha_n\} \subseteq \Gamma$ tal que $\Gamma_0 \models \alpha$.
4. Se uma fórmula α é insatisfatível (ou **contraditória**), então $\models \neg\alpha$.
5. Se $\Gamma \models \alpha$ e $\Theta \models \alpha \rightarrow \beta$, então $\Gamma \cup \Theta \models \beta$

Definição 6.5.16 *Seja L^P uma linguagem de 1^a ordem. A* **lógica de predicados clássica** *ou* **lógica de 1^a ordem** *associada a L^P, denotada por $Log^P_{L^P}$[1], é o subconjunto de todas as fórmulas universalmente válidas de L^P. Isto é,*

$$Log^P = \{\alpha \in L^P \ / \ \models \alpha\}.$$

É claro que em todas as tautologia de L_P, se substituirmos as variáveis proposicionais por fórmulas de L^P obteremos sempre uma fórmula universalmente válida. Chamamos estas fórmulas de 1^a ordem que podem ser obtidas a partir de uma tautologia de **fórmulas tautológicas** ou instâncias de uma tautologia. No entanto, a recíproca é falsa, isto é, existem, como veremos mais adiante, fórmulas universalmente válidas que não são tautológicas. Assim, o conceito de fórmula válida é uma extensão do conceito de tautologia em lógica proposicional, e portanto, podemos dizer que a lógica de predicados associada a uma linguagem de 1^a ordem L^P contém propriamente a lógica proposicional.

[1]Quando a linguagem L^P estiver clara do contexto omitiremos este sufixo, isto é, simplesmente usaremos a notação Log^P para a lógica de predicados clássicas sobre L^P.

Como os análogos teorema da dedução e o teorema da compacidade valem também para a lógica de predicados, então, analogamente à lógica proposicional, a lógica de predicados Log^P pode ser vista como um "aparato dedutivo", ou seja um ambiente onde possamos realizar deduções lógicas do tipo: "podemos deduzir α a partir das hipóteses Γ" que em notação lógica seria $\Gamma \models \alpha$.

Como podemos demonstrar que $\alpha_1, \ldots, \alpha_n \models \alpha$? Mostrando que todos os modelos de $\alpha_1, \ldots, \alpha_n$, também são modelos de α. Isto é impossível porque devemos analisar infinitos modelos. Uma alternativa, talvez, fosse demonstrar que $\alpha_1, \ldots, \alpha_n \models \alpha$ se e somente se $\models (\alpha_1 \rightarrow (\alpha_2 \rightarrow \ldots \alpha_n \rightarrow \alpha)) \ldots)$. Ou seja que certa fórmula β, no caso $\beta \equiv (\alpha_1 \rightarrow (\alpha_2 \rightarrow \ldots \alpha_n \rightarrow \alpha)) \ldots)$, é universalmente válida. Mas isso é também impossível, pois teríamos de mostrar que β é verdadeira para todas as interpretações (as quais são infinitas) e todas as atribuições. Isso significa que é impossível determinar a lógica de predicados através de interpretações ou modelos. A única alternativa é através de sistemas formais.

A seguir, algumas observações sobre o conceito de universalmente válida as quais são óbvias, mas algo dificultosas de demonstrar que são corretas.

Observações

1. Seja α uma fórmula cujas variáveis livres são $x_1, \ldots, x_n$. Então:
   (a) α é universalmente válida se, e somente se, seu fecho universal $\forall x_1. \ldots . \forall x_n . \alpha$ é universalmente válida. Onde fecho universal de α é a fórmula obtida de α por prefixar α com quantificadores universais ligando cada variável livre de α.
   (b) α é insatisfatível se, e somente se, seu fecho existencial $\exists x_1. \ldots . \exists x_n . \alpha$ é insatisfatível. Onde fecho existencial de α é a fórmula obtida de α por prefixar α com quantificadores existenciais ligando cada variável livre de α.
2. Se α não contém nenhuma ocorrência livre de x, então $\forall x.(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \forall x.\beta)$ é universalmente válida.
3. Se t é livre para x em α, então $\forall x.\alpha \rightarrow \alpha[t/x]$ é universalmente válida.

6.6 Exercícios

1. Determine se cada uma das variáveis x, y e z são livres, ligadas ou ambas em cada uma das fórmulas

   (a) $\forall x.P(f(x), y, z) \rightarrow \neg P(z, x, y)$

   (b) $\forall z.P(g(z), a, y) \wedge \forall y.\neg Q(x, y)$

   (c) $\forall x.P(x, y, z) \wedge \forall y.(Q(x, y) \wedge P(x, a, z))$

   (d) $\forall y.(P(y, a, z) \rightarrow \forall z.(Q(x, z) \wedge P(x, a, z)))$

2. Determine se ou não cada um dos termos: $h(x, y)$ e $g(a, f(y, z))$ é livre para cada uma das variáveis x, y, z em cada uma das seguintes fórmulas:

   (a) $\forall z.P(z, y, a) \wedge \forall x.Q(g(z, x))$

   (b) $\forall x.P(f(x, z), y) \rightarrow \neg P(f(x, a), z)$

   (c) $P(f(x, z), x) \rightarrow \forall z.\neg P(f(y, a), z)$

3. Diga e justifique para que variáveis o termo $f(b, g(y, z))$ é livre em

$$\neg(\forall x.(P(x, y) \rightarrow \exists y.R(y, z)) \vee (P(z, x) \rightarrow \exists x_1.P(y, x_1)))$$

4. Determine se cada uma das seguintes fórmulas é verdadeira, falsa ou contingente na interpretação:

   - Domínio: Conjuntos dos números inteiros $\mathbb{Z}$
   - Interpretação de P: $P_{\mathbb{Z}} \subseteq \mathbb{Z} \times \mathbb{Z}$ onde $P_{\mathbb{Z}} = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \ / \ x \leq y\}$. Assim, $P_{\mathbb{Z}}(n, m)$ se, e somente $n \leq m$.

   (a) $\forall x.\exists y.P(x, y) \rightarrow \exists y.\forall x.P(x, y)$

   (b) $\forall x.\forall y.P(x, y) \rightarrow \forall y.\forall x.P(x, y)$

   (c) $\exists x.\forall y.P(x, y)$

   (d) $\forall y.\exists x.P(x, y)$

   (e) $P(x, y) \rightarrow \forall x.\forall y.P(x, y)$

   (f) $\exists y.\forall x.(P(x, x) \rightarrow P(y, x))$

   (g) $\forall x.\exists y.(P(x, y) \rightarrow P(y, x))$

5. Indique quais das seguintes noções se aplica a cada uma das fórmula's listadas abaixo.

   (a) universalmente válida.

(b) verdadeira em alguma interpretação, mas não universalmente válida.

(c) satisfatível (verdadeira numa interpretação e para uma atribuição), porém não verdadeira em alguma (eventualmente outra) interpretação.

(d) falsa em alguma interpretação, mas não insatisfatível.

(e) insatisfatível.

i. $\forall x.P(z,x) \vee Q(x,y) \rightarrow (\forall x.P(z,x) \vee \forall x.Q(x,x))$
ii. $\forall x.\forall y.P(x,y) \rightarrow \forall x.\forall y.P(y,x)$
iii. $P(y,x) \rightarrow P(x,y)$
iv. $P(f(x)) \rightarrow P(x)$
v. $P(x,y) \wedge \neg\exists x.P(x,y)$
vi. $\forall x.(P(x) \rightarrow P(f(x)))$
vii. $\forall x.\exists y.(P(x,y) \wedge P(y,x))$
viii. $\forall x.P(x) \wedge \neg\exists y.P(y)$.

6. De uma interpretação que torne a seguinte fórmula falsa

$$\neg(\forall x.P(x,y) \wedge \exists y.P(y,z))$$

7. Dê exemplos de fórmulas para cada conjunto de condições relacionados a seguir. Justifique sua resposta.

(a) verdadeira em uma interpretação, mas não universalmente válida.

(b) nem verdadeira nem falsa para uma dada interpretação.

(c) satisfatível e não universalmente válida.

(d) verdadeira em alguma interpretação e falsa em outra interpretação .

(e) falsa em uma interpretação, mas não contraditória.

(f) universalmente válida.

(g) contraditória.

8. Sejam α_1, α_2 e α_3 as seguintes fórmulas:

α_1) $\forall x.(P(x,y) \vee \neg P(x,a)$
α_2) $\forall y.P(f(x),y)$
α_3) $P(f(a),a)$

Diga e justifique quais das seguintes alternativas é correta:

(a) $\alpha_1 \models \alpha_2$
(b) $\alpha_1 \models \alpha_3$

(c) $\alpha_2 \models \alpha_3$

(d) $\alpha_2 \models \alpha_1$

(e) $\alpha_3 \models \alpha_1$

(f) $\alpha_3 \models \alpha_2$

(g) $\alpha_1, \alpha_2 \models \alpha_3$

(h) $\alpha_1, \alpha_3 \models \alpha_2$

(i) $\alpha_2, \alpha_3 \models \alpha_1$

(j) $\alpha_1 \models \alpha_2 \rightarrow \alpha_3$

(k) $\alpha_1 \vee \alpha_2 \models \alpha_3$

(l) $\alpha_1 \wedge \alpha_2 \not\models \alpha_3$

(m) $\alpha_3 \not\models \alpha_1 \wedge \alpha_2$

(n) $\neg\alpha_1, \neg\alpha_2 \not\models \neg\alpha_3$

9. Demonstre que

(a) Se $\Gamma \models \alpha$ e $\Gamma \subseteq \Theta$, então $\Theta \models \alpha$

(b) Se $\Gamma \models \alpha$ e $\alpha \models \beta$ então $\Gamma \models \beta$

(c) Se para cada $\alpha \in \Theta$, $\Gamma \models \alpha$ e $\Theta \models \beta$ então $\Gamma \models \beta$

(d) Se $\Gamma \models \alpha$ então $\Gamma \not\models \neg\alpha$

Capítulo 7

A Teoria Formal da Lógica de Predicados

Um dos interesses na apresentação axiomática de lógicas elementares é a economia que pode ser conseguida nas regras e axiomas, isto é, quais axiomas e regras são usados para descrever a teoria formal requerida. Isto, combinado com uma estrutura de provas muito simples e direta do sistema, pode ser de considerável ajuda na investigação do que pode e não pode ser provado no sistema. Mas economia pode ir muito longe, e em alguns casos pode ser muito difícil manipular tal sistema. Assim, teremos como compromisso procurar por economia quando for relativamente simples, mas não ao custo de perda da legibilidade. Podemos economizar na linguagem, por considerar somente os símbolos lógicos $\neg$, $\rightarrow$ e $\forall$.

7.1 Teoria Formal do Cálculo de Predicados

Como toda tautologia é logicamente válida na lógica de predicados, então é razoável que todos os teoremas do cálculo proposicional (que pelo teorema da completude são todas as tautologias) sejam teoremas do cálculo de predicados. Assim, todos os axiomas e regras do cálculo proposicional devem fazer parte dos axiomas e regras do cálculo de predicados. Mas, a estes axiomas e regras devemos adicionar alguns que reflitam as características próprias do quantificador universal. Por exemplo, uma das mais óbvias propriedades é que se algo é válido para todo elemento de um domínio em discurso, então o mesmo é válido para um elemento específico, o qual é refletido pelo axioma

$$\forall x.\alpha(x) \rightarrow \alpha[t/x], \text{ onde } t \text{ é livre para } x \text{ em } \alpha.$$

Mas isto não é suficiente, devemos adicionar mais um axioma e uma regra de inferência. Assim, o sistema de provas ou **teoria formal para o cálculo de predicados**, $T^P = \langle \mathcal{L}^P, \Delta^P, \Re^P \rangle$, é dado por uma linguagem formal de 1ª ordem e pelos seguintes conjuntos de axiomas e regras de inferências.

7.1.1 Linguagem Formal:

Analogamente como fizemos para o caso proposicional, a linguagem formal de 1ª ordem considerada aqui, não considerará todos os conectivos lógicos nem todos os quantificadores. De fato só consideraremos o subconjunto $\{\neg, \rightarrow, \forall\}$ de Σ_L. Assim, a linguagem formal da teoria formal de primeira ordem, é $\mathcal{L}^P_{\rightarrow,\forall} = \langle \Sigma - \{\vee, \wedge, \leftrightarrow, \exists\}, \mathsf{F_1}, \mathsf{F_2}, \mathsf{F_5}, \mathsf{F_8}\rangle$, onde Σ é um alfabeto de 1ª ordem como definido no capítulo anterior. A linguagem gerada por esta linguagem formal será denotada por $L^P{}_{\rightarrow,\forall}$.

Axiomas:

(A_1) $\alpha \rightarrow (\beta \rightarrow \alpha)$

(A_2) $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$

(A_3) $(\neg\alpha \rightarrow \neg\beta) \rightarrow ((\neg\alpha \rightarrow \beta) \rightarrow \alpha)$

(A_4) $\forall x.(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \forall x.\beta)$, onde x não ocorre livre em α

(A_5) $\forall x.\alpha \rightarrow \alpha[t/x]$, onde t é livre para x em α.

Regras de Inferências:

Modus Ponens:

$$\mathsf{MP} : \frac{\alpha, \quad \alpha \rightarrow \beta}{\beta}$$

Generalização Universal:

$$\mathsf{Gen} : \frac{\alpha}{\forall x.\alpha}$$

Observações:

1. As definições de axiomas, prova, teorema e conseqüência, definidas de modo geral para qualquer teoria formal, assim como as observações 1-6 da seção 3.1, valem para a teoria formal de 1ª ordem aqui descrita. Portanto, a notação $\Gamma \vdash \alpha$ significa que temos uma prova de α nas hipótese de Γ. No caso particular de $\Gamma = \emptyset$, $\vdash \alpha$, denota que α é um teorema da lógica de predicados.

2. A lógica apresentada pela teoria formal $T^P = \langle \mathcal{L}^P_{\rightarrow,\forall}, \{A_1, \ldots, A_5\}, \{\mathsf{MP}, \mathsf{Gen}\}\rangle$ é denotada por $\mathcal{T}(T^P)$. Isto é, $\mathcal{T}(T^P)$ é o conjunto de todos os teoremas da teoria formal T^P:

$$\mathcal{T}(T^P) = \{\alpha \in L^P{}_{\rightarrow,\forall} \ / \ \vdash \alpha\}$$

3. O axioma A_5 é uma formalização de que se uma lei vale para todos, então, em particular, vale para um indivíduo escolhido aleatoriamente.

4. A regra Gen tem a seguinte explicação: Suponha que afirmamos a fórmula $cos^2 x + sen^2 x = 1$, digamos para $x \in \mathbb{R}$. O que, de fato, estamos dizendo é: $\forall x.(cos^2 x + sen^2 x = 1)$.

5. Teoremas envolvendo o quantificador existencial podem ser provadas usando a equivalência:

$$\exists x.\alpha \equiv \neg\forall x.\neg\alpha.$$

   Por exemplo, para provar a fórmula universalmente válida $\forall x.\alpha \to \exists x.\alpha$, se deve provar a fórmula $\forall x.\alpha \to \neg\forall x.\neg\alpha$.

Exemplo 7.1.1 *Seja o argumento "Todos os homens são mortais. Sócrates é homem. Logo Sócrates é mortal". Este argumento é análogo a outros argumentos como "Todos os políticos são corruptos. Fernando é político. Logo Fernando é corrupto", "Todos os nordestinos são brasileiros. João é Nordestino. Logo João é brasileiro", etc. Podemos provar este argumento usando a teoria formal de primeira ordem T^P. O primeiro passo é pôr o argumento como uma sentença da linguagem de 1^a ordem $L^P{}_{\to,\forall}$. Uma forma razoável seria usar símbolos de predicados mnemônicos, como H e M para denotar "é homem" e "é mortal" respectivamente. Assim, com estas considerações, o argumento é descrito através da sentença:*

$$\forall x.(H(x) \to M(x)), H(s) \vdash M(s)$$

Uma prova para este argumento é a seguinte:

$\alpha_1)$	$\forall x.(H(x) \to M(x))$	*hipótese*
$\alpha_2)$	$\forall x.(H(x) \to M(x)) \to (H(s) \to M(s))$	*instância de* A_5
$\alpha_3)$	$H(s) \to M(s)$	MP α_1, α_2
$\alpha_4)$	$H(s)$	*hipótese*
$\alpha_5)$	$M(s)$	MP α_4, α_3

O objeto principal da lógica clássica é o argumento ou raciocínio, isto é uma série de afirmações (premissas) das quais se extrai ou infere uma conclusão. Este processo recebe o nome de **inferência lógica**. No caso do exemplo 7.1.1 anterior a conclusão é "Sócrates é mortal" enquanto as premissas são: "Todos os homens são mortais" e "Sócrates é homem". Uma inferência lógica poderá ser válida ou inválida. Será válida se ela pode ser provada na teoria formal e inválida caso contrário. Como o argumento do exemplo 7.1.1 foi provado na teoria formal ele é válido.

Exemplo 7.1.2 *Vejamos o seguinte argumento lógico:*

"Nenhum homem pode ser homem e mulher ao mesmo tempo. Ora, Tiago é homem. Logo, Tiago não é mulher."

Neste argumento tem-se como premissas: "Nenhum homem pode ser homem e mulher ao mesmo tempo" e "Tiago é homem".

Considerando os seguintes símbolos de predicados: H para denotar "é homem" e M para denotar "é mulher".

Assim, com estas considerações, o argumento é descrito através da sentença:

$$\forall x.(M(x) \to \neg H(x)) \wedge (H(x) \to \neg M(x)), H(t) \vdash \neg M(t)$$

Ou equivalentemente:

$$\forall x.(M(x) \to \neg H(x)), \forall x.(H(x) \to \neg M(x)), H(t) \vdash \neg M(t)$$

Uma prova para este argumento é a seguinte:

$\alpha_1)$	$\forall x.(H(x) \to \neg M(x))$	*hipótese*
$\alpha_2)$	$\forall x.(H(x) \to \neg M(x)) \to (H(t) \to \neg M(t))$	*instância de* A_5
$\alpha_3)$	$H(t) \to \neg M(t)$	MP α_1, α_2
$\alpha_4)$	$H(t)$	*hipótese*
$\alpha_5)$	$\neg M(t)$	MP α_4, α_3

É de se esperar que adicionando dois novos axiomas e uma regra de inferência, o número de teoremas em T^P não diminua com respeito aos teoremas de T_P (a teoria formal proposicional). De fato podemos provar o seguinte resultado.

Proposição 7.1.3 *Toda fórmula $\alpha \in L^P{}_{\to,\forall}$ que é uma instância de uma tautologia, ou seja que é tautológica, é um teorema de T^P e pode ser provado usando-se somente os axiomas $A_1 - A_3$ e a regra* MP.

Demonstração: Se α resulta de uma tautologia β por substituição dos símbolos proposicionais em β por fórmulas atômicas em α, então pelo teorema da completude da lógica proposicional, β tem uma prova em T_P. Seja $\beta_1, \ldots, \beta_n$ uma prova de β em T_P. Substituímos em cada β_i as variáveis proposicionais pelas fórmulas atômicas em $L^P{}_{\to,\forall}$, da mesma maneira como realizado para obter α de β. Todo símbolo proposicional na prova que não ocorre em β é substituído por uma fórmula atômica arbitrária de $L^P{}_{\to,\forall}$. A sequência de fórmulas resultantes é uma prova de α a qual usa somente os axiomas $A_1 - A_3$ e a regra de inferência MP. ■

Portanto, tudo que pode ser provado em T_P, também, pode ser provado em T^P. No que segue estabeleceremos que T^P é consistente. Ou seja, $\mathcal{T}(T^P)$ é uma extensão de

$\mathcal{T}(T_P)$, mas não suficientemente grande para conter uma fórmula α e sua negação, $\neg\alpha$. Para provar esse resultado introduziremos uma função que transforma uma fórmula de L^P numa proposição.

Definição 7.1.4 *Seja $\mathcal{L} = \langle \Sigma, \mathcal{G} \rangle$ uma linguagem formal de 1^a ordem e $\mathcal{L}'$ uma linguagem proposicional cujos símbolos proposicionais são Σ_R. Então a função $\varphi : Ling(\mathcal{L}) \longrightarrow Ling(\mathcal{L}')$ definida por*

$\varphi(P(t_1, \ldots, t_n)) = p$ *para cada* $P \in \Sigma_R$

$\varphi(\neg\alpha) = \neg\varphi(\alpha)$

$\varphi(\alpha \wedge \beta) = \varphi(\alpha) \wedge \varphi(\beta)$

$\varphi(\alpha \vee \beta) = \varphi(\alpha) \vee \varphi(\beta)$

$\varphi(\alpha \rightarrow \beta) = \varphi(\alpha) \rightarrow \varphi(\beta)$

$\varphi(\alpha \leftrightarrow \beta) = \varphi(\alpha) \leftrightarrow \varphi(\beta)$

$\varphi(\forall x.\alpha) = \varphi(\alpha)$

$\varphi(\exists x.\alpha) = \varphi(\alpha)$

é denominada de **função de transformação de fórmulas em proposições**, *pois ela transforma uma fórmula α numa proposição, simplesmente extraindo de α os quantificadores e termos.*

Note que se $\alpha \in L^P{}_{\rightarrow,\forall}$ então $\varphi(\alpha) \in L_{P\rightarrow}$.

Exemplo 7.1.5 *Seja α a fórmula $\neg\forall x.P(x, y) \rightarrow R(z)$ e φ a função de transformação de fórmulas em proposições. Então*

$$\varphi(\alpha) = \varphi(\neg\forall x.P(x, y) \rightarrow R(z)) = \neg p \rightarrow r.$$

Assim $\varphi(\alpha)$ é uma fórmula proposicional com os símbolos de predicados de α como símbolos proposicionais.

Proposição 7.1.6 *Se α é um teorema em T^P, então $\varphi(\alpha)$ é uma tautologia.*

Demonstração: Para mostrar que $\varphi(\alpha)$ é uma tautologia cada vez que α é um teorema, devemos mostrar que $\varphi(\alpha)$ é uma tautologia para cada axioma α, e que as regras de inferência transformam fórmulas cujas imagens sob φ são tautologias em fórmulas cujas imagens sob φ são tautologias.

Por definição, $\varphi(\neg\alpha) = \neg(\varphi(\alpha))$ e $\varphi(\alpha \rightarrow \beta) = \varphi(\alpha) \rightarrow \varphi(\beta)$. Para cada axioma α $(A_1, \cdots, A_5)$, $\varphi(\alpha)$ é uma tautologia. Para A_1 a A_3, isto é imediato, pois eles são livres de quantificadores e termos.

A_4: Uma instância do axioma $\forall x.(\alpha \to \beta) \to (\alpha \to \forall x.\beta)$ será transformada pelo φ na tautologia $(\varphi(\alpha) \to \varphi(\beta)) \to (\varphi(\alpha) \to \varphi(\beta))$.

A_5: Uma instância de $\forall x.\alpha \to \alpha[t/x]$ será transformada pelo φ na tautologia $\varphi(\alpha) \to \varphi(\alpha)$.

O passo seguinte é checar as regras de inferências. Se $\varphi(\alpha)$ e $\varphi(\alpha \to \beta)$ (ou $\varphi(\alpha) \to \varphi(\beta)$) são tautologias, então pela definição de $\to$, também é $\varphi(\beta)$ e se $\varphi(\alpha)$ é uma tautologia, então $\varphi(\forall x.\alpha)$, que pela definição de φ é $\varphi(\alpha)$, também é uma tautologia.

Portanto $\varphi(\alpha)$ é uma tautologia para qualquer teorema α de T^P. ■

Note que a reversa não verdade, ou seja há fórmulas que não são teoremas, por exemplo $\alpha = \exists x.P(x) \to \forall x.P(x)$, mas que sua transformação canônica em fórmula proposicional, isto é $\varphi(\alpha)$, é uma tautologia.

Agora podemos provar o esperado resultado de consistência de $\mathcal{T}(T^P)$.

Teorema 7.1.7 *O Cálculo de Predicados é consistente.*

Demonstração: Se existe uma fórmula $\alpha \in L^P{}_{\to,\forall}$ tal que $\vdash \alpha$ e $\vdash \neg\alpha$ em T^P, então, pela proposição 7.1.6, ambos $\varphi(\alpha)$ e $\varphi(\neg\alpha)$ (ou $\neg\varphi(\alpha)$) devem ser tautologias, o qual é impossível. Portanto $\mathcal{T}(T^P)$ é consistente. ■

7.2 Teorema da Dedução

Na lógica proposicional o teorema da dedução foi uma ferramenta poderosa para auxiliar na prova de teoremas. O teorema da dedução para a lógica de predicados é também de muita utilidade, porém devemos tomar certos cuidados com alguns problemas sutis, tais como:

Para qualquer fórmula α, usando a regra de inferência Gen deduzimos $\forall x.\alpha$, e portanto provamos $\alpha \vdash \forall x.\alpha$. O "teorema da dedução" nos permitiria concluir que $\vdash \alpha \to \forall x.\alpha$, mas esta fórmula, como mostra o exemplo 6.5.12, não é universalmente válida.

O problema resulta no uso do "teorema da dedução" descartando uma hipótese para incluí-la como antecedente de uma implicação. Só poderíamos fazer isto se não tivéssemos aplicado Gen à hipótese (que esta sendo incluída como antecedente da implicação) ou a qualquer outra fórmula que dependa dela e que seja relevante na dedução da conclusão. Para esclarecer esta condição, precisamente, introduziremos o conceito "depende de".

Definição 7.2.1 *Seja α uma fbf no conjunto Γ de fbf's e seja $\alpha_1, \ldots, \alpha_n$ uma dedução de α_n a partir de Γ, junto com uma justificativa para cada passo da dedução. Dizemos que α_i, com $i = 1, \ldots, n$,* **depende de** *α nesta prova se:*

1. *α_i é α e a justificativa de α_i na prova é que pertence a Γ, ou*

2. *α_i é justificado como conseqüência direta do* MP *ou* Gen *de algumas das fórmulas anteriores, onde ao menos uma destas fórmulas depende de α.*

Exemplo 7.2.2 *Seja a seguinte dedução de $\alpha, \forall x.\alpha \to \beta \vdash \forall x.\beta$.*

$\alpha_1) \ \alpha$	*hipótese*
$\alpha_2) \ \forall x.\alpha$	Gen, α_1
$\alpha_3) \ \forall x.\alpha \to \beta$	*hipótese*
$\alpha_4) \ \beta$	MP, α_2, α_3
$\alpha_5) \ \forall x.\beta$	Gen, α_4

- *α_1 depende de α, pois α_1 é α e a justificativa é ser hipóteses.*
- *α_2 depende de α, pois α_2 é uma generalização de α_1 e α_1 depende de α.*
- *α_3 depende de $\forall x.\alpha \to \beta$, pois α_3 é $\forall x.\alpha \to \beta$ e a justifica é ser hipóteses.*
- *α_4 depende de α e de $\forall x.\alpha \to \beta$, pois α_4 é o modus ponens de α_2 e α_3, e α_2 depende de α e α_3 depende de $\forall x.\alpha \to \beta$.*
- *α_5 depende de α e de $\forall x.\alpha \to \beta$, pois α_5 é a generalização de α_4 e α_4 depende de âmbos (α e $\forall x.\alpha \to \beta$).*

O seguinte lema declara que se uma conclusão não depende de uma hipótese, então podemos realizar a prova sem precisar desta hipótese. Portanto, podemos omiti-la da dedução.

Lema 7.2.3 *Se β não depende de α numa dedução de $\Gamma, \alpha \vdash \beta$, então $\Gamma \vdash \beta$.*

Demonstração: Seja $\beta_1, \ldots, \beta_n = \beta$ uma prova de β a partir de Γ e α, na qual β não depende de α. Provaremos por indução sobre n, o número de passos na prova, que $\Gamma \vdash \beta_i$, para todo β_i, com $1 \leq i \leq n$, que não depende de α. Assim, no caso de $i = n$, $\Gamma \vdash \beta$, pois β não depende de α.

Caso base: $i = 1$.
Se β_1 não depende de α, então ou β_1 é um axioma ou $\beta_1 \in \Gamma$. Logo, $\Gamma \vdash \beta_1$.

Etapa indutiva: Assuma que para cada $1 \leq i < k$, se β_i não depende de α então $\Gamma \vdash \beta_i$. Mostre que se β_k não depende de α então $\Gamma \vdash \beta_k$.

Se β_k pertence a Γ ou é um axioma, então $\Gamma \vdash \beta_k$. Se β é conseqüência direta de uma ou duas fórmulas anteriores, então, por β_k não depender de α, estas fórmulas também não dependem de α. Logo, pela hipótese indutiva, estas fórmulas são dedutíveis de Γ. Portanto, β_k também é dedutível a partir de Γ. ■

Teorema 7.2.4 (Teorema da Dedução para T^P) *Se existe uma prova de $\Gamma, \alpha \vdash \beta$, tal que toda aplicação de* Gen *a uma fórmula que depende de α tem como variável quantificada uma variável não livre de α. Então existe uma prova de $\Gamma \vdash \alpha \to \beta$.*

DEMONSTRAÇÃO: Seja $\beta_1, \ldots, \beta_n = \beta$ uma prova de β a partir de Γ, α tal que satisfaz a condição do teorema. Mostraremos por indução, sobre i, que $\Gamma \vdash \alpha \to \beta_i$ para cada $i \leq n$.

Caso base: $i = 1$.

1. β_1 é um axioma ou pertence a Γ.

   $\alpha_1)$ β_1 algum dos axiomas ou uma hipóteses
   $\alpha_2)$ $\beta_1 \to (\alpha \to \beta_1)$ A_1
   $\alpha_3)$ $\alpha \to \beta_1$ MP, α_1, α_2

   Logo, $\alpha_1, \alpha_2, \alpha_3$ é uma prova de $\Gamma \vdash \alpha \to \beta_1$

2. β_1 é α.

   $\alpha_1)$ $\alpha \to \beta_1$ proposição 4.2.1

   Logo, α_1 é uma prova de $\Gamma \vdash \alpha \to \beta_1$

Etapa indutiva: Assuma que $\Gamma \vdash \alpha \to \beta_i$ para cada $i < k$; mostre que $\Gamma \vdash \alpha \to \beta_k$.

1. Se β_k é um axioma ou um elemento de Γ, o resultado segue como no caso base 1.

2. Se β_k é α, o resultado segue como no caso base 2.

3. Se β_k sai por MP de β_i e $\beta_j = \beta_i \to \beta_k$ com $i, j < k$, então pela hipóteses indutiva $\Gamma \vdash \alpha \to \beta_i$ e $\Gamma \vdash \alpha \to \beta_j$. Sejam $\alpha_{i_1}, \ldots, \alpha_{i_m}$ e $\alpha_{j_1}, \ldots, \alpha_{j_n}$ provas de $\Gamma \vdash \alpha \to \beta_i$ e $\Gamma \vdash \alpha \to \beta_j$, respectivamente. A seqüência

$\alpha_1.$	α_{i_1}	
$\vdots$	$\vdots$	
$\alpha_m.$	α_{i_m}	onde $\alpha_{i_m} = \beta_i$
$\alpha_{m+1}.$	α_{j_1}	
$\vdots$	$\vdots$	
$\alpha_{m+n}.$	α_{j_n}	onde $\alpha_{j_n} = \beta_j = \beta_i \to \beta_k$
$\alpha_{m+n+1}.$	$(\alpha \to (\beta_i \to \beta_k)) \to ((\alpha \to \beta_i) \to (\alpha \to \beta_k))$	A_2
$\alpha_{m+n+2}.$	$(\alpha \to \beta_i) \to (\alpha \to \beta_k)$	MP, $\alpha_{m+n}, \alpha_{m+n+1}$
$\alpha_{m+n+3}.$	$\alpha \to \beta_k$	MP, α_m, α_{m+n+2}

   é uma prova de $\Gamma \vdash \alpha \to \beta_k$.

4. Se β_k sai por Gen de um β_i, $i < k$, então $\beta_k = \forall x.\beta_i$. Pela hipótese indutiva $\Gamma \vdash \alpha \to \beta_i$. Neste caso teremos duas possibilidades:

(a) Se β_i não depende de α, na dedução de β a partir de Γ e α, então pelo lema 7.2.3 aplicado à hipóteses indutiva temos que $\Gamma \vdash \beta_i$. Seja $\alpha_{i_1}, \ldots, \alpha_{i_n}$ uma prova de $\Gamma \vdash \beta_i$. Então a seqüência

α_1.	α_{i_1}	
$\vdots$	$\vdots$	
α_n.	α_{i_n}	onde $\alpha_{i_n} = \beta_i$
α_{n+1}.	$\forall x.\beta_i$	Gen(onde $\forall x.\beta_i = \beta_k$)
α_{n+2}.	$\beta_k \to (\alpha \to \beta_k)$	A_1
α_{n+3}.	$\alpha \to \beta_k$	MP, $\alpha_{n+1}, \alpha_{n+2}$

é uma prova de $\Gamma \vdash \alpha \to \beta_k$.

(b) Se β_i não depende de α, na dedução de β a partir de Γ e α, então, pelas condições do enunciado deste teorema, x não é livre em α. Seja $\alpha_{i_1}, \ldots, \alpha_{i_n}$ uma prova de $\Gamma \vdash \alpha \to \beta_i$. Logo, a seqüência

α_1.	α_{i_1}	
$\vdots$	$\vdots$	
α_n.	α_{i_n}	onde $\alpha_{i_n} = \alpha \to \beta_i$
α_{n+1}.	$\forall x.(\alpha \to \beta_i)$	Genα_{i_n}
α_{n+2}.	$\forall x.(\alpha \to \beta_i) \to (\alpha \to \forall x.\beta_i)$	A_5
α_{n+3}.	$\alpha \to \forall x.\beta_i$	MP, $\alpha_{n+1}, \alpha_{n+2}$

é uma prova de $\Gamma \vdash \alpha \to \beta_k$.

Assim, $\Gamma \vdash \alpha \to \beta_i$ para cada $1 \leq i \leq n$. O teorema resulta do caso $i = n$. ■

As hipóteses (restrições) do teorema da dedução para T^P são pouco claras. Os seguintes corolários tornam o teorema mais transparente, e portanto mais útil.

Corolário 7.2.5 *Se uma prova de $\Gamma, \alpha \vdash \beta$ não envolve nenhuma aplicação de* Gen *na qual a variável quantificada é livre em α, então $\Gamma \vdash \alpha \to \beta$.* ■

Corolário 7.2.6 *Se α é fechado, isto é não contém variáveis livres, e $\Gamma, \alpha \vdash \beta$, então $\Gamma \vdash \alpha \to \beta$.* ■

Corolário 7.2.7 *Se $\alpha_1, \ldots, \alpha_n$ é uma prova de $\Gamma, \alpha \vdash \beta$ e β não depende de α na prova, então $\Gamma \vdash \alpha \to \beta$.* ■

Para não ter que lidar com uma série de casos especiais, como os três corolários anteriores (que não são suficientes para considerar todas as possibilidades), nem ter que verificar se uma determinada prova de $\Gamma, \alpha \vdash \beta$ satisfaz ou não as restrições do teorema da dedução (que pode ser motivo de confusão), é melhor realizar a seguinte seqüência de perguntas:

1. α tem alguma variável livre?
2. Gen foi utilizado na prova?
3. Algumas das variáveis introduzidas pelo Gen é livre em α?
4. Algumas das fórmulas na qual se aplicou Gen introduzindo variáveis livres de α depende de α?
5. β depende de algumas das fórmulas na qual se aplicou Gen introduzindo variáveis livres de α?

Se algumas das perguntas receber uma resposta "não", então podemos concluir que existe uma prova de $\Gamma \vdash \alpha \to \beta$. Do contrário, ou seja caso todas as perguntas tenham respostas "sim", essa prova em particular não nos autoriza a "aplicar" o teorema da dedução[1] e portanto não podemos afirmar que existe uma prova para $\Gamma \vdash \alpha \to \beta$.

Exemplo 7.2.8 *Seja a seguinte prova de* $\alpha \vdash \neg\forall x.\alpha \to \forall x.\neg\alpha$*:*

$\alpha_1)$	α	*hip.*
$\alpha_2)$	$\forall x.\alpha$	Gen, α_1
$\alpha_3)$	$\forall x.\alpha \to (\alpha \to \forall x.\alpha)$	A_1
$\alpha_4)$	$\alpha \to \forall x.\alpha$	MP, α_2, α_3
$\alpha_5)$	$(\alpha \to \forall x.\alpha) \to (\neg\forall x.\alpha \to \neg\alpha)$	*proposição* 4.3.3.*h*
$\alpha_6)$	$\neg\forall x.\alpha \to \neg\alpha$	MP, α_4, α_5
$\alpha_7)$	$\forall x.(\neg\forall x.\alpha \to \neg\alpha)$	Gen, α_6
$\alpha_8)$	$\forall x.(\neg\forall x.\alpha \to \alpha) \to (\neg\forall x.\alpha \to \forall x.\neg\alpha)$	A_4
$\alpha_9)$	$\neg\forall x.\alpha \to \forall x.\neg\alpha$	MP, α_7, α_8

Será que esta prova junto com o teorema da dedução garantem que podemos transformar $\alpha_1, \ldots, \alpha_9$ *numa prova de* $\vdash \alpha \to (\neg\forall x.\alpha \to \forall x.\neg\alpha)$*? Para responder isto teremos que verificar se esta prova satisfaz as condições do teorema da dedução. Para isto façamos as 5 perguntas acima:*

1. *α tem alguma variável livre?*

   Como α é uma meta-variável, ou seja pode ser instanciada com qualquer fórmula, inclusive uma tendo variáveis livres. Logo, a resposta a esta pergunta é: Sim.

2. Gen *foi utilizado na prova?*

   Sim, em α_2 e α_7.

[1] Observe que um (meta)teorema da forma "Se α então β" sempre se aplica, só que quando α é falso, não podemos concluir nada de β. Assim, a expressão "não é possível aplicar o teorema da dedução" não é do todo correta, mas nós a usaremos com a interpretação que as premissas do teorema da dedução não são satisfeitas. Analogamente, a expressão " é possível aplicar o teorema da dedução" será usada para indicar que as premissas do teorema da dedução foram satisfeitas.

3. *Algumas das variáveis introduzidas pelo* Gen *é livre em* α*?*

   Como α *é uma meta-variável, ou seja pode ser instanciada com qualquer fórmula, inclusive uma tendo* x *como variável livre. Logo, a resposta a esta pergunta é: Sim.*

4. *Algumas das fórmulas na qual se aplicou* Gen *introduzindo variáveis livres de* α *depende de* α*?*

   Sim, tanto α_2 *quanto* α_7 *dependem de* α*.*

5. β *(neste caso* $\alpha_9 = \neg\forall x.\alpha \rightarrow \forall x.\neg\alpha$*) depende de algumas das fórmulas na qual se aplicou* Gen *introduzindo variáveis livres de* α*?*

   Sim, ela depende tanto de α_2 *quanto de* α_7*.*

Logo, como as cinco questões tiveram respostas positivas, podemos concluir que esta prova não nos permite concluir a existência de uma prova para $\vdash \alpha \rightarrow (\neg\forall x.\alpha \rightarrow \forall x.\neg\alpha)$*.*

Exemplo 7.2.9 *Seja a seguinte prova de* $P(x,z), \neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y) \vdash R(a,f(a))$

α_1	$P(x,z)$	$Hip.$
α_2	$\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)$	$Hip.$
α_3	$(\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)) \rightarrow (\forall x.P(x,y) \rightarrow \forall y.R(y,z))$	$prop.3.3.4.d$
α_4	$\forall x.P(x,y) \rightarrow \forall y.R(y,z)$	$MP\ \ \alpha_2, \alpha_3$
α_5	$\forall z.P(x,z)$	$Gen\ \ \alpha_1$
α_6	$\forall z.P(x,z) \rightarrow P(x,y)$	A_5
α_7	$P(x,y)$	$MP\ \ \alpha_5, \alpha_6$
α_8	$\forall x.P(x,y)$	$Gen\ \ \alpha_7$
α_9	$\forall y.R(y,z)$	$MP\ \ \alpha_8, \alpha_4$
α_{10}	$\forall y.R(y,z) \rightarrow R(a,z)$	A_5
α_{11}	$R(a,z)$	$MP\ \ \alpha_9, \alpha_{10}$
α_{12}	$\forall z.R(a,z)$	$Gen\ \ \alpha_{11}$
α_{13}	$\forall z.R(a,z) \rightarrow R(a,f(a))$	A_5
α_{14}	$R(a,f(a))$	$MP\ \ \alpha_{12}, \alpha_{13}$

Como, tanto a hipóteses $P(x,z)$ *quanto* $\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)$ *têm variáveis livres.* Gen *foi usado na prova. A variável* z *introduzida pelo* Gen *em* α_5 *é livre em ambas as hipóteses. E como* α_5 *depende de* $P(x,z)$ *e* α_{12} *(onde foi aplicado* Gen *introduzindo a variável* z*) depende de* $\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)$*, não podemos garantir que exista uma prova para*

$$P(x,z) \vdash ((\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)) \rightarrow R(a,f(a)))$$

nem para

$$(\neg\forall y.R(y,z) \rightarrow \neg\forall x.P(x,y)) \vdash P(x,z) \rightarrow R(a,f(a)).$$

Exemplo 7.2.10 *Seja a seguinte prova de* $\forall x.P(x,z) \vdash (\neg\forall y.P(y,a) \to \neg\forall x.P(x,a)) \to P(a,z)$

$\alpha_1)$	$\forall x.P(x,z)$	*hip.*
$\alpha_2)$	$\forall x.P(x,z) \to ((\neg\forall y.P(y,a) \to \neg\forall x.P(x,a)) \to \forall x.P(x,z))$	A_1
$\alpha_3)$	$(\neg\forall y.P(y,a) \to \neg\forall x.P(x,a)) \to \forall x.P(x,z)$	MP α_1, α_2
$\alpha_4)$	$\forall x.P(x,z) \to P(a,z)$	A_5
$\alpha_5)$	$(\neg\forall y.P(y,a) \to \neg\forall x.P(x,a)) \to P(a,z)$	*proposição 4.3.2.i* α_3, α_4

Como, Gen *não foi usado na prova então o teorema da dedução garante a existência de uma prova para*

$$\vdash \forall x.P(x,z) \to ((\neg\forall y.P(y,a) \to \neg\forall x.P(x,a)) \to P(a,z))$$

Observe que na prova do teorema da dedução para T^P, β_i depende de uma premissa γ de Γ na prova original de $\Gamma, \alpha \vdash \beta_i$ se, e somente se, $\alpha \to \beta_i$ depende de γ na nova prova de $\Gamma \vdash \alpha \to \beta_i$. Esta observação é útil quando desejamos aplicar o teorema da dedução diversas vezes consecutivas (por exemplo, para obter $\Gamma \vdash \gamma \to (\alpha \to \beta)$ de $\Gamma, \gamma, \alpha \vdash \beta$).

Exemplo 7.2.11 *Para provar que* $\forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$ *é um teorema de* $\mathcal{T}(T^P)$*, primeiro provamos que*

$$\forall x.(\alpha \to \beta), \forall x.\alpha \vdash \forall x.\beta$$

para depois argumentar que, pelo teorema de dedução, existe uma prova de $\forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$.

Seja a seguinte prova de $\forall x.(\alpha \to \beta), \forall x.\alpha \vdash \forall x.\beta$*:*

$\alpha_1)$	$\forall x.\alpha$	*hip.*
$\alpha_2)$	$\forall x.(\alpha \to \beta)$	*hip.*
$\alpha_3)$	$\forall x.(\alpha \to \beta) \to (\alpha \to \beta)$	A_5
$\alpha_4)$	$\alpha \to \beta$	MP, α_2, α_3
$\alpha_5)$	$\forall x.\alpha \to \alpha$	A_5
$\alpha_6)$	α	MP, α_1, α_5
$\alpha_7)$	β	MP, α_6, α_4
$\alpha_8)$	$\forall x.\beta$	Gen, α_7

Como a única generalização realizada na prova introduz a variável x*, a qual não é livre em* $\forall x.\alpha$*, podemos aplicar o teorema da dedução e concluir que existe uma prova para* $\forall x.(\alpha \to \beta) \vdash \forall x.\alpha \to \forall x.\beta$.

Observe que o teorema da dedução só garante que existe uma prova para $\forall x.(\alpha \to \beta) \vdash \forall x.\alpha \to \forall x.\beta$*, mas não proporciona efetivamente essa prova. Logo pode ficar a seguinte pergunta: Como saber se é possível ou não aplicar novamente o teorema da dedução se não possuímos nenhuma prova concreta de* $\forall x.(\alpha \to \beta) \vdash \forall x.\alpha \to \forall x.\beta$*?*

Ou seja, não temos nenhuma prova tangível sob a qual possamos analisar e descobrir se essa prova satisfaz ou não as restrições impostas pelo teorema da dedução para seu uso.

Mas, embora a resposta à pergunta seja aparentemente: "não podemos saber", baseados na demonstração do teorema da dedução para a lógica de 1ª ordem (teorema 7.2.4), sabemos que uma prova de $\forall x.(\alpha \to \beta) \vdash \forall x.\alpha \to \forall x.\beta$, pode ser obtida substituindo cada β_i na prova de $\forall x.(\alpha \to \beta), \forall x.\alpha \vdash \forall x.\beta$ por uma série de fórmulas de acordo com o tratamento indutivo da demonstração do teorema da dedução. Assim, por exemplo a partir de $\alpha_1, \ldots, \alpha_8$, obteríamos a seguinte prova de $\forall x.(\alpha \to \beta) \vdash \forall x.\alpha \to \forall x.\beta$:

$\alpha_{1.a})$	$\forall x.\alpha \to \forall x.\alpha$	*prop.3.2.1.*
$\alpha_{2.a})$	$\forall x.(\alpha \to \beta)$	*hip.*
$\alpha_{2.b})$	$\forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.(\alpha \to \beta))$	*A1*
$\alpha_{2.c})$	$\forall x.\alpha \to \forall x.(\alpha \to \beta)$	MP, $\alpha_{2.a}, \alpha_{2.b}$
$\alpha_{3.a})$	$\forall x.(\alpha \to \beta) \to (\alpha \to \beta)$	A_5
$\alpha_{3.b})$	$(\forall x.(\alpha \to \beta) \to (\alpha \to \beta)) \to (\forall x.\alpha \to (\forall x.(\alpha \to \beta) \to (\alpha \to \beta)))$	*A1*
$\alpha_{3.c})$	$\forall x.\alpha \to (\forall x.(\alpha \to \beta) \to (\alpha \to \beta))$	MP, $\alpha_{3.a}, \alpha_{3.b}$
$\alpha_{4.a})$	$(\forall x.\alpha \to (\forall x.(\alpha \to \beta) \to (\alpha \to \beta))) \to ((\forall x.\alpha \to \forall x.(\alpha \to \beta)) \to (\forall x.\alpha \to (\alpha \to \beta)))$	*A2*
$\alpha_{4.b})$	$((\forall x.\alpha \to \forall x.(\alpha \to \beta)) \to (\forall x.\alpha \to (\alpha \to \beta)))$	MP, $\alpha_{3.c}, \alpha_{4.a}$
$\alpha_{4.c})$	$\forall x.\alpha \to (\alpha \to \beta)$	MP, $\alpha_{2.c}, \alpha_{4.b}$
$\alpha_{5.a})$	$\forall x.\alpha \to \alpha$	A_5
$\alpha_{5.b})$	$(\forall x.\alpha \to \alpha) \to (\forall x.\alpha \to (\forall x.\alpha \to \alpha))$	*A1*
$\alpha_{5.c})$	$\forall x.\alpha \to (\forall x.\alpha \to \alpha)$	MP, $\alpha_{5.a}, \alpha_{5.b}$
$\alpha_{6.a})$	$(\forall x.\alpha \to (\forall x.\alpha \to \alpha)) \to ((\forall x.\alpha \to \forall x.\alpha) \to (\forall x.\alpha \to \alpha))$	*A2*
$\alpha_{6.b})$	$(\forall x.\alpha \to \forall x.\alpha) \to (\forall x.\alpha \to \alpha)$	MP, $\alpha_{5.c}, \alpha_{6.a}$
$\alpha_{6.c})$	$\forall x.\alpha \to \alpha$	MP, $\alpha_{1.a}, \alpha_{6.b}$
$\alpha_{7.a})$	$(\forall x.\alpha \to (\alpha \to \beta)) \to ((\forall x.\alpha \to \alpha) \to (\forall x.\alpha \to \beta))$	*A2*
$\alpha_{7.b})$	$(\forall x.\alpha \to \alpha) \to (\forall x.\alpha \to \beta)$	MP, $\alpha_{4.c}, \alpha_{7.a}$
$\alpha_{7.c})$	$\forall x.\alpha \to \beta$	MP, $\alpha_{6.c}, \alpha_{7.a}$
$\alpha_{8.a})$	$\forall x.(\forall x.\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$	*A5*
$\alpha_{8.b})$	$\forall x.(\forall x.\alpha \to \beta)$	Gen, $\alpha_{7.c}$
$\alpha_{8.c})$	$\forall x.\alpha \to \forall x.\beta$	MP, $\alpha_{8.b}, \alpha_{8.a}$

Logo, podemos analisar esta prova e verificar se realmente satisfaz as restrições do teorema da dedução. No entanto, fica evidente que seria um trabalho intrincado e tedioso construir uma prova de este forma. Felizmente, se fizermos uma análises mais minuciosa da prova assim obtida, verificaremos que esta prova tem as mesmas propriedades que a original: só foi usada uma generalização, a qual depende de uma fórmula $\alpha_{7.c}$, a fórmula $\alpha_{7.c}$ depende das fórmulas $\alpha_{6.c}$ e $\alpha_{7.a}$ (este é um axioma), a $\alpha_{6.c}$ depende da $\alpha_{1.a}$ e $\alpha_{6.b}$. e $\alpha_{1.a}$ é a fórmula correspondente à hipótese $\forall x.\alpha$ (α_1) na prova original.

Assim, para saber se podemos aplicar o teorema da dedução baseados na prova construída anteriormente, basta analisar a prova original. Logo, usando o mesmo critério da primeira análises, podemos afirmar que é possível aplicar novamente o teorema da dedução e concluir, desse modo, que existe uma prova para $\vdash \forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$ como desejado.

Observação: Generalizando o resultado do exemplo anterior, podemos dizer que se $\beta_1, \ldots, \beta_n$ é uma prova de $\alpha_1, \ldots, \alpha_k \vdash \alpha$ tal que nos permite concluir, pelo teorema

da dedução, que existe uma prova de $\alpha_1, \ldots, \alpha_{k-1} \vdash \alpha_k \rightarrow \alpha$, então para saber se é possível aplicar novamente o teorema da dedução e assim concluir a existência de uma prova para $\alpha_1, \ldots, \alpha_{k-2} \vdash \alpha_{k-1} \rightarrow (\alpha_k \rightarrow \alpha)$, basta analisar a prova $\beta_1, \ldots, \beta_n$. Obviamente, se queremos verificar se novamente podemos aplicar o teorema da dedução e concluir a existência de uma prova para $\alpha_1, \ldots, \alpha_{k-3} \vdash \alpha_{k-2} \rightarrow (\alpha_{k-1} \rightarrow (\alpha_k \rightarrow \alpha))$ basta verificar na prova $\beta_1, \ldots, \beta_n$. E assim por diante. Portanto, se $\beta_1, \ldots, \beta_n$ é uma prova de $\alpha_1, \ldots, \alpha_k \vdash \alpha$ e queremos saber se existe uma prova de $\vdash \alpha_1 \rightarrow (\alpha_2 \rightarrow \ldots (\alpha_k \rightarrow \alpha) \ldots)$, então devemos somente verificar se o teorema da dedução se aplica na prova $\beta_1, \ldots, \beta_n$ para α_1, para $\alpha_2, \ldots$, para α_k, separadamente.

Vejamos alguns resultados que serão úteis para provar teoremas em T^P.

Proposição 7.2.12 *a)* $\vdash \forall x.\forall y.\alpha \rightarrow \forall y.\forall x.\alpha$

b) $\vdash (\forall x.\alpha \rightarrow \forall x.\beta) \rightarrow \forall x.(\alpha \rightarrow \beta)$

c) $\vdash \neg\alpha \rightarrow \neg\forall x.\alpha$

d) $\vdash \beta \rightarrow \alpha \vdash \neg\forall x.\alpha \rightarrow \neg\forall x.\beta$

e) $\vdash \forall x.\alpha \rightarrow \neg\forall x.\neg\alpha$

f) $\vdash \neg\forall x.\neg\forall y.\alpha \rightarrow \forall y.\neg\forall x.\neg\alpha$

g) $\vdash \forall x.\neg\alpha \rightarrow \neg\forall x.\alpha$

Demonstração:

a) $\vdash \forall x.\forall y.\alpha \rightarrow \forall y.\forall x.\alpha$

α_1)	$\forall x.\forall y.\alpha$	*hip.*
α_2)	$\forall x.\forall y.\alpha \rightarrow \forall y.\alpha$	A_5
α_3)	$\forall y.\alpha$	MP α_1, α_2
α_4)	$\forall y.\alpha \rightarrow \alpha$	A_5
α_5)	α	MP α_3, α_4
α_6)	$\forall x.\alpha$	Gen α_5
α_7)	$\forall y.\forall x.\alpha$	Gen α_6

Logo, $\alpha_1, \ldots, \alpha_7$ é uma prova de $\forall x.\forall y.\alpha \vdash \forall y.\forall x.\alpha$. Fazendo as 4 perguntas temos que a hipótese $\forall x.\forall y.\alpha$ pode ter variáveis livres, pois α é uma meta variável representando qualquer fórmula e uma dessas pode ser $P(z)$ por exemplo. Generalização foi usada na prova, mas as variáveis introduzidas por Gen (x e y) não são livres em $\forall x.\forall y.\alpha$ independente de quem seja α. Logo, o teorema da dedução, garante que existe uma prova de $\vdash \forall x.\forall y.\alpha \rightarrow \forall y.\forall x.\alpha$.

b) $\vdash \forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$

α_1)	$\forall x.(\alpha \to \beta)$	*hip.*
α_2)	$\forall x.(\alpha \to \beta) \to (\alpha \to \beta)$	A_5
α_3)	$\alpha \to \beta$	MP α_1, α_2
α_4)	$\forall x.\alpha$	*hip.*
α_5)	$\forall x.\alpha \to \alpha$	A_5
α_6)	α	MP α_4, α_5
α_7)	β	MP α_6, α_3
α_8)	$\forall x.\beta$	Gen α_7

$\alpha_1, \ldots, \alpha_8$ ´é uma prova de $\forall x.(\alpha \to \beta), \forall x.\alpha \vdash \forall x.\beta$. Como a única generalização na prova introduz a variável x, a qual não é livre nem em $\forall x.(\alpha \to \beta)$ nem em $\forall x.\alpha$, então pelo teorema da dedução (junto com a observação **??**) temos que podemos construir uma prova para $\vdash \forall x.(\alpha \to \beta) \to (\forall x.\alpha \to \forall x.\beta)$ como desejado.

c) $\vdash \neg\alpha \to \neg\forall x.\alpha$

α_1)	$\forall x.\alpha \to \alpha$	A_5
α_2)	$(\forall x.\alpha \to \alpha) \to (\neg\alpha \to \neg\forall x.\alpha)$	proposição 4.3.3.h
α_3)	$\neg\alpha \to \neg\forall x.\alpha$	MP α_1, α_2

d) $\beta \to \alpha \vdash \neg\forall x.\alpha \to \neg\forall x.\beta$

α_1)	$\beta \to \alpha$	*hip.*
α_2)	$\forall x.(\beta \to \alpha)$	Gen α_1
α_3)	$\forall x.(\beta \to \alpha) \to (\forall x.\beta \to \forall x.\alpha)$	item b)
α_4)	$\forall x.\beta \to \forall x.\alpha$	MPα_2, α_3
α_5)	$(\forall x.\beta \to \forall x.\alpha) \to (\neg\forall x.\alpha \to \neg\forall x.\beta)$	proposição 4.3.3.h
α_6)	$\neg\forall x.\alpha \to \neg\forall x.\beta$	MPα_4, α_5

e) $\vdash \forall x.\alpha \to \neg\forall x.\neg\alpha$

α_1)	$\forall x.\alpha$	*hip.*
α_2)	$(\neg\neg\forall x.\neg\alpha \to \neg\alpha) \to ((\neg\neg\forall x.\neg\alpha \to \alpha) \to \neg\forall x.\neg\alpha)$	A_3
α_3)	$\neg\neg\forall x.\neg\alpha \to \forall x.\neg\alpha$	proposição 4.3.3.a
α_4)	$\forall x.\neg\alpha \to \neg\alpha$	A_5
α_5)	$\neg\neg\forall x.\neg\alpha \to \neg\alpha$	proposição 4.3.2.i α_3, α_4
α_6)	$(\neg\neg\forall x.\neg\alpha \to \alpha) \to \neg\forall x.\neg\alpha$	MP α_5, α_2
α_7)	$\forall x.\alpha \to \alpha$	A_5
α_8)	α	MP α_1, α_7
α_9)	$\alpha \to (\neg\neg\forall x.\neg\alpha \to \alpha)$	A_1
α_{10})	$\neg\neg\forall x.\neg\alpha \to \alpha$	MP α_8, α_9
α_{11})	$\neg\forall x.\neg\alpha$	MP α_{10}, α_6

Logo, $\alpha_1, \ldots, \alpha_{11}$ é uma prova de $\forall x.\alpha \vdash \neg\forall x.\neg\alpha$. Como Gen não foi usado na prova então o teorema da dedução nos garante que existe uma prova de $\vdash \forall x.\alpha \to \neg\forall x.\neg\alpha$.

f) $\vdash \neg\forall x.\neg\forall y.\alpha \rightarrow \forall y.\neg\forall x.\neg\alpha$

$\alpha_1)$	$\neg\alpha \rightarrow \neg\forall y.\alpha$	item c
$\alpha_2)$	$\neg\forall x.\neg\forall y.\alpha \rightarrow \neg\forall x.\neg\alpha$	item d, α_1
$\alpha_3)$	$\neg\forall x.\neg\forall y.\alpha$	*hip.*
$\alpha_4)$	$\neg\forall x.\neg\alpha$	MP α_3, α_2
$\alpha_5)$	$\forall y.\neg\forall x.\neg\alpha$	Gen α_4

Logo, $\alpha_1, \ldots, \alpha_5$ é uma prova de $\neg\forall x.\neg\forall y.\alpha \vdash \forall y.\neg\forall x.\neg\alpha$. Como a única generalização usada na prova introduz a variável y, que não é livre na hipótese $\neg\forall x.\neg\forall y.\alpha$, podemos garantir, pelo teorema da dedução, que existe uma prova para $\vdash \neg\forall x.\neg\forall y.\alpha \rightarrow \forall y.\neg\forall x.\neg\alpha$.

g) $\vdash \forall x.\neg\alpha \rightarrow \neg\forall x.\alpha$

$\alpha_1)$	$\forall x.\alpha \rightarrow \neg\forall x.\neg\alpha$	esta proposição item e.
$\alpha_2)$	$(\forall x.\alpha \rightarrow \neg\forall x.\neg\alpha) \rightarrow (\forall x.\neg\alpha \rightarrow \neg\forall x.\alpha)$	proposição 4.3.3.f
$\alpha_3)$	$\forall x.\neg\alpha \rightarrow \neg\forall x.\alpha$	MP α_1, α_2

Retornando ao teorema da dedução, observe que o fato de termos uma prova que não satisfaça as premissas do teorema da dedução, não significa que não possa ser achada uma outra prova a qual sim as satisfaça.

Exemplo 7.2.13 *Seja a seguinte prova de* $P(x) \vdash \neg\forall x.\neg P(x)$

$\alpha_1)$	$P(x)$	*hip.*
$\alpha_2)$	$\forall x.P(x)$	Gen α_1
$\alpha_3)$	$\forall x.P(x) \rightarrow \neg\forall x.\neg P(x)$	*proposição 7.2.12.b*
$\alpha_4)$	$\neg\forall x.\neg P(x)$	MP α_2, α_3

Como $P(x)$ *tem variáveis livres,* Gen *foi aplicado na prova* (α_2)*, a variável introduzida em* α_2 (x) *é livre em* $P(x)$ *e* α_2 *depende de* $P(x)$*, então não podemos aplicar o teorema da dedução e portanto não podemos garantir a existência de uma prova para* $\vdash P(x) \rightarrow \neg\forall x.\neg P(x)$*.*

Embora esta prova seja muito natural, ela não é a única. Seja a seguinte prova de $P(x) \vdash \neg\forall x.\neg P(x)$

$\alpha_1)$	$\forall x.\neg P(x) \rightarrow \neg P(x)$	A_5
$\alpha_2)$	$(\forall x.\neg P(x) \rightarrow \neg P(x)) \rightarrow (P(x) \rightarrow \neg\forall x.\neg P(x))$	*proposição 4.3.3.e*
$\alpha_3)$	$P(x) \rightarrow \neg\forall x.\neg P(x)$	MP α_1, α_2
$\alpha_4)$	$P(x)$	*Hip.*
$\alpha_5)$	$\neg\forall x.\neg P(x)$	MP α_4, α_3

Como Gen *não foi usado na prova, então é possível aplicar o teorema da dedução e concluir que existe uma prova de* $\vdash P(x) \rightarrow \neg\forall x.\neg P(x)$*. Observe que poderíamos ter provado este teorema diretamente, isto é, sem necessidade de apelar ao teorema da dedução. De fato* $\alpha_1, \alpha_2, \alpha_3$*, na prova acima, é uma prova de* $\vdash P(x) \rightarrow \neg\forall x.\neg P(x)$*.*

7.3 Exercícios

1. Prove cada uma das seguintes fórmulas na teoria formal T^P:

   (a) $\forall x.\forall y.\forall z.\alpha \to \forall z.\forall y.\forall x.\alpha$

   (b) $\forall x_1.\cdots\forall x_n.\alpha \to \forall x_n.\forall x_1.\cdots\forall x_{n-1}.\alpha$

   (c) $\neg\forall x.\neg\neg\alpha \to \neg\forall x.\alpha$

   (d) $\neg\forall x.\forall y.\neg\alpha \to \neg\forall y.\forall x.\neg\alpha$

   (e) $\forall x.(\alpha \to \beta) \to (\neg\forall x.\neg\alpha \to \neg\forall x.\neg\beta)$

   (f) $\forall x.\neg(\alpha \to \neg\beta) \to \neg(\forall x.\alpha \to \neg\forall x.\beta)$

   (g) $\neg\forall x.\forall y.\neg\alpha \to \neg\forall y.\forall x.\neg\alpha$

2. Prove as seguintes deduções

   (a) $\vdash \forall x.\forall y.P(x,y) \to \forall x.\forall y.P(y,x)$

   (b) $P(x) \vdash R(a) \to \forall y.P(y)$

   (c) $P(x) \vdash (\forall x.P(x) \to \forall y.P(y)) \to P(a)$

   (d) $P(x) \vdash (\forall x.P(x) \to \forall y.R(y)) \to R(a)$

   (e) $\forall x.(P(x) \to R(x)) \vdash \neg\forall x.\neg P(x) \to \neg\forall x.\neg R(x)$

   (f) $R(a) \to \forall x.P(x) \vdash \forall x.(R(a) \to P(x))$

   (g) $\forall x.(P(x) \to R(a)) \vdash \neg\forall x.\neg P(x) \to R(a)$

   (h) $\neg\forall x.\neg P(x) \to R(a) \vdash \forall x.(P(x) \to R(a))$

   (i) $\neg\forall x.\neg\forall y.P(x,y) \vdash \forall y.\neg\forall x.\neg P(x,y)$

   (j) $\neg\forall x.\neg(P(x) \to R(x)) \vdash \forall x.P(x) \to \neg\forall x.\neg R(x)$

   (k) $\forall x.P(x) \to \neg\forall x.\neg R(x) \vdash \neg\forall x.\neg(P(x) \to R(x))$

   (l) $\vdash \neg P(y) \to (\forall x.\neg(P(x) \to \neg P(y)) \to P(y))$

3. Mostre que a inversa da proposição 7.1.6 não é verdadeira. Ou seja, se $\varphi(\alpha)$ é uma tautologia, então α não necessariamente é um teorema em T^P.

4. Diga porque não podemos aplicar o teorema da dedução nas seguintes provas

   (a)

$\alpha_1)\ \alpha$	hip.
$\alpha_2)\ \forall x.\alpha$	Gen, α_1
$\alpha_3)\ \forall x.\alpha \to (\beta \to \forall x.\alpha)$	A_1
$\alpha_4)\ \beta \to \forall x.\alpha$	MP, α_2, α_3

(b)

$\alpha_1)\ \forall x.(P(y) \to Q(x,y))$	hip.
$\alpha_2)\ \forall x.(P(y) \to Q(x,y)) \to (P(y) \to \forall x.Q(x,y))$	A_4
$\alpha_3)\ P(y) \to \forall x.Q(x,y)$	MP, α_1, α_2
$\alpha_4)\ \forall y.(P(y) \to \forall x.Q(x,y))$	Gen, α_3
$\alpha_5)\ \forall y.(P(y) \to \forall x.Q(x,y)) \to (P(z) \to \forall x.Q(x,z))$	A_5
$\alpha_6)\ P(z) \to \forall x.Q(x,z)$	MP, α_4, α_5

(c)

$\alpha_1)\ P(x)$	hip.
$\alpha_2)\ \forall x.P(x)$	Gen α_1
$\alpha_3)\ P(x) \to (Q(x) \to P(x))$	A_1
$\alpha_4)\ Q(x) \to P(x)$	MP, α_1, α_3

Observação: Esta última prova não nos permite aplicar o teorema da dedução, porém isto não significa que o teorema da dedução não seja aplicável a $P(x) \vdash Q(x) \to P(x)$. De fato se retiramos α_2., da prova acima, podemos aplicar o teorema da dedução.

5. Diga porque podemos aplicar o teorema da dedução nas seguintes provas (use os 5 critérios usados neste capítulo).

(a)

$\alpha_1)\ \alpha$	hip.
$\alpha_2)\ \forall x.(\alpha \to \beta)$	hip.
$\alpha_3)\ \forall x.(\alpha \to \beta) \to (\alpha \to \beta)$	A_5
$\alpha_4)\ \alpha \to \beta$	MP, α_2, α_3
$\alpha_5)\ \beta$	MP, α_1, α_4

(b)

$\alpha_1)\ P(x) \to \forall y.Q(x,y)$	hip.
$\alpha_2)\ \forall y.(P(x) \to \forall y.Q(x,y))$	Gen α_1
$\alpha_3)\ \forall y.(P(x) \to \forall y.Q(x,y)) \to (P(x) \to \forall y.\forall y.Q(x,y))$	A_4
$\alpha_4)\ P(x) \to \forall y.\forall y.Q(x,y)$	MP, α_2, α_3

Capítulo 8

Resolução na Lógica de Predicados

Neste capítulo descreveremos o esquema de computação para o cálculo de predicados, chamado *resolução geral*, este processo é essencialmente o mesmo que o descrito para o caso proposicional, ou seja, para provar que uma fórmula α é um teorema, primeiro negamos a fórmula, em seguida transformamos $\neg\alpha$ numa fórmula equivalente numa certa forma normal, e depois aplicamos o princípio de eliminação de literais complementares. Só que, no caso da lógica de predicados, por ter uma linguagem mais complexa, a transformação à forma normal vai ser mais intrincada, pois antes de eliminar dois literais, as vezes, teremos que unificá-los para torná-los complementares. Por exemplo, os literais $P(x)$ e $\neg P(a)$ não poderiam ser eliminados diretamente, pois $P(x) \neq P(a)$ e portanto não são complementares. No entanto, como x pode tomar qualquer valor, inclusive o valor que tome a constante a, podemos dizer que $P(a)$ é uma instância de $P(x)$. Agora sim, $P(a)$ e $\neg P(a)$ são complementares, e portanto podem ser eliminados. O processo de substituir x por a para tornar complementares $P(x)$ com $\neg P(a)$ se chama de unificação.

8.1 Forma Clausal

Quando descrevemos o processo de computação para o caso proposicional discutimos, então, a resolução "*básica*". A qual era baseada em cláusulas básicas mas não dizemos o porque do nome "básica" a essa definição de cláusula. Uma expressão ou cláusula é **básica** (*ground* em inglês) se ela não contém variáveis.

Na lógica proposicional só encontramos cláusulas básicas, uma vez que a linguagem não contém variáveis. Resolução básica por si só não é suficiente para manusear a linguagem de predicados, que é mais expressiva. Para LP usaremos uma forma mais geral de resolução, que consiste de unificação junto com eliminação do literal complementar (ELC). Usaremos, também, aqui, procedimentos de refutação, que consiste em provar um teorema mostrando que sua negação é contraditória.

Como introduziremos novos símbolos em LP, redefiniremos algumas noções já definidas anteriormente, adicionando noções novas.

1. termo :=constante | variável | símbolo funcional"(" lista de termos ")"
2. lista de termos :=termo| termo"," lista de termos
3. fórmula atômica :=símbolo de predicado"(" lista de termos")"
4. literal :=fórmula atômica | "¬" fórmula atômica
5. cláusula :="□" | literal | literal "∨" cláusula
6. sentença :={cláusula} | cláusula "∧" sentença

Definição 8.1.1 *Uma fbf de L^P está na* **forma clausal** *se ela é uma sentença como descrita na gramática acima.*

Da mesma forma que no caso proposicional, uma cláusula será vista desde duas óticas: como uma disjunção ("ou") de seus elementos (literais) e como um conjunto de literais. Analogamente, veremos uma sentença como uma conjunção ("e") de seus elementos (cláusulas) e como um conjunto de cláusulas.

Observe a ausência de quantificadores na definição de forma clausal. Sabemos que uma fbf de L^P é provável (isto é, existe uma prova para ela) se e somente se seu fecho é provável (pelo axioma A_5 e por Gen). Vimos que uma fbf é universalmente válida se e somente se seu fecho é universalmente válido. Assim, toda fbf na forma clausal está implicitamente fechada, pois consideraremos cada variável como estando universalmente quantificada sobre a cláusula em que ela aparece.

8.2 Forma Normal Prenex

Requereremos que as fbf's que usaremos estejam na **forma normal conjuntiva reduzida sem quantificadores**. Mas antes de definir quando uma sentença está nesta forma normal e como podemos transformar qualquer sentença numa fórmula equivalente na forma normal conjuntiva reduzida sem quantificadores, teremos que descrever alguns passos intermediários.

Uma fbf está na **forma normal prenex** se cada variável é quantificada, e todos os quantificadores estão juntos precedendo uma sentença livre de quantificadores.

Definição 8.2.1 *Uma fórmula está na* **forma normal prenex** *se tem a seguinte forma*

$$Q_1x_1.\cdots Q_nx_n.\alpha$$

onde α é uma fórmula livre de quantificadores, $x_1,\ldots,x_n$ são variáveis diferentes e para cada $1 \leq i \leq n$, $Q_i = \forall$ ou $Q_i = \exists$. Se todos os Q_i são quantificadores universais a fórmula é chamada de **universalmente fechada**. *$Q_1x_1.\cdots Q_nx_n$ é chamado de* **prefixo** *da fórmula e α de* **matriz**.

Por exemplo, as fórmulas $\forall x.\exists y.\forall z.(P(x,y) \to P(f(x,y),z))$ e $\forall x.\forall y.((P(x) \wedge Q(y)) \to (P(f(x,y)) \vee \neg Q(f(x,y))))$ estão ambas na forma normal prenex. Porém enquanto a segunda delas está universalmente fechada a primeira não.

Toda fbf do cálculo de predicado é equivalente a uma fbf na forma normal prenex com as mesmas variáveis livres. Para pôr uma fbf na forma normal prenex, movemos todos os quantificadores à esquerda dos conectivos lógicos $\neg$, $\wedge$, $\vee$ e $\to$. Se a fórmula contiver algum conectivo $\leftrightarrow$, então antes de proceder a este processo, devemos eliminar este conectivo da fórmula usando a redução:

$$\alpha \leftrightarrow \beta \rightarrowtail\!\!\!\!\!\rightarrow (\alpha \to \beta) \wedge (\beta \to \alpha)$$

Para mover cada quantificador à esquerda de cada tipo destes conectivos devemos usar uma das reduções descritas a seguir.

$$\begin{array}{rcl}
\neg\exists x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \forall x.\neg\alpha \\
\neg\forall x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \exists x.\neg\alpha \\
\forall x.\alpha \wedge \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\alpha[y/x] \wedge \beta) \\
\beta \wedge \forall x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\beta \wedge \alpha[y/x]) \\
\exists x.\alpha \wedge \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\alpha[y/x] \wedge \beta) \\
\beta \wedge \exists x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\beta \wedge \alpha[y/x]) \\
\forall x.\alpha \vee \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\alpha[y/x] \vee \beta) \\
\beta \vee \forall x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\beta \vee \alpha[y/x]) \\
\exists x.\alpha \vee \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\alpha[y/x] \vee \beta) \\
\beta \vee \exists x.\alpha & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\beta \vee \alpha[y/x]) \\
\forall x.\alpha \to \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\alpha[y/x] \to \beta) \\
\alpha \to \forall x.\beta & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\alpha \to \beta[y/x]) \\
\exists x.\alpha \to \beta & \rightarrowtail\!\!\!\!\!\rightarrow & \forall y.(\alpha[y/x] \to \beta) \\
\alpha \to \exists x.\beta & \rightarrowtail\!\!\!\!\!\rightarrow & \exists y.(\alpha \to \beta[y/x])
\end{array}$$

onde y é uma variável nova.

Observe que a renomeação de todas as variáveis é para se garantir que todas as variáveis diferentes tenham distintos nomes. Se não tivermos este cuidado, pode acontecer que a fórmula resultante não seja equivalente à original. Por exemplo, no caso da fórmula $\forall x.P(x) \wedge R(a,x)$, se aplicarmos a redução $\forall x.\alpha \wedge \beta \rightarrowtail\!\!\!\!\!\rightarrow \forall y.(\alpha[y/x] \wedge \beta)$, obtemos a fórmula $\forall y.(P(y) \wedge R(a,x))$, onde x permanece livre na subfórmula $R(a,x)$. Mas, senão tivéssemos realizado a renomeação de variáveis, teríamos obtido a fórmula $\forall x.(P(x) \wedge R(a,x))$, onde a ocorrência da variável x em $R(a,x)$, que antes era livre, passaria a ser ligada, alterando completamente o sentido da fórmula e portanto transformando a fórmula original numa fórmula não equivalente.

Exemplo 8.2.2 *Seja a fórmula $\neg(\neg\forall x.\forall y.P(x,y) \vee \forall y.\forall x.P(x,y))$. Uma seqüência de reduções para transformar esta fórmula numa fórmula na forma normal prenex equivalente é a seguinte:*

$$
\begin{aligned}
\neg(\neg\forall x.\forall y.P(x,y) \vee \forall y.\forall x.P(x,y)) &\longrightarrow\!\!\!\!\!\rightarrow \neg(\exists x.\neg\forall y.P(x,y) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg(\exists x.\exists y.\neg P(x,y) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\exists x_1.(\exists y.\neg P(x_1,y) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\exists x_1.\exists y_1.(\neg P(x_1,y_1) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\exists x_1.\exists y_1.\forall y_2.(\neg P(x_1,y_1) \vee \forall x.P(x,y_2)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\exists x_1.\exists y_1.\forall y_2.\forall x_2.(\neg P(x_1,y_1) \vee P(x_2,y_2)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \forall x_1.\neg\exists y_1.\forall y_2.\forall x_2.(\neg P(x_1,y_1) \vee P(x_2,y_2)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \forall x_1.\forall y_1.\neg\forall y_2.\forall x_2.(\neg P(x_1,y_1) \vee P(x_2,y_2)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \forall x_1.\forall y_1.\exists y_2.\neg\forall x_2.(\neg P(x_1,y_1) \vee P(x_2,y_2)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \forall x_1.\forall y_1.\exists y_2.\exists x_2.\neg(\neg P(x_1,y_1) \vee P(x_2,y_2))
\end{aligned}
$$

A forma normal prenex assim obtida não é necessariamente única, mas é possível garantir sua equivalência com a fórmula original. Por exemplo, no caso da fórmula do exemplo anterior, poderíamos também ter realizado a seguinte transformação:

$$
\begin{aligned}
\neg(\neg\forall x.\forall y.P(x,y) \vee \forall y.\forall x.P(x,y)) &\longrightarrow\!\!\!\!\!\rightarrow \neg(\exists x.\neg\forall y.P(x,y) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg(\exists x.\exists y.\neg P(x,y) \vee \forall y.\forall x.P(x,y)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\forall y_1.(\exists x.\exists y.\neg P(x,y) \vee \forall x.P(x,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\forall y_1.\forall x_1.(\exists x.\exists y.\neg P(x,y) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\forall y_1.\forall x_1.\exists x_2.(\exists y.\neg P(x_2,y) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \neg\forall y_1.\forall x_1.\exists x_2.\exists y_2.(\neg P(x_2,y_2) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \exists y_1.\neg\forall x_1.\exists x_2.\exists y_2.(\neg P(x_2,y_2) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \exists y_1.\exists x_1.\neg\exists x_2.\exists y_2.(\neg P(x_2,y_2) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \exists y_1.\exists x_1.\forall x_2.\neg\exists y_2.(\neg P(x_2,y_2) \vee P(x_1,y_1)) \\
&\longrightarrow\!\!\!\!\!\rightarrow \exists y_1.\exists x_1.\forall x_2.\forall y_2.\neg(\neg P(x_2,y_2) \vee P(x_1,y_1))
\end{aligned}
$$

Mas ambas são logicamente equivalentes à fórmula original. Generalizando:

Teorema 8.2.3 (Forma Normal Prenex) *Qualquer fórmula na linguagem de predicados, existe uma fórmula na forma normal prenex logicamente equivalente a ela e com as mesmas variáveis livres.*

Demonstração: Veja [Men87]. ■

Note que se em vez de x_2 e y_2 tivéssemos deixado as variáveis originais x e y, respectivamente, as fórmulas resultante seriam equivalentes. Assim, de agora em diante só renomearemos as variáveis quando necessário.

Depois de colocar todos os quantificadores ao lado esquerdo da fórmula, formamos o **fecho existencial** da fbf introduzindo, à esquerda, quantificadores existenciais para cada variável livre da fórmula afim de fechar a fórmula. Assim, pelo teorema **??** e pela terceira observação ao final do capítulo 6, a fórmula original é insatisfatível se e somente se o fecho existencial de sua forma normal prenex também é insatisfatível.

Exemplo 8.2.4 *Seja a fórmula*

$$\neg\forall x.(P(x,z) \wedge R(x,y)) \vee (\exists x.P(x,z) \vee \forall x.R(x,y))$$

Uma fórmula equivalente na forma normal prenex, pode ser obtida da seguinte maneira: Movendo $\forall$ à esquerda de $\neg$ e renomeando variáveis quando necessário obtemos

$$\exists x.\neg(P(x,z) \wedge R(x,y)) \vee \exists x_1.\forall x_2.(P(x_1,z) \vee R(x_2,y)$$

Pondo os quantificadores na frente, temos

$$\exists x.\exists x_1.\forall x_2.(\neg(P(x,z) \wedge R(x,y)) \vee (P(x_1,z) \vee R(x_2,y)))$$

e formando o fecho existencial temos

$$\exists z.\exists y.\exists x.\exists x_1.\forall x_2.(\neg(P(x,z) \wedge R(x,y)) \vee (P(x_1,z) \vee R(x_2,y)))$$

Às vezes é mais fácil de visualizar este processo quando mostrado graficamente.

Poderemos representar uma fbf por uma árvore cujas folhas são fórmulas atômicas e cujos nós não folhas contém ou um conectivo lógico ($\neg$, $\wedge$, $\vee$) ou uma quantificação ($\forall x$, $\exists x$). Descendentes de um nó representam as subfórmulas para as quais o conectivo ou quantificação se aplica. Qualquer quantificação que está mais acima na árvore (não tem quantificações no caminho dela até o prefixo da fbf na raíz) pode ser movida acima da árvore para abaixo do prefixo. Neste processo as variáveis podem ser renomeadas e os quantificadores podem mudar de tipo, dependendo do tipo de redução empregada. Por exemplo, a s redução $\neg\forall x\alpha \longrightarrow\!\!\!\!\rightarrow exists x\neg\alpha$ tem o efeito mostrado na figura 8.1. Já a redução $\exists x.\alpha \vee \beta \longrightarrow\!\!\!\!\rightarrow \exists y.(\alpha[y/x] \vee \beta)$ tem o efeito ilustrado na figura 8.2.

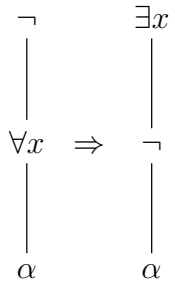


Figura 8.1: Efeito na árvore da segunda redução

Considere a mesma fórmula do exemplo 8.2.4. Uma representação gráfica do processo de transformação à forma normal prenex é mostrada nas árvores das figuras 8.3 a 8.7. A

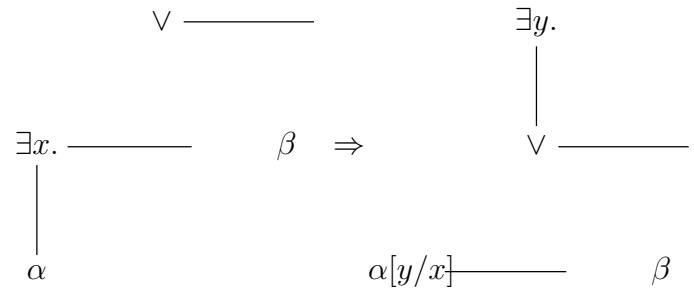


Figura 8.2: Efeito na árvore da nona redução

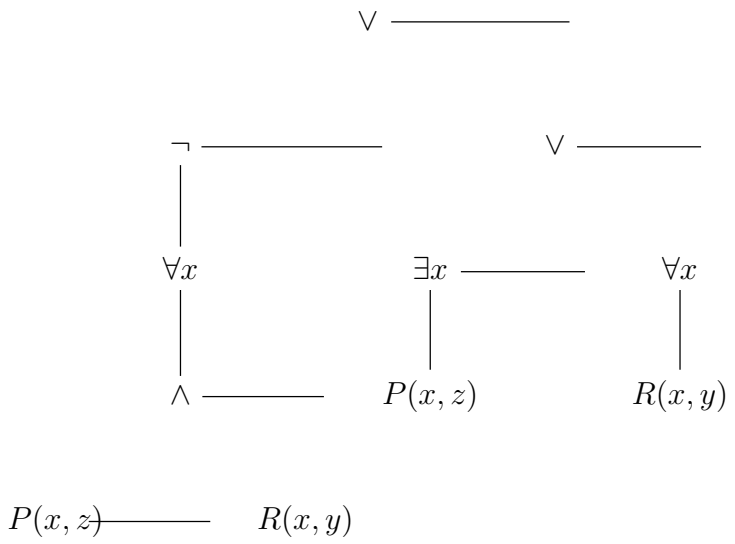


Figura 8.3: Árvore da fórmula $\neg\forall x.(P(x,z) \wedge R(x,y)) \vee (\exists x.P(x,z) \vee \forall x.R(x,y))$

figura 8.3 apresenta a fórmula do exemplo 8.2.4 na forma de árvore. A figura 8.4 mostra a transformação de um quantificador universal num existencial pela ação da negação. A posição original da quantificação na árvore é tomada por seu nó descendente.

Agora vamos deslocando sucessivamente para o topo da árvore as quantificações, como mostram as árvores das figuras 8.5, 8.6 e 8.7, mas devemos ter cuidado com o perigo de capturar variáveis livres. Se uma quantificação é movida para uma posição na qual governa ocorrências de uma variável que anteriormente era livre, estas ocorrências serão capturadas e portanto, para não mudar o sentido da fórmula, deveremos renomear essas ocorrências.

Note que a renomeação de x por x_1 não é necessária, mas como existem diversas ocorrências da variável x ligadas a distintos quantificadores, então se obtém uma melhor visualização de qual x está ligado a que quantificador através um subíndice em cada um deles.

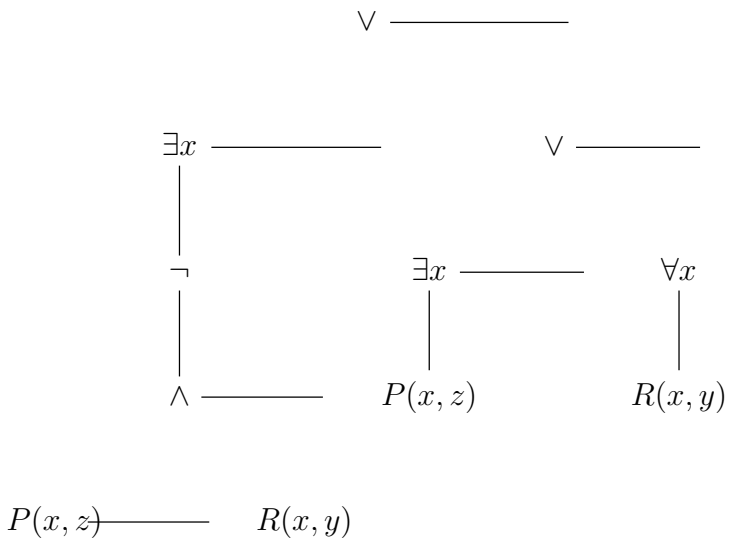


Figura 8.4: Deslocamento do quantificador universal para acima da negação, transformando-o num quantificador existencial.

Finalmente, realizando o fecho existencial da fbf, obtemos a árvore da figura 8.8.

Note que a fórmula resultante desta árvore ($\exists z.\exists y.\exists x_1.\exists x_2.\forall x_3.(\neg(P(x_1,z)\wedge R(x_1,y))\vee (P(x_1,z)\vee R(x_2,y))))$ é diferente da resultante do exemplo 8.2.4. Porém como esta diferença somente reside nos nomes dados às variáveis, ambas as fórmulas são essencialmente as mesmas e portanto são equivalentes.

8.3 Forma Normal de Skolem

Se declaramos que para cada x existe um y tal que alguma propriedade é verdadeira, então estamos afirmando que podemos encontrar tal y, mas sua escolha pode depender do valor de x, e portanto a escolha de y pode ser vista como uma função sobre x. Por exemplo, se dizermos que "para todo número natural x existe um número y tal que y é menor ou igual que x" então o valor que venha tomar y está em função do valor de x, assim uma função que acha esse número y dado um x qualquer poderia ser $f(x) = x$ ou ainda $f(x) = \lfloor \frac{x}{2} \rfloor$. Porém, quando temos uma fórmula do tipo $\forall x.\exists y.\alpha$, a escolha da função é contingente à interpretação da fórmula.

O lógico norueguês, Thoralf Albert Skolem (1887–1963), para se livrar desta arapuca semântica, simplesmente atribuiu um símbolo de função arbitrário para representar esta escolha. Por isso este tipo de função hoje é conhecida como **função de Skolem**.

No exemplo "existe um número y tal que para todo número natural x, y é menor ou igual que x" o valor de y não depende do valor de x, de fato a única possibilidade de se satisfazer esta afirmação é y tomando o valor constante 0, pois 0 é menor ou igual

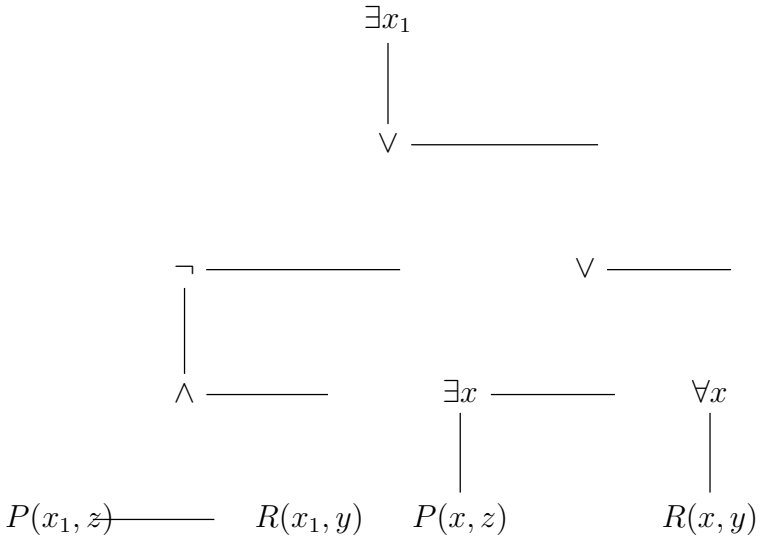


Figura 8.5: Deslocamento da quantificação mais próximo da raiz para o topo da árvore.

que qualquer número natural x. Assim, a ordem em que vêm os quantificadores pode ser transcendental não só do ponto de vista de interpretação, mas também do ponto de vista de dependência da variável quantificada existencialmente. Por outro lado, observe que a ordem entre um quantificador universal com um existencial é irrelevante quando a variável de um deles não está em função da variável do outro, ou mais precisamente quando não há um predicado relacionando ambas variáveis. Por exemplo, na afirmação, "para todo corvo preto existe um homem com seis dedos" a escolha do homem com seis dedos não depende do corvo preto e portanto diferentes corvos pretos podem ter associados o mesmo homem de seis dedos. Logo essa afirmação só será falsa se não existir homem com seis dedos, caso ele exista ela sera verdadeira. Assim, essa afirmação é equivalente à afirmação "existe um homem com seis dedos para todo corvo preto".

Seja α uma fórmula fechada na forma normal prenex. Podemos, agora, eliminar todas as quantificações existenciais em α. Cada variável quantificada existencialmente é substituída por uma função de Skolem sob as variáveis universalmente quantificadas que governam esta, ou seja as variáveis universalmente quantificadas que precedem o quantificador existencial no prefixo da fórmula e que participam de uma mesma fórmula atômica com a variável quantificada existencialmente. Uma função 0-ária (sem argumentos) é vista como uma constante. Assim, se uma variável quantificada existencialmente não é governada por qualquer variável quantificada universalmente então ela deve ser substituída por uma constante. Este processo de eliminação de quantificadores existenciais em fórmulas fechadas na forma normal prenex é chamado de **skolemização**. Tanto os símbolos de função como as constantes introduzidas na skolemização devem ser novos na fórmula. Seja α uma fórmula. A **forma normal de Skolem** de α é a fórmula resultante da skolemização da forma normal prenex de α.

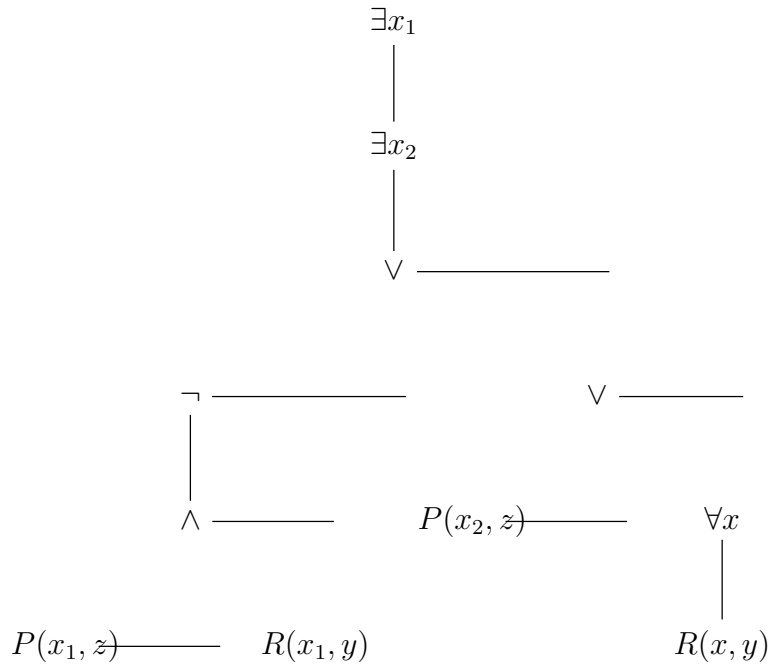


Figura 8.6: Deslocação das quantificações para o topo da árvore

Exemplo 8.3.1 *Seja a fórmula na forma normal prenex* $\forall x.\exists y.(P(x) \rightarrow P(y))$. *Embora a quantificação* $\forall x$ *precede a quantificação* $\exists y$, *não temos nenhum caso onde as variáveis* y *e* x *façam parte da mesma fórmula atômica e portanto o valor que venha tomar* x *não influência no valor de* y. *Logo, o* y *é substituído por uma constante e não por uma função em* x. *Assim, a fórmula resultante da skolemização é:*

$$\forall x.(P(x) \rightarrow P(f(x))).$$

Já no caso da fórmula

$$\forall y.\exists z.\forall x.(P(x,y) \rightarrow (R(x,z) \vee R(z,y)))$$

a variável z *esta quantificada existencialmente e está precedida por uma quantificação universal sobre a variável* y. *Como na fórmula original tem-se a subfórmula* $R(z,y)$, *então podemos concluir que a variável* z *está em função da variável* y, *logo substituímos todas as ocorrências de* z *por* $f(y)$. *Assim a forma normal de Skolem desta fórmula é a seguinte:*

$$\forall y.\forall x.(P(x,y) \rightarrow (R(x,f(y)) \vee R(f(y),y))).$$

Exemplo 8.3.2 *Considere a fórmula* α

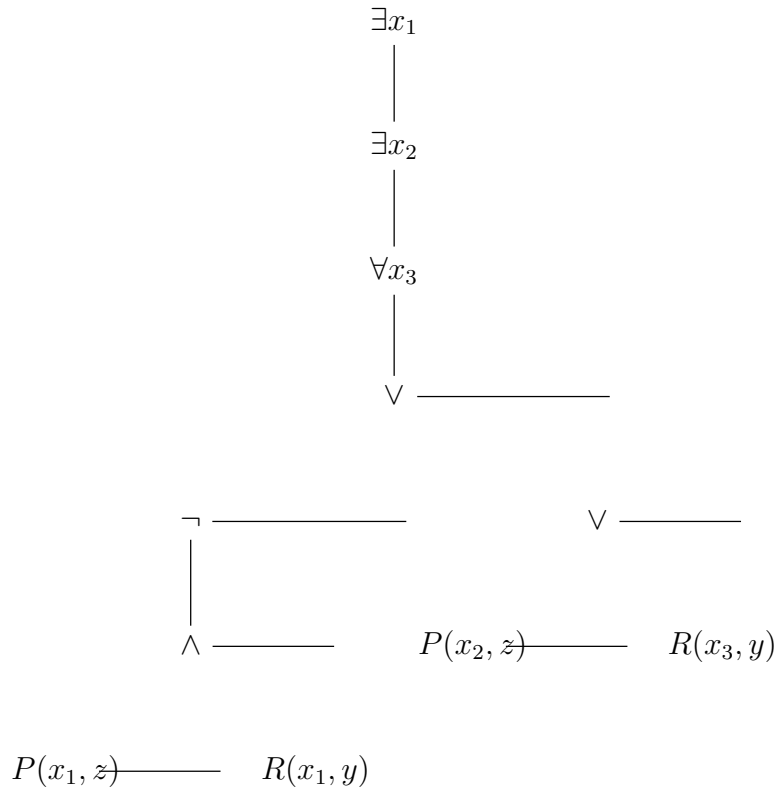


Figura 8.7: Árvore para a forma normal prenex da fórmula $\neg\forall x.(P(x,z) \wedge R(x,y)) \vee (\exists x.P(x,z) \vee \forall x.R(x,y))$.

$$\exists x_1.\forall x_2.\forall x_3.\exists x_4.R(x_1,x_2,x_3,x_4)$$

Eliminando quantificadores existenciais em α, obtemos a fórmula

$$\alpha_1 = \forall x_2.\forall x_3.R(a,x_2,x_3,f(x_2,x_3))$$

Exemplo 8.3.3 *Seja α a fórmula*

$$\forall x_1.\exists x_2.\exists x_3.\forall x_4.\exists x_5.(P(x_1,f(x_2),x_3) \wedge \neg P(x_2,x_4,x_5)).$$

Como a variável x_2 é existencialmente quantificada, é precedida pela quantificação universal $\forall x_1$ e x_1 e x_2 fazem parte de um mesma subfórmula atômica ($P(x_1,f(x_2),x_3)$) então no processo de skolemização x_2 deve ser substituída pelo termo $g_1(x_1)$. Analogamente, com x_3 a qual deve ser substituída pelo termo $g_2(x_1)$. No caso de x_5, podemos observar que $\exists x_5$ é precedido por dois quantificações universais ($\forall x_1$ e $\forall x_4$), mas somente a variável x_4 faz parte junto com x_5 de uma mesma subfórmula atômica. Logo, x_5 deve ser substituído pelo termo $g_3(x_4)$. Assim, como resultado da skolemização obtemos a fórmula

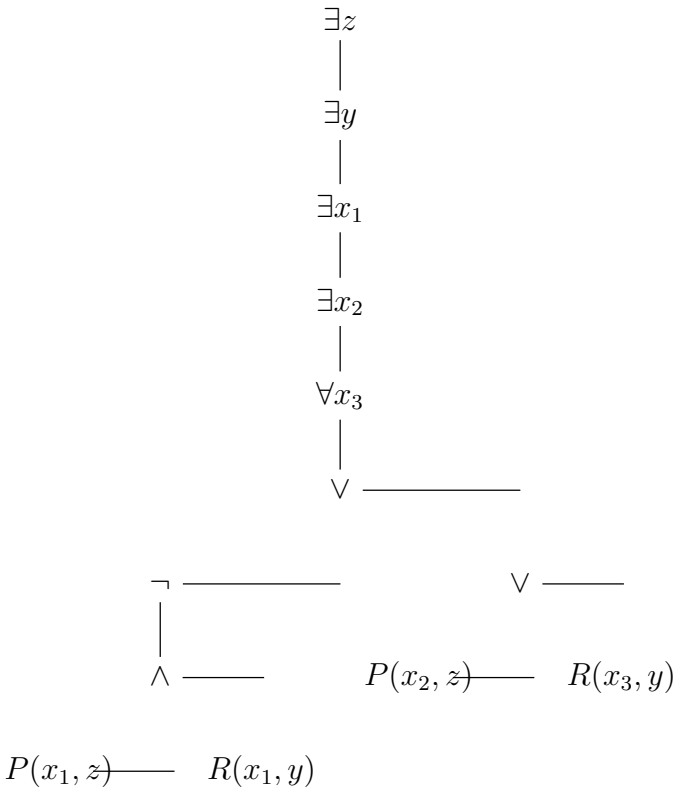


Figura 8.8: Árvore do fecho universal da fórmula $\exists x_1.\exists x_2.\forall x_3.(\neg(P(x_1,z) \wedge R(x_1,y)) \vee (P(x_1,z) \vee R(x_2,y)))$.

$$\forall x_1.\forall x_4.(P(x_1, f(g_1(x_1)), g_2(x_1)) \wedge \neg P(g_1(x_1), x_4, g_3(x_4)))$$

Como resultado da skolemização de uma fórmula obteremos uma fórmula universalmente fechada que não necessariamente é equivalente à original, mas que têm uma estreita relação.

Proposição 8.3.4 *Para cada fórmula na forma normal prenex existe uma fórmula fechada universalmente (contendo geralmente novas constantes e símbolos de função) que é satisfatível se, e somente se, a fórmula original também é satisfatível.*

Demonstração: A eliminação dos quantificadores existenciais fazendo skolemização leva a uma fórmula fechada universalmente que é satisfatível se e somente se a fórmula original é satisfatível. [Doe94]. ■

Portanto se a fórmula na forma normal prenex original é universalmente válida o único que podemos afirmar da fórmula resultante após o processo de skolemização é que ela é

satisfatível (pode ser universalmente válida ou não, mas nunca uma contradição). Porém se a fórmula na forma normal prenex original é uma contradição a fórmula resultante após o processo de skolemização necessariamente é uma contradição, pois ela é insatisfatível. Este resultado será essencial para optarmos por um método de refutação para provar teoremas, ou seja quando queremos mostrar que uma determinada fórmula é universalmente válida, o mostramos provando que a negação dela leva a um absurdo, nessa prova usamos a fórmula resultante do processo de skolemização.

Corolário 8.3.5 *Para cada fórmula α na forma normal prenex, existe uma fórmula na forma normal de Skolem que é uma contradição se, e somente se, a fórmula α também é uma contradição.*

Demonstração: Direto da proposição 8.3.4. ■

Assim, a skolemização preserva insatisfatibilidade de uma fórmula.

8.4 Forma Clausal

Uma fórmula fechada universalmente está na **forma clausal** se estiver na forma normal prenex e sua matriz estiver na forma normal conjuntiva reduzida. Assim, a forma clausal de uma fórmula α é a forma normal conjuntiva reduzida da fórmula obtida usando as reduções da seção **??** na matriz da forma normal de Skolem de α. Como foi usado o mesmo algoritmo para obter a forma normal conjuntiva reduzida de uma fórmula proposicional, segue imediatamente o seguinte resultado.

Proposição 8.4.1 *Seja α uma fórmula em L^P. Se α está na forma normal de Skolem então α e sua forma clausal são equivalentes.*

Demonstração: Direto da proposição 5.1.4. ■

Assim, qualquer fórmula α de L^P pode ser transformada numa fórmula β de L^P na forma clausal tal que α é insatisfatível se, e somente se, β é insatisfatível. Para isto, devemos realizar os seguintes passos:

1. transformar a fórmula a sua forma normal prenex
2. realizar o fecho existencial da fórmula
3. eliminar quantificadores existenciais aplicando o processo de skolemização
4. ocultar todos os quantificadores universais

5. transformar à forma normal conjuntiva reduzida usando o algoritmo da proposição 5.1.4 na matriz da fórmula

Exemplo 8.4.2 *Seja α a seguinte fórmula*

$$\neg(\exists x_1.P(x_1,x_2) \to \neg\forall x_3.(P(x_3,x_4) \wedge \exists x_5.R(x_5))).$$

Transformaremos α numa fórmula na forma clausal seguindo os cinco passos mencionados.

Passo 1: Transformar à forma normal prenex

$$\begin{aligned}
\alpha &\twoheadrightarrow \neg(\exists x_1.P(x_1,x_2) \to \exists x_3.\neg(P(x_3,x_4) \wedge \exists x_5.R(x_5))) \\
&\twoheadrightarrow \neg(\exists x_1.P(x_1,x_2) \to \exists x_3.\neg\exists x_5.(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \neg(\exists x_1.P(x_1,x_2) \to \exists x_3.\forall x_5.\neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \neg\forall x_1.(P(x_1,x_2) \to \exists x_3.\forall x_5.\neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \neg\forall x_1.\exists x_3.(P(x_1,x_2) \to \forall x_5.\neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \neg\forall x_1.\exists x_3.\forall x_5.(P(x_1,x_2) \to \neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \exists x_1.\neg\exists x_3.\forall x_5.(P(x_1,x_2) \to \neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \exists x_1.\forall x_3.\neg\forall x_5.(P(x_1,x_2) \to \neg(P(x_3,x_4) \wedge R(x_5))) \\
&\twoheadrightarrow \exists x_1.\forall x_3.\exists x_5.\neg(P(x_1,x_2) \to \neg(P(x_3,x_4) \wedge R(x_5)))
\end{aligned}$$

Passo 2: realizar o fecho existencial da fórmula

$$\alpha_2 \stackrel{\text{def}}{=} \exists x_2.\exists x_4.\exists x_1.\forall x_3.\exists x_5.\neg(P(x_1,x_2) \to \neg(P(x_3,x_4) \wedge R(x_5)))$$

Passo 3: eliminar quantificadores existenciais aplicando o processo de skolemização

$$\alpha_3 \stackrel{\text{def}}{=} \forall x_3.\neg(P(a,b) \to \neg(P(x_3,c) \wedge R(f(x_3))))$$

Passo 4: ocultar todos os quantificadores universais

$$\alpha_4 \stackrel{\text{def}}{=} \neg(P(a,b) \to \neg(P(x_3,c) \wedge R(f(x_3))))$$

Passo 5: transformar à forma normal conjuntiva reduzida usando o algoritmo da proposição 5.1.4

$$\begin{aligned}
\alpha_4 &\twoheadrightarrow \neg(\neg(P(a,b) \to \neg(P(x_3,c) \wedge R(f(x_3))))) \\
&\twoheadrightarrow \neg\neg P(a,b) \wedge \neg\neg(P(x_3,c) \wedge R(f(x_3))) \\
&\twoheadrightarrow P(a,b) \wedge \neg\neg(P(x_3,c) \wedge R(f(x_3))) \\
&\twoheadrightarrow P(a,b) \wedge P(x_3,c) \wedge R(f(x_3))
\end{aligned}$$

Como todos os passos acima preservam insatisfabilidade, então se uma fórmula α for universalmente válida, então a forma clausal de $\neg\alpha$ seguindo os passos acima, será uma contradição.

Exemplo 8.4.3 *A fórmula*

$$\begin{array}{rl}\alpha \stackrel{\text{def}}{=} & \neg((\forall x.(P(x,z) \vee R(z,x)) \rightarrow \forall x.(P(x,x) \wedge \neg R(z,z))) \rightarrow \\ & \forall x.((P(x,z) \vee R(z,x)) \rightarrow (P(x,x) \wedge \neg R(z,z))))\end{array}$$

é a negação de uma instância da proposição 7.2.12.b e portanto é uma contradição.

1. *Forma normal prenex de α:*

$$\begin{array}{rl}\alpha & \rightarrowtail\!\!\!\!\!\rightarrow \neg(\exists x_1.((P(x_1,z) \vee R(z,x_1)) \rightarrow \forall x.(P(x,x) \wedge \neg R(z,z))) \rightarrow \\ & \quad \forall x.((P(x,z) \vee R(z,x)) \rightarrow (P(x,x) \wedge \neg R(z,z)))) \\ & \rightarrowtail\!\!\!\!\!\rightarrow \neg(\exists x_1.\forall x_2.((P(x_1,z) \vee R(z,x_1)) \rightarrow (P(x_2,x_2) \wedge \neg R(z,z))) \rightarrow \\ & \quad \forall x.((P(x,z) \vee R(z,x)) \rightarrow (P(x,x) \wedge \neg R(z,z)))) \\ & \rightarrowtail\!\!\!\!\!\rightarrow \neg\forall x_1.\exists x_2.(((P(x_1,z) \vee R(z,x_1)) \rightarrow (P(x_2,x_2) \wedge \neg R(z,z))) \rightarrow \\ & \quad \forall x.((P(x,z) \vee R(z,x)) \rightarrow (P(x,x) \wedge \neg R(z,z)))) \\ & \rightarrowtail\!\!\!\!\!\rightarrow \neg\forall x_1.\exists x_2.\forall x_3.(((P(x_1,z) \vee R(z,x_1)) \rightarrow (P(x_2,x_2) \wedge \neg R(z,z))) \rightarrow \\ & \quad ((P(x_3,z) \vee R(z,x_3)) \rightarrow (P(x_3,x_3) \wedge \neg R(z,z)))) \\ & \rightarrowtail\!\!\!\!\!\rightarrow \exists x_1.\forall x_2.\exists x_3.\neg(((P(x_1,z) \vee R(z,x_1)) \rightarrow (P(x_2,x_2) \wedge \neg R(z,z))) \rightarrow \\ & \quad ((P(x_3,z) \vee R(z,x_3)) \rightarrow (P(x_3,x_3) \wedge \neg R(z,z))))\end{array}$$

   A qual, pelo teorema 8.2.3 é uma contradição, uma vez que ambas fórmulas são equivalentes.

2. *Fecho existencial:*

$$\begin{array}{rl}\exists z.\exists x_1.\forall x_2.\exists x_3. & \neg(((P(x_1,z) \vee R(z,x_1)) \rightarrow (P(x_2,x_2) \wedge \neg R(z,z))) \rightarrow \\ & ((P(x_3,z) \vee R(z,x_3)) \rightarrow (P(x_3,x_3) \wedge \neg R(z,z))))\end{array}$$

   Que também é um contradição, pois fecho existencial preserva contradições (observação 1.2 ao final da seção 6.5).

3. *Skolemização:*

$$\begin{array}{rl}\forall x_2. & \neg(((P(b,a) \vee R(a,b)) \rightarrow (P(x_2,x_2) \wedge \neg R(a,a))) \rightarrow \\ & ((P(f(x_2),a) \vee R(a,f(x_2))) \rightarrow (P(f(x_2),f(x_2)) \wedge \neg R(a,a))))\end{array}$$

   Que, pelo corolário 8.3.5, é uma contradição.

4. *Ocultando os quantificadores universais:*

$$\begin{array}{rl}\beta \stackrel{\text{def}}{=} & \neg(((P(b,a) \vee R(a,b)) \rightarrow (P(x_2,x_2) \wedge \neg R(a,a))) \rightarrow \\ & ((P(f(x_2),a) \vee R(a,f(x_2))) \rightarrow (P(f(x_2),f(x_2)) \wedge \neg R(a,a))))\end{array}$$

   Aqui os quantificadores universais são apenas escondidos, eles ainda quantificam a fórmula, só não aparecem explicitamente para simplificar o resto do processo. Portanto a fórmula β é exatamente a mesma resultante da skolemização.

5. *Forma clausal:*

$$
\begin{aligned}
\beta \quad &\rightarrowtail\!\!\!\!\!\rightarrow \neg((\neg(P(b,a) \vee R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \rightarrow \\
&\qquad (\neg(P(f(x_2),a) \vee R(a,f(x_2))) \vee (P(f(x_2),f(x_2)) \wedge \neg R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow \neg(\neg(\neg(P(b,a) \vee R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \vee \\
&\qquad (\neg(P(f(x_2),a) \vee R(a,f(x_2))) \vee (P(f(x_2),f(x_2)) \wedge \neg R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow \neg\neg(\neg(P(b,a) \vee R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \wedge \\
&\qquad \neg(\neg(P(f(x_2),a) \vee R(a,f(x_2))) \vee (P(f(x_2),f(x_2)) \wedge \neg R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow (\neg(P(b,a) \vee R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \wedge \\
&\qquad (\neg\neg(P(f(x_2),a) \vee R(a,f(x_2))) \wedge \neg(P(f(x_2),f(x_2)) \wedge \neg R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow (\neg(P(b,a) \vee R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \wedge \\
&\qquad ((P(f(x_2),a) \vee R(a,f(x_2))) \wedge (\neg P(f(x_2),f(x_2)) \vee \neg\neg R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow ((\neg P(b,a) \wedge \neg R(a,b)) \vee (P(x_2,x_2) \wedge \neg R(a,a))) \wedge \\
&\qquad ((P(f(x_2),a) \vee R(a,f(x_2))) \wedge (\neg P(f(x_2),f(x_2)) \vee R(a,a)))) \\
&\rightarrowtail\!\!\!\!\!\rightarrow ((\neg P(b,a) \vee P(x_2,x_2)) \wedge (\neg P(b,a) \vee \neg R(a,a)) \wedge (\neg R(a,b) \vee P(x_2,x_2))) \wedge \\
&\qquad (\neg R(a,b) \vee \neg R(a,a))) \wedge (P(f(x_2),a) \vee R(a,f(x_2))) \wedge \\
&\qquad (\neg P(f(x_2),f(x_2)) \vee R(a,a))
\end{aligned}
$$

Que é uma contradição uma vez que β é uma contradição e que pela proposição 8.4.1 ambas as fórmulas são equivalentes.

8.5 Domínio de Herbrand

Seja Γ um conjunto de fórmulas em $Ling(\mathcal{L})$ e α uma fórmula em $Ling(\mathcal{L})$ tais que $\Gamma \models \alpha$. Assim, pela definição de conseqüência lógica (definição 6.5.14, se $\langle \mathcal{D}, \rho \rangle$ é um modelo de Γ, então $\langle \mathcal{D}, \rho \rangle$ é um modelo de α. Logo, nenhum modelo de Γ pode ser um modelo de $\neg\alpha$ e, portanto, nenhum modelo pode satisfazer $\Gamma \cup \{\neg\alpha\}$, ou seja $\Gamma \cup \{\neg\alpha\}$ é insatisfatível. Assim, se $\Gamma \models \alpha$ então $\Gamma \cup \{\neg\alpha\}$ é insatisfatível. Mas, um conjunto de cláusulas é insatisfatível se e somente se é falsa em qualquer interpretação e atribuição de valores a suas variáveis livres.

Uma vez que não é possível considerar todas as interpretações para um conjunto de fórmulas Γ nem todos as possíveis atribuições (em domínios infinitos), um domínio particular H será definido. Este domínio terá a propriedade que Γ é insatisfatível se e somente se é falso em todas as interpretações de Γ no domínio H. Este domínio particular é chamado de domínio de Herbrand de Γ e as interpretações são chamadas de expansões de Herbrand. O nome é devido a lógico frances Jacques Herbrand (1908–1931), quem a pesar de sua curta vida foi quem introduziu estes conceitos e provou a propriedade descrita acima.

Definição 8.5.1 *Seja α uma fbf. O* **domínio de Herbrand** *de α é o conjunto de termos $\mathcal{H}_\alpha$ obtido a partir de α usando as seguintes regras:*

1. *Se a é uma constante em α então $a \in \mathcal{H}_\alpha$. Se α não contém nenhuma constante, então um símbolo de constante arbitrário a é incluído em $\mathcal{H}_\alpha$.*

2. *Se $t_1, \ldots, t_n$ são termos em $\mathcal{H}_\alpha$, e f é um símbolo de função n-ária em α então $f(t_1, \ldots, t_n) \in \mathcal{H}_\alpha$. Se nenhum símbolo de função ocorre em α então um símbolo de função f de aridade hum arbitrário é considerado.*

Usamos a notação $T((\Gamma))$ para denotar o conjunto de instâncias obtidas ao aplicar no conjunto de fbf's, Γ, todas as possíveis combinações de substituir as variáveis livres das fórmulas em Γ por termos pertencentes a um conjunto T de termos.

Definição 8.5.2 *Seja α uma fbf. A* **expansão de Herbrand** *de α, denotado por $\mathcal{H}((\alpha))$, é o conjunto de todas as instâncias de α, obtidas ao substituir cada variável livre em α por um termo de $\mathcal{H}_\alpha$.*

Note que pela forma como foi construído o conjunto $\mathcal{H}_\alpha$, todos seus termos são livres para qualquer variável em α.

Exemplo 8.5.3 *Seja $\alpha = R(a, x_2, f(x_2), x_4)$. O domínio de Herbrand para α é dado pelo seguinte conjunto*

$$\mathcal{H}_\alpha = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}.$$

Portanto, a expansão de Herbrand de α é descrita na terceira coluna da seguinte tabela.

x_2	x_4	$R(a, x_2, f(x_2), x_4)$
a	a	$R(a, a, f(a), a)$
a	$f(a)$	$R(a, a, f(a), f(a))$
$f(a)$	a	$R(a, f(a), f(f(a)), a)$
$f(a)$	$f(a)$	$R(a, f(a), f(f(a)), f(a))$
a	$f(f(a))$	$R(a, a, f(a), f(f(a)))$
$\vdots$	$\vdots$	$\vdots$

onde x_2 e x_4 são substituídos em todas as combinações possíveis pelos termos em $\mathcal{H}_\alpha$, com cada variável sendo substituídas em todas suas ocorrências pelo mesmo termo.

Teorema 8.5.4 *[Teorema de Herbrand] Seja α uma fórmula clausal. α é insatisfatível se, e somente se, algum subconjunto finito de $\mathcal{H}((\alpha))$ é inconsistente.*

Demonstração: Veja [Bun83]. ■

Observe os seguintes pontos:

1. Se α é universalmente válida então α é satisfatível.

2. α é satisfatível se e somente se existe uma interpretação $\mathcal{D}$, no caso $\mathcal{H}_\alpha$, e uma atribuição ρ tal que $\mathcal{V}_\rho(\alpha) = 1$.

3. Se α é insatisfatível então α não é universalmente válida.

4. Os termos de $\mathcal{H}_\alpha$ podem ser gerados automaticamente um após o outro.

Assim, uma forma de achar um subconjunto finito de $\mathcal{H}((\alpha))$ inconsistente é seguir o seguinte algoritmo de busca: Ir gerando instâncias de $\mathcal{H}((\alpha))$ e a cada certo intervalo de tempo testar a conjunção de instâncias de $\mathcal{H}((\alpha))$ para verificar se geramos inconsistência. Se a conjunção é consistente, geramos novas instâncias de $\mathcal{H}((\alpha))$ e testamos novamente esta conjunção maior. Continuamos desta maneira até um conjunto inconsistente ser achado, o qual só acontecerá no caso de α ser insatisfatível (e portanto, $\neg\alpha$ seria universalmente válida). Como $\mathcal{H}((\alpha))$ é infinito, o processo não parará quando α for satisfatível.

A desvantagem com esta forma de mostrar que α é insatisfatível, não é so o tempo gasto para testar a inconsistência de um subconjunto finito de $\mathcal{H}((\alpha))$, mas principalmente o fato de que se $\mathcal{H}((\alpha))$ é gerado de acordo com a complexidade crescente de termos básicos nas instâncias, uma grande quantidade de instâncias deveram ser geradas antes de obter uma conjunção inconsistente. No entanto, na maioria das vezes somente uma subconjunção inconsistente pequena é necessária. Portanto, o problema é descartar a geração das instâncias consistentes. Uma maneira para conseguir isto, é prever qual instância de duas cláusulas conterá literais complementares. Usando esta informação teríamos como controlar a maneira na qual os termos de Herbrand são construídos.

Exemplo 8.5.5 *Suponha que as cláusulas de α são*

α_1. $P(x, a, y)$

α_2. $\neg P(f(z), w, a) \vee \neg P(z, d, w)$

α_3. $P(f(a), u, a) \vee \neg P(u, a, f(a))$

Relembremos nossa convenção de que letras do começo do alfabeto, neste caso a, b, c e d, são símbolos constantes, e letras do final do alfabeto (u, v, w, x, y e z) são variáveis.

Assim, substituindo em α_1 e α_2, x por $f(z)$, e w e y por a, terão instâncias dessas cláusulas contendo literais complementares. As instâncias de α_1 e de α_2 obtidas desta forma são

$$\alpha'_1.\ P(f(z), a, a) \qquad \alpha'_2.\ \neg P(f(z), a, a) \vee \neg P(z, d, a)$$

Como os literais $P(f(z), a, a)$ e $\neg P(f(z), a, a)$ são complementares, de forma análoga a como se procedia no método de eliminação de literais complementares visto no capítulo 5, esses literais podem ser eliminados obtendo a seguinte cláusula:

α_4. $\neg P(z, d, a)$.

Uma instância de α_4 poderá casar com uma instância de α_3. Para isto ocorrer, devemos substituir z por $f(a)$ em α_4 e u por d em α_3. Resultando nas seguintes instâncias de α_3 e de α_4

$$\alpha'_3.P(f(a), d, a) \vee \neg P(d, a, f(a)) \qquad \alpha'_4.\neg P(f(a), d, a)$$

Como, nessas cláusulas há dois literais complementares eles podem ser removido e se obter a cláusula

α_5. $\neg P(d, a, f(a))$

Como, α_5 é um literal que colide com α_1 se substituirmos x por d e y por $f(a)$, isto é à instância

$$\alpha''_1.\ P(d, a, f(a))$$

a qual se contradiz com α_5.

Portanto o seguinte conjunto de cláusulas é inconsistente:

$$S = \{P(f(z), a, a), \neg P(f(z), a, a) \vee \neg P(z, d, a), P(f(a), d, a) \vee \neg P(d, a, f(a)), \neg P(f(a), d, a), P(d, a, f(a)), \neg P(d, a, f(a))\}$$

Onde cada elemento de s é uma instância de α_1, α_2 ou α_3 ou de algum de seus "resolventes"[1]. Porém, como algumas destas cláusulas têm variáveis, elas não são um subconjunto de $\mathcal{H}((\alpha))$, mas para achar esse subconjunto é suficiente substituir as variáveis por uma constante em $\mathcal{H}_\alpha$.

Deste exemplo podemos observar que só um pequeno segmento de $\mathcal{H}((\alpha))$ é contraditório. No entanto, se $\mathcal{H}((\alpha))$ fosse gerado pelo método de complexidade crescente de termos, o primeiro segmento contraditório conteria entre 3^6 a 15^6 cláusulas!

8.6 Unificador Mais Geral

A técnica usada no exemplo 8.5.5 consiste em aplicar uma sucessão de substituições de termos (não necessariamente básicos) por variáveis, cada substituição sendo escolhida de tal modo que o resultado de aplicá-la a duas cláusulas é que alguns dos literais α na primeira cláusula colidirá com um $\neg\alpha$ literal da segunda cláusula. Para formalizar esta noção introduziremos o conceito de *unificador mais geral*, abreviado por *umg*. Antes

[1] A noção de resolvente para o caso de cláusulas de primeira ordem somente será vista mais adiante.

porém, definiremos substituições de variáveis por termos, pois o método de resolução depende dessa operação básica sobre fbf's. Uma substituição fornece uma atribuição de termos às variáveis. Neste caso, cada atribuição é indicada por um par ordenado (termo, variável).

Definição 8.6.1 *Uma* **substituição** σ *é um conjunto de componentes de substituições independentes:*

$$\sigma = [(t_1, v_1), \ldots, (t_n, v_n)]$$

O primeiro elemento de cada componente é um termo e o segundo elemento é uma variável, tal que

1. $v_i \neq v_j$ *se* $i \neq j$
2. v_i *não ocorre em* t_j *para qualquer* i *e* j.

Como as substituições são feitas simultaneamente, então primeira restrição na definição de substituição é para evitar que uma variável seja trocada por termos diferentes o que resultaria numa indeterminação do resultado da substituição. A segunda restrição evita que uma variável que está sendo substituída seja novamente incorporada, o qual novamente resultaria numa indeterminação. Ou seja as restrições são para tornar a substituição completamente determinística.

Definição 8.6.2 *Se uma variável* v *é trocada em cada uma de suas ocorrências livres, na fbf* α*, pelo termo* t*, a fórmula resultante é chamada uma* **instância de substituição** *de* α*, e denotada por* $\alpha[(t, v)]$*, cuja leitura é "*α *com todas as ocorrências livres* t *substituídas por* v*". Analogamente,* $\alpha[(t_1, v_1), \ldots, (t_n, v_n)]$ *denota o resultado de trocar simultaneamente todas as ocorrências livres das diferentes variáveis* $v_1, \ldots, v_n$*, em* α*, por* $t_1, \ldots, t_n$*, respectivamente.*

A independência das componentes de uma substituição, que é assegurada pelas duas condições na definição 8.6.1, garante que duas substituições são, conjuntistamente, equivalentes se, e somente se, elas têm o mesmo efeito sobre todas as expressões. Portanto, aplicando uma substituição a uma expressão, não importa a ordem das componentes.

Seja σ uma substituição. α_σ denotará a fórmula resultante de aplicar a substituição σ a α.

Exemplo 8.6.3 *Sejam*

- $\alpha = P(x, z, y) \vee P(e, f(u, y), f(a, b))$

- $\sigma = [(e, x), (f(u, v), z), (f(a, b), y)]$
- $\tau = [(f(a, b), v)]$.

Então

- $\alpha_\sigma = P(e, f(u, v), f(a, b)) \vee P(e, f(u, f(a, b)), f(a, b))$
- $(\alpha_\sigma)_\tau = P(e, f(u, f(a, b)), f(a, b)) \vee P(e, f(u, f(a, b)), f(a, b))$.

Como as variáveis v_i's não ocorrem nos termos t_j's, então nenhuma ocorrência nova de qualquer variável v_i pode ser re-introduzida. Logo, podemos combinar a componente de substituição (t, x) com a substituição $\sigma = [(t_1, v_1), \ldots, (t_n, v_n)]$, desde que $x \neq v_i$ e v_i não ocorre em t, para todo i, $1 \leq i \leq n$, como segue

$$\sigma[(t, x)] = [(t_1[(t, x)], v_1), \ldots, (t_n[(t, x)], v_n), (t, x)]$$

Proposição 8.6.4 *Seja $\sigma = [(t_1, v_1), \ldots, (t_n, v_n)]$ e $\tau = [(t'_1, u_1), \ldots, (t'_m, u_m)]$ duas substituições, tais que $u_i \neq v_j$ e v_j não ocorre em t'_i, para cada i e j com $1 \leq i \leq m$ e $1 \leq j \leq n$. Então:*

$$\sigma\tau = [(t_1\tau, v_1), \ldots, (t_n\tau, v_n)] \cup \tau$$

é uma substituição

Demonstração: Direta. ■

Neste caso $\sigma\tau$ é chamada de **combinação** de σ com τ.

Exemplo 8.6.5 *Sejam as substituições σ e τ do exemplo 8.6.3. Note que as variáveis que são substituídas em σ (x, z e y) são diferentes das de τ (v) e que nenhuma variável de σ ocorre nos termos de τ ($f(a, b)$), caso contrário a variável seria re-introduzida. Logo, é possível combinar ambas substituições. Assim, a combinação das substituições σ com τ resulta na substituição:*

$$\begin{aligned}
\sigma\tau &= [(e, x), (f(u, v), z), (f(a, b), y)][(f(a, b), v)] \\
&= [(e[(f(a, b), v)], x), (f(u, v)[(f(a, b), v)], z), (f(a, b)[(f(a, b), v)], y)] \cup [(f(a, b), v)] \\
&= [(e, x), (f(u, f(a, b)), z), (f(a, b), y)] \cup [(f(a, b), v)] \\
&= [(e, x), (f(u, f(a, b)), z), (f(a, b), y), (f(a, b), v)]
\end{aligned}$$

Observe que se aplicarmos a substituição $\sigma\tau$ à fórmula α do exemplo 8.6.3 ($P(x, z, y) \vee P(e, f(u, y), f(a, b))$), obteremos a fórmula

$$\begin{aligned}\alpha_{\sigma\tau} &= P(x,z,y) \vee P(e, f(u,y), f(a,b))[(e,x),(f(u,f(a,b)),z),(f(a,b),y),(f(a,b),v)] \\ &= P(e, f(u,f(a,b)), f(a,b)) \vee P(e, f(u,f(a,b)), f(a,b)) \\ &= (\alpha_\sigma)_\tau.\end{aligned}$$

Esta proposição motiva a seguinte proposição.

Proposição 8.6.6 *Seja $\sigma = [(t_1, v_1), \ldots, (t_n, v_n)]$ e $\tau = [(t'_1, u_1), \ldots, (t'_m, u_m)]$ duas substituições, tais que $u_i \neq v_j$ e v_j não ocorre em t'_i, para cada i e j com $1 \leq i \leq m$ e $1 \leq j \leq n$. Então para toda fbf α*

$$(\alpha_\sigma)\tau = \alpha_{\sigma\tau}$$

Demonstração: Direta da definição de composição e de substituição. ■

A partir do conceito de substituição podemos estabelecer uma ordem parcial entre as fórmulas de L^P. Sejam α e β fórmulas de L^P. Dizemos que β é **mais geral ou igual** a α, denotado por $\alpha \preceq \beta$ se existe uma substituição σ tal que $\alpha = \beta_\sigma$. Assim,

Corolário 8.6.7 *Seja $\sigma = [(t_1, v_1), \ldots, (t_n, v_n)]$ e $\tau = [(t'_1, u_1), \ldots, (t'_m, u_m)]$ duas substituições, tais que $u_i \neq v_j$ e v_j não ocorre em t'_i, para cada i e j com $1 \leq i \leq m$ e $1 \leq j \leq n$. Então para toda fbf α*

$$\alpha_{\sigma\tau} \preceq \alpha_\sigma.$$

Demonstração: Direto da proposição 8.6.6. ■

Note que se $\alpha \preceq \beta$ então as fórmulas α e β casam (são unificadas) pela ação de uma substituição σ. A seguir estenderemos esta noção.

Definição 8.6.8 *Seja $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$ um conjunto de cláusulas e σ uma substituição. Γ é* **unificado** *por σ se $\alpha_{1_\sigma} = \alpha_{2_\sigma} = \cdots = \alpha_{n_\sigma}$. Neste caso chamamos σ de* **unificador**.

Exemplo 8.6.9 *Seja o conjunto de cláusulas $\Gamma = \{P(f(a), x) \vee \neg R(a, y), P(z, f(z)) \vee \neg R(u, f(u))\}$. Um unificador para Γ pode ser a substituição:*

$$\sigma = [(f(a), z), (f(f(a)), x), (a, u), (f(a), y)]$$

pois

$$(P(f(a), x) \vee \neg R(a, y))_\sigma = P(f(a), f(f(a))) \vee \neg R(a, f(a))$$

e

$$(P(z, f(z)) \vee \neg R(u, f(u)))_\sigma = P(f(a), f(f(a))) \vee \neg R(a, f(a))$$

Seja o conjunto de cláusulas $\Gamma = \{P(x, f(y)), P(a, f(f(u))), P(z, f(f(f(w))))\}$. *Um unificador para* Γ *pode ser a substituição:*

$$\sigma = [(a, x), (a, z), (f(w), u), (f(f(w)), y)\}$$

pois

$P(x, f(y))_\sigma = P(a, f(f(f(w))),$

$P(a, f(f(u)))_\sigma = P(a, f(f(f(w))),$ *e*

$P(z, f(f(f(w))))_\sigma = P(a, f(f(f(w)))$

Portanto um unificador é uma substituição que torna as expressões nas quais ele é aplicado iguais do ponto de vista sintático. É importante estar consciente de que embora as expressões resultantes sejam as mesmas, as expressões originais não necessariamente o são. O único que podemos afirmar delas que elas possuem instâncias que tornam as expressões iguais. Mas, existem situações que expressões denotam o mesmo valor semântico mas elas não podem ser unificadas pois suas expressões são sintaticamente diferentes. Assim, unificação é um processo estritamente sintático. Por exemplo, $x + 2$ pode ser unificado com $3 + 2$ amarrando x a 3, mas $x + 2$ não pode ser unificado com 5 ou $2 + 3$, que apesar de denotarem o mesmo valor, são sintaticamente distintas.

Definição 8.6.10 *Uma substituição* σ *é um* **unificador mais geral** *de um conjunto de cláusulas* Γ*, se para qualquer outro unificador* θ*,* $\Gamma_\theta \preceq \Gamma_\sigma$.

Portanto, pela definição de $\preceq$, existe uma substituição ϕ tal que $\Gamma_\sigma = (\Gamma_\theta)_\phi$.

Intuitivamente, entendemos que o unificador mais geral é aquele que além de fazer a menor quantidade possível de substituições deixa a fórmula mais genérica. No restante deste capítulo unificadores serão aplicados a fórmulas atômicas ocorrendo em cláusulas. Em todo caso um conjunto de fórmulas atômicas é também um conjunto de cláusulas.

Matematicamente, dizemos que o unificador mais geral de um conjunto de fórmulas atômicas fornece o ínfimo desse conjunto com respeito à ordem parcial $\preceq$. Porém, em muitos casos não existirá um unificador mais geral. Mas, se um conjunto de fórmulas atômicas Γ é unificável, então existe uma substituição mais geral unificando Γ e, além disso, podemos fornecer um algoritmo para achar esta substituição. Este algoritmo, devido a J.A. Robinson [Rob65], começa com o conjunto vazio de substituições e constrói, etapa por etapa, um unificador mais geral para o conjunto Γ de fórmulas atômicas. Construiremos nossas componentes de substituições unificadoras componente a componente.

Se, na k-ésima etapa, a substituição até então obtida é σ_k, e as fórmulas atômicas $\alpha_{1\sigma_k}, \ldots, \alpha_{n\sigma_k}$ em Γ_{σ_k} não são todas iguais, então o procedimento troca σ_k na base de um conjunto de discordância contendo a primeira expressão bem-formada em cada $\alpha_{i\sigma_k}$ que precisa ser mudada.

Definição 8.6.11 *Seja $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$, onde cada α_i é uma fórmula atômica com o mesmo símbolo de predicado. O* **conjunto de discordâncias** *de Γ é o conjunto de todas os termos das fórmulas atômicas em Γ que começam na posição do primeiro símbolo no qual nem todas as fórmulas atômicas coincidem.*

O conjunto de discordâncias ou é vazio (em cujo caso Γ já estaria unificado) ou ele contém um termo ou subtermo de cada fórmula atômica em Γ.

Exemplo 8.6.12 *O conjunto de discordâncias de*

$$\Gamma = \{P(x, f(g(y)), v), P(x, f(z), w), P(x, f(f(a)), w), P(x, f(z), a)\}$$

é $\{z, g(y), f(a)\}$. Já $\{v, w, a\}$ é o conjunto de discordância da etapa posterior.

Antes de exibir o algoritmo que fornece o unificador mais geral (umg) precisamos impor uma ordem total sobre L^P. Fórmulas bem formadas e seqüências de fórmulas bem formadas podem estar **ordenadas lexicograficamente** como segue:

1. Os símbolos primitivos são ordenados de acordo com o tipo (variáveis, constantes, símbolos de funções, símbolos de predicados e conectivos). Dentro de cada tipo, de acordo ao número de seus argumentos ou aridade (para símbolos de funções e predicados), depois por subíndices e finalmente, alfabeticamente. Os conectivos são ordenados de acordo com seu escopo, isto é, $\neg$, $\wedge$, $\vee$, $\rightarrow$, e $\leftrightarrow$. Graficamente, os símbolos são ordenados como segue

    variáveis
        subíndices
            alfabeticamente
    constantes
        subíndices
            alfabeticamente
    símbolos de funções
        aridade
            subíndices
                alfabeticamente
    símbolos de predicado
        aridade
            subíndices
                alfabeticamente

2. Fórmulas bem formadas são ordenadas pela quantidade de fórmulas atômicas que a compõem. Duas fbf's com a mesma quantidade de fórmulas atômicas são ordenadas pela ordem dos símbolos na primeira posição na qual elas diferem.

Exemplo 8.6.13 *Os literais* $P_1(x_1, f(x_2), a), P_2(a, f(a)), P_2(x_3, x_2), P_2(x_2, f(x_3)), P_2(f(x_1), a)$ *têm a seguinte ordem lexicográfica:*

$$P_2(x_2, f(x_3)), P_2(x_3, x_2), P_2(a, f(a)), P_2(f(x_1), a), P_1(x_1, f(x_2), a)$$

8.6.1 Algoritmo de unificação de Robinson

1. Faça $\sigma_1 = [\,]$ (a substituição vazia), faça $k = 0$, e vá para a etapa 2.

2. Faça $k \leftarrow k + 1$. Se os elementos de Γ_{σ_k} são todos iguais, faça $\sigma_0 = \sigma_k$ e pare. Caso contrário vá para a etapa 3.

3. Sejam s_k, t_k os dois primeiros termos na ordem lexicográfica do conjunto de discordâncias de Γ_{σ_k}. Se s_k é uma variável e não ocorre em t_k, faça $\sigma_{k+1} = \sigma_k[(t_k, s_k)]$ e vá para 2. Caso contrário pare (não existe nenhum umg).

Como $\Gamma_{\sigma_{k+1}}$ contém uma variável a menos que Γ_{σ_k}, o procedimento deve parar em no máximo m etapas, onde m é o número de variáveis de Γ. é claro, também, que se o procedimento pára na etapa 2, σ_0 é determinado unicamente e seus termos contém somente símbolos de função ocorrendo em Γ. é também verdade que se Γ é unificável então σ_0 está definido e é um umg. Para uma prova desse fato ver [Lan64].

Exemplo 8.6.14 *Seja o conjunto de fórmulas atômicas:*

$$\Gamma = \{P_1(x_1, f_1(x_1)), P_1(f_1(x_2), x_2), P_1(f_1(y_2), f_1(a))\}$$

A seguinte tabela mostra passo a passo a aplicação deste algoritmo

k	σ_k	Γ_{σ_k}	s_k	t_k
1	$[\,]$	$\{P_1(x_1, f_1(y_1)), P_1(f_1(x_2), x_2),$ $P_1(f_1(y_2), f_1(a))\}$	x_1	$f_1(x_2)$
2	$[(f_1(x_2), x_1)]$	$\{P_1(f_1(x_2), f_1(y_1)), P_1(f_1(x_2), x_2),$ $P_1(f_1(y_2), f_1(a))\}$	x_2	y_2
3	$[(f_1(x_2), x_1), (x_2, y_2)]$	$\{P_1(f_1(x_2), f_1(y_1)), P_1(f_1(x_2), x_2),$ $P_1(f_1(x_2), f_1(a))\}$	x_2	$f_1(y_1)$
4	$[(f_1(f_1(y_1)), x_1), (f_1(y_1), y_2),$ $(f_1(y_1), x_2)]$	$\{P_1(f_1(f_1(y_1)), f_1(y_1)),$ $P_1(f_1(f_1(y_1)), f_1(a))\}$	y_1	a
5	$[(f_1(f_1(a)), x_1), (f_1(a), y_2),$ $(f_1(a), x_2), (a, y_1)]$	$\{P_1(f_1(f_1(a)), f_1(a))\}$		
0	$[(f_1(f_1(a)), x_1), (f_1(a), y_2),$ $(f_1(a), x_2), (a, y_1)]$			

8.7 Resolução

Nesta seção apresentaremos um algoritmo para mostrar quando um conjunto de cláusulas é insatisfatível. Este algoritmo é uma extensão do algoritmo de resolução apresentado no capítulo 5. Antes porém será preciso introduzir alguns conceitos.

Definição 8.7.1 *Sejam C_1 e C_2 duas cláusulas e $x_1, \ldots, x_m$ as variáveis em ordem léxica que ocorrem tanto em C_1 quanto em C_2. A **separação** de C_1 e C_2 é a substituição $\theta = [(x_1, y_1), \ldots, (x_m, y_m)]$ onde $y_1, \ldots, y_m$ são variáveis que não ocorrem nem em C_1 nem em C_2. A separação é dita **padrão** se as variáveis $y_1, \ldots, y_m$ são as variáveis que seguem na ordem léxica á maior variável na ordem léxica que ocorre em C_1 e C_2.*

Assim, C_{2_θ} e C_1 não possuem variáveis em comum. Portanto, a separação nada mais é a renomeação de variáveis de uma das cláusulas (C_2) de tal forma que a nova cláusula não tenha qualquer variável em comum com a outra cláusula (C_1). A separação padrão pode ser vista como um algoritmo para se achar um separador de duas cláusulas.

Exemplo 8.7.2 *Sejam as cláusulas*

- $C_1 = P(x_1, f(x_2, a)) \vee \neg P(x_2, g(x_3))$ *e*
- $C_2 = P(g(x_2), f(x_3, b)) \vee \neg P(g(x_4), x_2)$.

Uma separação destas cláusula consiste é a substituição das variáveis x_2 e x_3 por uma variável diferente a x_1, x_2, x_3 e x_4. Já a separação padrão obrigaria a substituir x_2 por x_5 e x_3 por x_6, pois x_5 e x_6 são as próximas variáveis na ordem léxica. Assim,

$$\theta = [(x_5, x_2), (x_6, x_3)]$$

é a separação padrão das cláusulas C_1 *e* C_2*. A aplicação de* θ *à cláusula* C_1 *resulta na cláusula*

$$C_{2_\theta} = P(g(x_5), f(x_6, b)) \vee \neg P(g(x_4), x_5)$$

Claramente C_{2_θ} *não contém qualquer variável ocorrendo em* C_1*.*

A seguir estenderemos o conceito de um resolvente de duas cláusulas básicas (definição 5.3.1) para o caso onde as cláusulas não são necessariamente cláusulas básicas. Analogamente como no caso proposicional, cláusulas também podem ser vistas como um conjunto de literais.

Definição 8.7.3 *Um* **resolvente** *de duas cláusulas* C_1 *e* C_2 *é uma terceira cláusula* C_3 *obtida como segue:*

1. *Encontre a separação padrão* θ *de* C_1 *com* C_2*.*
2. *Se existe um par de conjuntos de literais* Γ *e* Δ*, tal que* $\Gamma = \{\alpha_1, \ldots, \alpha_k\} \subseteq C_1$*,* $\Delta = \{\beta_1, \ldots, \beta_l\} \subseteq C_2$ *e o conjunto* $\Gamma \cup \overline{\Delta}_\theta$*, onde* $\overline{\Delta} = \{\overline{\beta_1}, \ldots, \overline{\beta_l}\}$*, é unificável, então considere como unificador mais geral o unificador* σ_0*, obtido pelo algoritmo de Robinson. Portanto* Γ_{σ_0} *e* $(\Delta_\theta)_{\sigma_0}$ *são literais complementares (de fato eles são conjuntos unitários cujos elementos são literais complementares). Logo, defina* C_3 *como sendo o seguinte conjunto de literais:*

$$(C_1 - \Gamma)_{\sigma_0} \cup ((C_2 - \Delta)_\theta)_{\sigma_0}.$$

Observe que qualquer par de cláusulas tem no máximo um número finito de resolventes, pois existem somente uma quantidade finita de pares de conjuntos de literais Δ e Γ, e para cada par existe, no máximo, uma substituição σ_0. Assim, como para tem-se algoritmos tanto para achar separador padrão quanto o unificador mais geral, então claramente tem-se um algoritmo para achar o resolvente de duas cláusulas. Porém, por simplicidade, daqui em diante usaremos separadores e unificadores mais gerais os quais não necessariamente são obtidos por estes algoritmos.

Uma dedução por resolução é definida como antes, com o termo mais genérico "**resolvente** no lugar de "resolvente básico".

Definição 8.7.4 *Uma* **dedução por resolução** *de uma cláusula* α *a partir de um conjunto de cláusulas* Γ*, denotado por* $\Gamma \xrightarrow{*} \alpha$*, é uma seqüência finita de cláusulas* $\beta_1, \ldots, \beta_n$ *tal que*

1. β_n *é* α
2. *para cada* i, $1 \leq i \leq n$, *ou* β_i
   (a) *é um elemento de* Γ, *ou*
   (b) *é um resolvente de* β_j *e* β_k, *onde* $j, k < i$.

Note que, analogamente ao caso proposicional, a seqüência de cláusulas $\beta_1, \ldots, \beta_n$ de uma dedução por resolução, pode ser disposta na forma de uma árvore invertida.

Exemplo 8.7.5 *Seja o seguinte conjunto de cláusulas* Γ*:*

1. $P(x, h(x, y)) \vee P(x, f(y))$
2. $P(x, h(y, z)) \vee \neg P(x, z) \vee P(x, f(y))$
3. $\neg P(c, h(b, h(b, h(c, b)))) \vee P(c, f(b))$

E sejam as separações $\theta_1 = [(x_1, x), (y_1, y), (z_1, z)]$, $\theta_2 = [(x_2, x), (y_2, y), (z_3, z)]$ *e* $\theta_3 = [(x_3, x), (y_3, y)]$.

A árvore na figura 8.9, mostra a dedução por resolução de $\Gamma \xrightarrow{*} P(c, f(b))$.

Definição 8.7.6 *Uma dedução por resolução de* $\Box$ *a partir de um conjunto de cláusulas* Γ*, denotado por* $\Gamma \xrightarrow{*} \Box$*, é chamada uma* **refutação de resolução** *ou simplesmente uma* **refutação***, de* Γ.

Proposição 8.7.7 *Existe uma* **refutação** *de um conjunto de cláusulas* Γ (*isto é,* $\Gamma \xrightarrow{*} \Box$) *se e somente se* $\Box \in \mathcal{R}^n(\Gamma)$ *para algum* n*, onde* $\mathcal{R}^n$ *é definido como na definição 5.3.13, omitindo o adjetivo "básico".*

Demonstração: Análoga à prova da proposição 5.3.15. ■

Exemplo 8.7.8 *Suponha que desejamos mostrar que o seguinte conjunto de cláusulas é insatisfatível.*

1. $P(y, x)$
2. $\neg P(f(x), f(z)) \vee \neg P(b, x)$
3. $P(f(f(a)), x)$

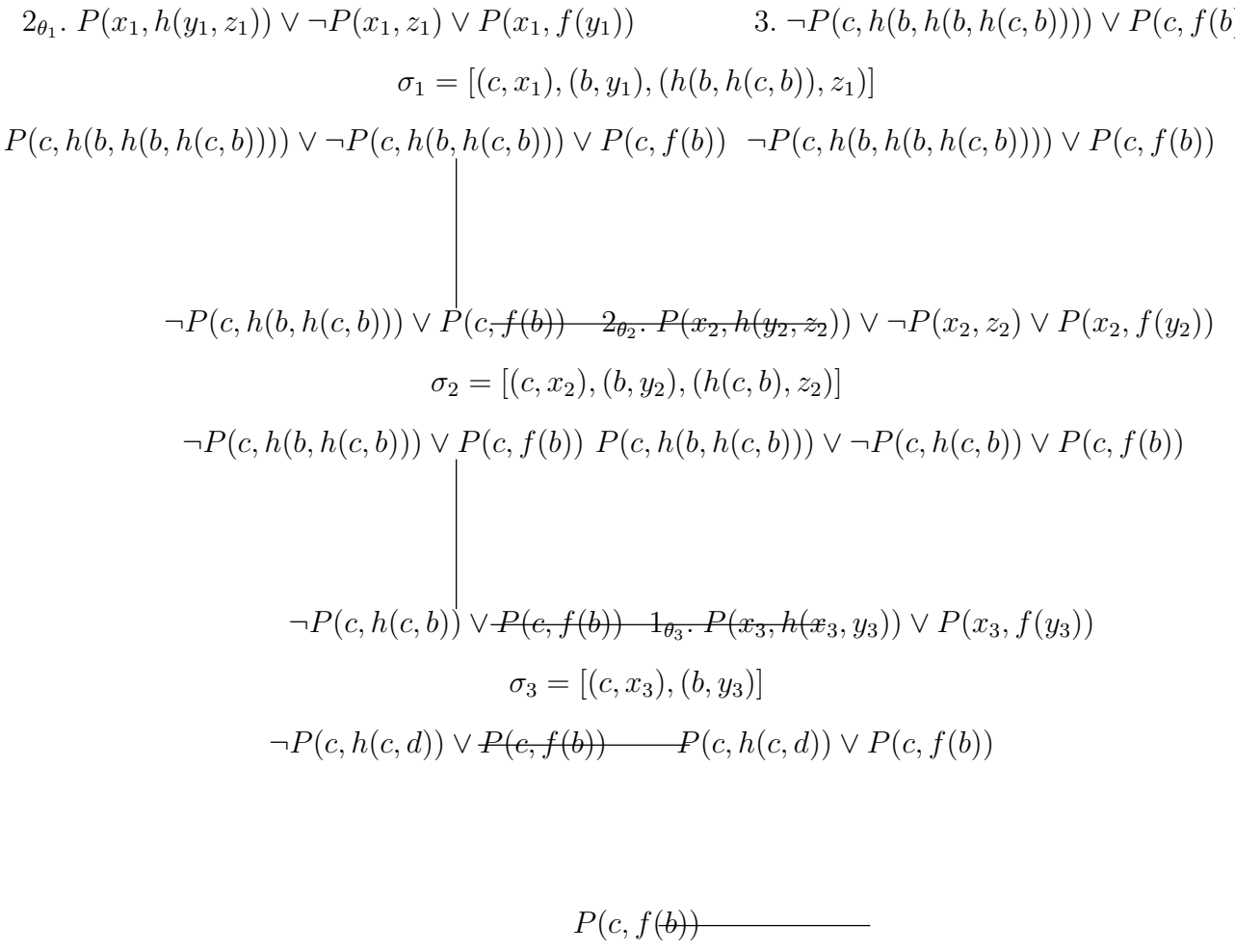


Figura 8.9: Árvore para a dedução por resolução de uma cláusula a partir de um conjunto de cláusulas.

Analogamente ao caso proposicional, nosso objetivo é mostrar que as cláusulas são inconsistentes exibindo uma refutação por resolução delas. Temos, agora, um problema mais complicado, uma vez que temos de fazer a unificação das cláusulas. No entanto a estratégia geral é a mesma.

Uma maneira fácil para fazer a separação das variáveis das cláusulas é indexar todas as variáveis da cláusula que pretendemos resolver, mesmo que às vezes não seja necessário renomear todas elas. Crescendo os índices cada vez que uma etapa na resolução é efetuada. No conjunto de cláusulas, acima, faremos o resolvente das cláusulas 1 e 2. Primeiro escrevemos uma nova versão das cláusulas 1 e 2:

$1'.\ P(y_1, x_1)$

$2'.\ \neg P(f(x_2), f(z_2)) \vee \neg P(b, x_2)$

Unificando $1'$ *com* $P(b, x_2)$ *da cláusula* $2'$*, achamos o umg* $\sigma_1[(b, y_1), (x_2, x_1)]$*. Aplicando* σ_1 *a* $1'$ *e* $2'$ *derivamos o resolvente* 4*, como ilustra a figura 8.10.*

$1'.\ P(b, x_2)$——————— $2'.\ \neg P(f(x_2), f(z_2)) \vee \neg P(b, x_2)$

$4. \neg P(f(x_2), f(z_2))$———

Figura 8.10: Derivação de um resolvente.

Renomeamos as variáveis em 3., obtendo a cláusula

$3'.\ P(f(f(a)), x_3)$

para unificar com $P(f(x_2), f(z_2))$*. Um unificador para estas fórmulas atômicas é* $\sigma_2 = [(f(a), x_2), (f(z_2), x_3)]$*. Aplicando este unificador às cláusulas* $3'$ *e* 4*, derivamos* $\square$*, como mostra a figura 8.11.*

$3'.\ P(f(f(a)), x_3)$——— $5.\ \neg P(f(x_2), f(z_2))$

$\square$———————

Figura 8.11: Árvore para a refutação de dois literais complementares.

É claro que nem todas as renomeações de variáveis do exemplo anterior foram necessárias, pois, por exemplo, a variável z só ocorria na cláusula 2. No entanto, não existe nenhum

dano em fazê-lo, somente trazendo lucro ao criar-se o hábito de renomeá-las, prevenindo os erros nos casos mais complicados. De fato, no exemplo **??** é mostrado porque é necessário fazer a separação padrão.

Exemplo 8.7.9 *Considere, agora, o seguinte conjunto de cláusulas*

1. $P(x, h(x, y))$
2. $\neg P(x, z) \vee P(x, h(y, z))$
3. $\neg P(c, h(b, h(a, h(c, b))))$

Para assegurar que as variáveis dessas cláusulas estão separadas na forma padrão geramos novas instâncias de cada cláusula, subescrevendo todas as variáveis com sub-índices na medida que vamos usando as cláusulas na resolução. Denotaremos essa nova instância por uma linha sobre o número original. Cada etapa da dedução por resolução de $\square$ *das cláusulas 1-3 procede como segue:*

1. *Listamos as cláusulas que pretendemos resolver,*
2. *Fazemos a separação padrão delas,*
3. *Fornecemos um umg dos literais selecionados nas cláusulas em 1,*
4. *Listamos as cláusulas novamente, tendo aplicado o umg a cada uma, e*
5. *Mostramos o resolvente derivado das cláusulas.*

Analogamente à técnica usada para eliminação de literais complementares em lógica proposicional, a refutação pode ser ilustrada através de uma árvore, só que aqui precisamos considerar as unificações que serão usadas em cada etapa. A árvore da figura 8.12 mostra a refutação, em forma de árvore, do conjunto de cláusulas acima.

Este exemplo mostra que uma cláusula (neste caso a cláusula 2)pode ser usada mais de uma vez numa refutação. Cada cláusula contendo variáveis, encarna uma quantidade infinita de instâncias. Como em ELC, não eliminamos uma cláusula ao usá-la, pois sempre podemos re-utilizá-la, nem somos obrigados a usar todas as cláusulas que nos são dadas.

Em cada um dos exemplos acima o conjunto de literais Γ e Δ (da definição de resolvente) foram unitários. Usualmente escolhemos cláusulas para resolvê-las e fixamos nossa atenção sobre um único literal em cada cláusula. No entanto, devemos estar atentos para tratar cada cláusula como um conjunto de literais, de vez que vários literais podem se colapsar num único literal após a unificação. Ignorando este fato podemos perder uma refutação mesmo ela existindo. Vejamos o seguinte exemplo de duas cláusulas:

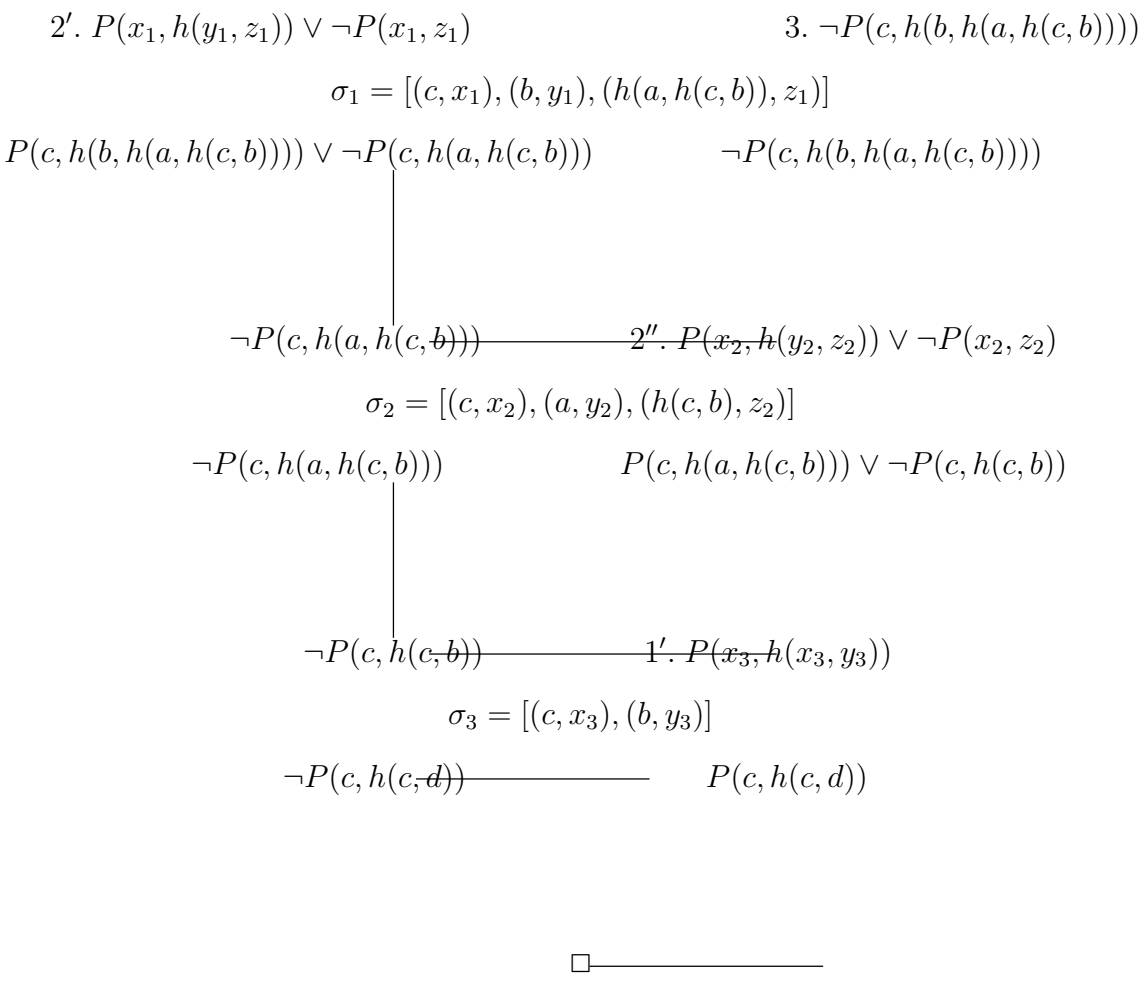


Figura 8.12: Árvore para a refutação de um conjunto de cláusulas.

C_1 : $P(a) \vee P(x)$

C_2 : $\neg P(a) \vee \neg P(y)$

Olhando somente um único literal de cada vez em cada cláusula, e ignorando o colapso de um conjunto a um único após a unificação, poderíamos gerar os seguintes resolventes.

1. Resolvendo o primeiro literal em cada cláusula obtemos

$$P(x) \vee \neg P(y)$$

2. Resolvendo o primeiro literal de C_1 e o segundo de C_2, usando o unificador $[(a, y)]$, obtemos

$$P(x) \vee \neg P(a)$$

3. Resolvendo o segundo literal de C_1 e o primeiro literal de C_2, usando o unificador $[(a, x)]$, obtemos

$$P(a) \vee \neg P(y)$$

4. Resolvendo o segundo literal em cada cláusula, usando $[(x, y)]$, obtemos

$$P(a) \vee \neg P(a)$$

Permitindo que cláusulas, que é um conjunto de literais, contenham duas instâncias do mesmo literal e eliminando somente uma quando derivamos um resolvente, força-nos a derivar somente cláusulas de dois elementos, tornando impossível derivar $\square$.

Olhando mais cuidadosamente as resoluções fornecidas acima, no segundo caso deveríamos ter obtido somente $P(x)$, o qual, quando usado apropriadamente com C_2, permitiria derivar $\square$. No terceiro item, deveríamos ter obtido $\neg P(y)$, que poderia ser usado com C_1 para derivar $\square$.

Poderíamos, também, ter escolhido considerar as cláusulas inteiras C_1 e C_2 como os conjuntos Γ e Δ, respectivamente, da definição de resolvente. Então usando o umg $[(a, x), (a, y)]$, derivaríamos imediatamente $\square$. Note que isto **não** é resolver os dois pares de literais ao mesmo tempo, o que, como observamos anteriormente, não é permitido. A razão para eliminar dois literais de cada cláusula é que antes da unificação existem dois literais em cada cláusula, mas após a unificação existe somente um em cada cláusula, e o resultado de resolver sobre essas cláusulas unitárias é $\square$.

Vimos que nossa computação para L^P é muito parecida com a usada para L_P. Entretanto, não podemos mais assegurar que o processo de tomar os resolventes parará, como sabemos temos um número infinito de instâncias possíveis de um literal toda vez que

nosso domínio de Herbrand contém uma constante e um símbolo de funções. O teorema de Herbrand assegura, dado um conjunto inconsistente de cláusulas Γ, algum subconjunto finito de $\mathcal{H}((\Gamma))$ é inconsistente, portanto $\square \in \mathcal{R}^n(\Gamma)$ para algum n. Mas não podemos colocar um limite sobre esse n. Não temos idéia quanto o programa deve executar para estabelecer a inconsistência de Γ embora sabemos que dado bastante tempo ele de fato o fará. O problema é grave quando Γ é consistente. Nosso processo de tomar os resolventes pode parar ou ficar executando para sempre. Portanto nos não temos um procedimento de decisão para determinar se, para um conjunto arbitrário de cláusulas Γ e algum n, $\square \in \mathcal{R}^n(\Gamma)$ ou não.

Recapitulando, se queremos provar, usando resolução, que uma fórmula α é um teorema, primeiro negamos a fórmula, convertemos-la à forma normal prenex, fechamos-la existencialmente, eliminamos quantificadores existenciais aplicando o processo de skolemização chegamos à forma normal de Skolem, ocultamos sumariamente quantificadores universais, a transformamos na forma clausal e evidenciamos as cláusulas para recém poder aplicar o método de eliminação de literais complementares.

Usamos o fecho existencial em vez do universal, pois queremos mostrar que $\neg\alpha$ é uma contradição, e uma fórmula é uma contradição se, e somente se, seu fecho existencial também é uma contradição.

Exemplo 8.7.10 *Seja a fórmula*

$$P(x) \to \exists x.P(x)$$

Vamos a mostrar que existe uma refutação da negação de esta fórmula.

Passo 1: Negar a fórmula

$$\neg(P(x) \to \exists x.P(x))$$

Passo 2: Forma normal prenex

$$\forall y.\neg(P(x) \to P(y))$$

Passo 3: Fechamento existencial da Fórmula

$$\exists x.\forall y.\neg(P(x) \to P(y))$$

Passo 4: Forma normal de Skolem

$$\forall y.\neg(P(a) \to P(y))$$

Passo 5: Ocultamento dos quantificadores universais

$$\neg(P(a) \rightarrow P(y))$$

Passo 6: Forma clausal

$$\begin{aligned} \neg(P(a) \rightarrow P(y)) &\twoheadrightarrow \neg(\neg P(a) \vee P(y)) \\ &\twoheadrightarrow \neg\neg P(a) \wedge \neg P(y) \\ &\twoheadrightarrow P(a) \wedge \neg P(y) \end{aligned}$$

Passo 7: Separação das cláusulas

1. $P(a)$
2. $\neg P(y)$

Passo 8: Resolução

A figura 8.13 ilustra a árvore que refuta as duas cláusulas anteriores. Na mesma árvore está indicado o unificador mais geral para essas duas cláusulas.

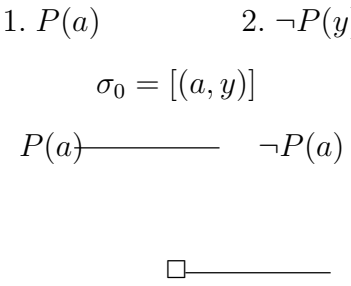


Figura 8.13: Árvore da refutação das cláusulas $P(a)$ e $\neg P(y)$.

O seguinte exemplo mostrará a importância de se realizar a separação padrão.

Exemplo 8.7.11 *Seja a fórmula*

$$\exists x.(\neg P(x) \vee P(f(x)))$$

Como mostrado no exemplo 6.5.13, esta fórmula é universalmente válida. Um prova disto usando resolução é a seguinte:

Passo 1) Negar a fórmula: $\neg\exists x.(\neg P(x) \vee P(f(x)))$

Passo 2) Forma normal prenex: $\forall x.\neg(\neg P(x) \vee P(f(x)))$

Passo 3) Fechar existencialmente a Fórmula: $\forall x.\neg(\neg P(x) \vee P(f(x)))$

Passo 4) Skolemizar: $\forall x.\neg(\neg P(x) \vee P(f(x)))$

Passo 5) Ocultar quantificadores universais: $\neg(\neg P(x) \vee P(f(x)))$

Passo 6) Forma clausal: $P(x) \wedge \neg P(f(x))$

Passo 7) Notação Clausal:

1. $P(x)$
2. $\neg P(f(x))$

Passo 8) Resolução: A figura 8.14 ilustra a árvore que refuta as duas cláusulas anteriores. Na mesma árvore está indicado o unificador mais geral para essas duas cláusulas, considerando a renomeação de x por y na segunda cláusula.

$$P(x) \qquad\qquad \neg P(f(y))$$
$$\sigma_0 = [(f(y), x)]$$
$$P(f(y)) \text{———} \neg P(f(y)))$$

□———

Figura 8.14: Árvore da refutação (com renomeação de variáveis) das cláusulas $P(x)$ e $\neg P(f(x))$.

Logo, a separação padrão foi fundamental neste exemplo, sem ela não teríamos como unificar as cláusulas $P(x)$ com $P(f(x))$ e portanto não teríamos uma refutação delas.

Note o que aconteceria se tentarmos refutar a cláusula $P(x) \rightarrow \forall x.P(x)$, a qual não é universalmente válida.

Exemplo 8.7.12 *Seja a fórmula*

$$P(x) \rightarrow \forall x.P(x)$$

Vamos tentar mostrar que existe uma refutação da negação desta fórmula.

Passo 1: Negar a fórmula

$$\neg(P(x) \rightarrow \forall x.P(x))$$

Passo 2: Forma normal prenex

$$\exists y.\neg(P(x) \to P(y))$$

Passo 3: Fechar existencialmente a Fórmula

$$\exists x.\exists y.\neg(P(x) \to P(y))$$

Passo 4: Skolemizar

$$\neg(P(a) \to P(b))$$

Passo 5: Eliminação de quantificadores universais: Como não, há quantificadores universais a fórmula fica a mesma.

Passo 6: Forma normal conjuntiva

$$\begin{aligned} \neg(P(a) \to P(b)) &\twoheadrightarrow \neg(\neg P(a) \vee P(b)) \\ &\twoheadrightarrow \neg\neg P(a) \wedge \neg P(b) \\ &\twoheadrightarrow P(a) \wedge \neg P(b) \end{aligned}$$

Passo 7: Notação Clausal

1. $P(a)$
2. $\neg P(b)$

Passo 8: Resolução: Como $P(a)$ e $P(b)$ não são unificáveis e portanto não é possível achar um resolvente para estas cláusulas. Logo, não são refutáveis.

Neste exemplo, o procedimento de refutação teria condições de parar indicando que não é possível se chegar a uma refutação, pois não há como unificar qualquer par de cláusulas. Já no próximo exemplo, veremos que sempre haverá uma possibilidade de se fazer um resolvente entre duas cláusulas, e portanto o algoritmo não terá como constatar que sua busca foi exaustiva.

Note que numa dedução por resolução (definição 8.7.4) nada impede incluir resolventes que não venham ser usados para chegar à contradição ($\square$). Assim, no exemplo a seguir, consideraremos a cada etapa todas as possibilidades de resolventes possíveis que podem ser obtidos das cláusulas encontradas até esse momento (a cada estágio essa quantidade é finita).

Exemplo 8.7.13 *Sejam as cláusulas*

- $C_1 = P(x_1, f(x_1)) \vee P(a, x_1)$

- $C_2 = \neg P(x_2, f(x_3)) \vee P(x_2, x_3)$

A partir delas (e fazendo separação padrão) teremos no máximo dois resolventes:

- $C_3 = P(f(x_3), f(f(x_3))) \vee P(a, x_3)$
- $C_4 = P(x_1, x_1) \vee P(a, x_1)$

No próximo estágio, procuram-se os resolventes entre as cláusulas C_1, C_2, C_3 e C_4. Assim, além do C_3 e C_4 obtém-se os resolventes

- $C_5 = P(f(f(x_3)), f(f(f(x_3)))) \vee P(a, x_3)$
- $C_6 = P(a, f(x_3)) \vee P(f(x_3), x_3)$
- $C_7 = P(f(x_3), f(x_3)) \vee P(a, x_3)$

Note que a cada estágio sempre serão geradas cláusulas contendo dois literais positivos. Além disso no estágio 1 foi gerada a cláusula C_3 ($P(f(x_3), f(f(x_3))) \vee P(a, x_3)$), no estágio 2 foi gerada a cláusula C_5 ($P(f(f(x_3)), f(f(f(x_3)))) \vee P(a, x_3)$), no estágio 3 será gerada a cláusula C_8 ($P(f(f(f(x_3))), f(f(f(f(x_3))))) \vee P(a, x_3)$), no estágio 4 será gerada a cláusula $P(f(f(f(f(x_3)))), f(f(f(f(f(x_3)))))) \vee P(a, x_3)$, etc. pelo que este processo nunca parará.

É claro que no exemplo acima é fácil identificar que essas duas cláusulas nunca levarão a uma contradição, porém como o que queremos é um algoritmo capaz de detectar isso para qualquer conjunto de cláusulas não satisfatíveis, teríamos que estabelecer critérios precisos para detectar isto. Lamentavelmente, não existem tais critérios. De fato, Alonso Church e Alan Turing, de forma independente, provaram em 1936 que a lógica clássica de 1^a ordem é indecidível, ou seja que independente do método usado, nunca poderemos ter um algoritmo capaz de decidir se um fórmula qualquer é um universalmente válida (ou equivalentemente sua negação é uma contradição) ou não.

8.8 Resultados de Completude

Sabendo o que significa verdade, dedução e completação no cálculo de predicados, pretendemos investigar as relações entre essas noções. Primeiro veremos que T^P é seguro. Isto é que todo teorema de T^P é uma fórmula universalmente válida. Segundo, estabeleceremos que se uma fórmula de L^P é universalmente válida, então é possível realizar uma refutação por resolução de sua negação. Finalmente, mostraremos que se é possível realizar uma refutação por resolução de sua negação então pode-se provar que ele é um teorema de T^P.

Teorema 8.8.1 (T^P é segura) *Todo teorema de T^P é uma fórmula universalmente válida. Isto é,*

$$\textit{se } \vdash \alpha, \textit{ então } \models \alpha$$

Demonstração: Como posto no capítulo 6 toda tautologia é universalmente válida, portanto os axiomas A_1, A_2 e A_3 são universalmente válidos. Das observações no final do capítulo 6 temos que os axiomas A_4 e A_5 também são universalmente válidos. Finalmente, das mesmas observações temos que a regra de generalização e *modus ponens* preservarão validade universal, isto é se α é universalmente válida, então $\forall x.\alpha$ também é universalmente válida. Analogamente, se α e $\alpha \to \beta$ são universalmente válidas, então β é universalmente válidas. ■

A seguir, mostraremos que a negação de uma fórmula é universalmente válida, então a fórmula é refutável. Mas, antes precisaremos mostrar uma série de resultados prévios.

Lema 8.8.2 *Seja $\mathcal{H}_\Gamma$ o domínio de Herbrand de Γ e T um conjunto de termos tal que $T \subseteq \mathcal{H}_\Gamma$, então $\mathcal{R}(T((\Gamma))) \subseteq T((\mathcal{R}(\Gamma)))$.*

Demonstração: Suponha que $\gamma \in T((\Gamma))$. Como $\Gamma \subseteq \mathcal{R}(\Gamma)$ e $\mathcal{R}(\Gamma) \subseteq T((\mathcal{R}(\Gamma)))$, então $\gamma \in T((\mathcal{R}(\Gamma)))$. Se γ é um resolvente, então $\gamma = (\alpha\psi - \mathtt{p}\psi) \cup (\beta\tau - \mathtt{q}\tau)$, onde $\alpha, \beta \in \Gamma$, $\alpha\psi, \beta\tau \in T((\Gamma))$ e $\mathtt{p}\sigma$ e $\mathtt{q}\tau$ são literais complementares. Sejam ψ e τ as seguintes substituições $\psi = [(s_1, x_1), \ldots, (s_k, x_k)]$ e $\tau = [(t_1, y_1), \ldots, (t_m, y_m)]$, onde $x_1, \ldots, x_k$ são todas as variáveis de α e $y_1, \ldots, y_m$ são todas as variáveis de β. Seja $\theta = [(x_{k+1}, y_1), \ldots, (x_{k+m}, y_m)]$ a separação padrão das varáveis de α e β. Defina, a substituição

$$\sigma = [(s_1, x_1), \ldots, (s_k, x_k), (t_1, x_{k+1}), \ldots, (t_m, x_{k+m})]$$

Claramente, $\alpha\sigma = \alpha\psi$, $\beta\theta\sigma = \beta\tau$, $\mathtt{p}\sigma = \mathtt{p}\psi$ e $\mathtt{q}\theta\sigma = \mathtt{q}\tau$. Logo, $\gamma = (\alpha - \mathtt{p})\sigma \cup (\beta - \mathtt{q})\theta\sigma$.

Como σ unifica o conjunto $\mathtt{p} \cup \mathtt{q}^{op}\tau$, existe um unificador mais geral σ_0 e um resolvente γ' de α e β, tal que

$$\gamma' = (\alpha - \mathtt{p})\sigma_0 \cup (\beta - \mathtt{q})\theta\sigma_0$$

Como σ_0 é um unificador mais geral e σ é um unificador de $\mathtt{p} \cup \mathtt{q}^{op}\tau$, então existe uma substituição λ tal que $(\mathtt{p} \cup \mathtt{q}^{op}\tau)\sigma = ((\mathtt{p} \cup \mathtt{q}^{op}\tau)\sigma_0)\lambda$ e portanto $\sigma_0\lambda = \sigma$. Logo, $\gamma = \gamma'\lambda$. Assim, $\gamma \in T((\mathcal{R}(\Gamma)))$, isto é γ é uma instância de substituição de um resolvente de cláusulas em Γ. ■

Lema 8.8.3 $\mathcal{R}^n(T((\Gamma))) \subseteq T((\mathcal{R}^n(\Gamma)))$

DEMONSTRAÇÃO: Provaremos por indução em n.

Caso base: Se $n = 1$ então como $\mathcal{R}^1(T((\Gamma))) = \mathcal{R}^1(T((\Gamma)))$ e $T((\mathcal{R}(\Gamma))) = T((\mathcal{R}^1(\Gamma)))$, temos que pelo lema 8.8.2, $\mathcal{R}^n(T((\Gamma))) \subseteq T((\mathcal{R}^n(\Gamma)))$.

Etapa indutiva: Assuma que $\mathcal{R}^k(T((\Gamma))) \subseteq T((\mathcal{R}^k(\Gamma)))$. Mostraremos que

$$\mathcal{R}^{k+1}(T((\Gamma))) \subseteq T((\mathcal{R}^{k+1}(\Gamma)))$$

$$\begin{array}{lll} \mathcal{R}^{k+1}(T((\Gamma))) & = \mathcal{R}(\mathcal{R}^k(T((\Gamma)))) & \text{definição de } \mathcal{R}^{k+1} \\ & \subseteq \mathcal{R}(T((\mathcal{R}^k(\Gamma)))) & \text{hipótese indutiva} \\ & \subseteq T(\mathcal{R}((\mathcal{R}^k(\Gamma)))) & \text{lema 8.8.2} \\ & \subseteq T(\mathcal{R}^{k+1}(\Gamma))) & \text{definição de } \mathcal{R}^{k+1} \ \blacksquare \end{array}$$

Teorema 8.8.4 *Se Γ é um subconjunto insatisfatível de L^P, então $\square \in \mathcal{R}^n(\Gamma)$, para algum n.*

DEMONSTRAÇÃO: Se Γ é insatisfatível, então, pelo teorema 8.5.4, algum conjunto finito de $\mathcal{H}((\Gamma))$ é inconsistente. Suponha que este conjunto inconsistente é $T((\Gamma))$. Como $T((\Gamma))$ é um conjunto de cláusulas básicas (sem variáveis) então pelo corolário 5.4.13, temos que $T((\Gamma)) \xrightarrow{*} \square$. Assim, pelo corolário 5.3.16 temos que $\square \in \mathcal{R}^n(T((\Gamma)))$ para algum n. Pelo lema 8.8.3 temos que $\square \in T((\mathcal{R}^n(\Gamma)))$ para algum n. Uma vez que substituir variáveis numa cláusula não vazia por termos, não pode gerar uma cláusula vazia, então $\square \in \mathcal{R}^n(\Gamma)$ para algum n. $\blacksquare$

Corolário 8.8.5 *Se $\models \alpha$ então $\neg\alpha \xrightarrow{*} \square$*

DEMONSTRAÇÃO: Se $\models \alpha$ então $\neg\alpha$ é insatisfatível. Logo, pelo teorema anterior, $\square \in \mathcal{R}^n(\neg\alpha)$, para algum n. Pelo corolário 5.3.16, temos que $\neg\alpha \xrightarrow{*} \square$. $\blacksquare$

Lema 8.8.6 *Se $\neg\alpha \xrightarrow{*} \beta$ então $\vdash \neg\alpha \to \beta$.*

DEMONSTRAÇÃO: Se $\neg\alpha \xrightarrow{*} \beta$ então existe uma dedução por resolução de β a partir de $\neg\alpha$. Seja $\beta_1, \ldots, \beta_n$ uma tal dedução por resolução. Mostraremos por indução em $1 \leq i \leq n$ que $\vdash \neg\alpha \to \beta_i$.

Caso base: Se $i = 1$ então, por definição de dedução por resolução, β_i só pode ser uma cláusula de $\neg\alpha$. Então, trivialmente, β_1 é uma prova de $\neg\alpha \vdash \beta_1$. Como não se aplicou Gen na prova, podemos aplicar o teorema da dedução e afirmar que existe uma prova de $\vdash \neg\alpha \to \beta_1$.

Etapa indutiva: Assuma que $\vdash \neg\alpha \to \beta_i$ para cada $i \leq k$. Devemos mostrar que $\vdash \neg\alpha \to \beta_{k+1}$. Se β_{k+1} é uma cláusula de $\neg\alpha$, então de modo análogo ao caso base, temos que $\vdash \neg\alpha \to \beta_{k+1}$. Se β_{k+1} é um resolvente de β_i e β_j, com $i, j \leq k$, então pelo lema

5.4.15, $\vdash (\beta_i \wedge \beta_j) \rightarrow \beta_{k+1}$, isto é $\vdash \neg(\beta_i \rightarrow \neg\beta_j) \rightarrow \beta_{k+1}$. Pela hipótese indutiva temos que $\vdash \neg\alpha \rightarrow \beta_i$ e $\vdash \neg\alpha \rightarrow \beta_j$. Logo, pelo exercício 1.f. do capítulo 4, temos que $\vdash \neg\alpha \rightarrow (\beta_i \wedge \beta_j)$. Assim, pela proposição 4.3.2.i, temos que $\vdash \neg\alpha \rightarrow \beta_{k+1}$.

Como, $\beta_n = \beta$, então $\vdash \neg\alpha \rightarrow \beta$. ■

Teorema 8.8.7 *Se existe uma refutação por resolução da negação de α, então α é um teorema. Isto é,*

$$\textit{se } \neg\alpha \xrightarrow{*} \square, \textit{ então } \vdash \alpha.$$

Demonstração: Se $\neg\alpha \xrightarrow{*} \square$ então, pelo lema 8.8.6, $\vdash \neg\alpha \rightarrow \square$. Como $\square$ é uma abreviação da fórmula $\beta \wedge \neg\beta$, ou equivalentemente da fórmula $\neg(\beta \rightarrow \beta)$ então $\vdash \neg\alpha \rightarrow \neg(\beta \rightarrow \beta)$. Assim,

α_1)	$\neg\alpha \rightarrow \neg(\beta \rightarrow \beta)$	
α_2)	$(\neg\alpha \rightarrow \neg(\beta \rightarrow \beta)) \rightarrow ((\beta \rightarrow \beta) \rightarrow \alpha)$	prop. 4.3.3.d
α_3)	$(\beta \rightarrow \beta) \rightarrow \alpha$	MP α_1, α_2
α_4)	$\beta \rightarrow \beta$	prop. 4.2.1
α_5)	α ■	

Teorema 8.8.8 *Para o cálculo de predicado as seguintes três afirmações são equivalentes:*

1. $\vdash \alpha$ *(α é um teorema);*
2. $\models \alpha$ *(α é universalmente válida); e*
3. $\neg\alpha \xrightarrow{*} \square$ *(existe uma refutação por resolução de $\neg\alpha$).*

Demonstração: Direto dos teoremas 8.8.1 8.8.7 e do corolário 8.8.5. ■

Assim lógica de predicado é segura, completa e consistente. No entanto ela não é decidível.

Observe que no caso proposicional temos um teorema da completude forte:

$$\Gamma \models \alpha \text{ se, e somente se, } \Gamma \vdash \alpha \text{ se, e somente se, } \Gamma \xrightarrow{*} \alpha$$

mas no caso da lógica de predicados isto não é verdade, pois nos temos, por exemplo, que $P(x) \vdash \forall x.P(x)$ mas $\forall x.P(x)$ não é conseqüência lógica de $P(x)$, isto é, $P(x) \not\models \forall x.P(x)$. Isto se deve a que as exigências da relação de conseqüência lógica são muito permissivas, mas se adicionarmos mais restrições a estas, conseguiremos ter uma noção de conseqüência semântica que coincida com $\vdash$.

Definição 8.8.9 *Seja* $\Gamma \subseteq L^P$ *e* $\alpha \in L^P$. α *é* **conseqüência lógica forte** *de* Γ*, denotado por* $\Gamma \mid\!\approx \alpha$*, se toda vez que todas as fórmulas de* Γ *são verdadeiras numa interpretação,* α *também é verdadeira na mesma interpretação. Isto é,*

$\Gamma \mid\!\approx \alpha$ *se, e somente se, para cada interpretação* $\mathcal{D}$ *tal que* $\mathcal{D} \models \Gamma$ *temos que* $\mathcal{D} \models \alpha$

onde, $\mathcal{D} \models \Gamma$ *abrevia* $\mathcal{D} \models \beta$ *para cada* $\beta \in \Gamma$.

Proposição 8.8.10 *Se* $\Gamma \models \alpha$ *então* $\Gamma \mid\!\approx \alpha$

Demonstração: Se $\mathcal{D} \models \Gamma$ para uma interpretação $\mathcal{D}$, então para cada atribuição $\rho : X \longrightarrow D$, temos que $\mathcal{D} \models_\rho \Gamma$. Como $\Gamma \models \alpha$, então $\mathcal{D} \models_\rho \alpha$, para cada atribuição $\rho : X \longrightarrow D$. Portanto, $\mathcal{D} \models \alpha$. ■

Proposição 8.8.11 *Se* $\Gamma \mid\!\approx \alpha$ *então* $FU(\Gamma) \models \alpha$*, onde* $FU(\Gamma)$ *é o fecho universal de todas as fórmulas em* Γ.

Demonstração: Note que uma fórmula é verdadeira numa interpretação se, e somente se, seu fecho universal também é verdadeira para essa interpretação. Note também que se uma fórmula fechada é verdadeira numa interpretação, então ela é verdadeira nessa interpretação para qualquer atribuição. Logo, se $FU(\Gamma)$ é verdadeira para uma interpretação $\mathcal{D}$ e uma atribuição ρ, então $FU(\Gamma)$ é verdadeira na interpretação $\mathcal{D}$. Portanto, Γ é verdadeira na interpretação $\mathcal{D}$. Logo, como $\Gamma \mid\!\approx \alpha$, então α é verdadeira para essa interpretação (e para a atribuição ρ). Portanto, $FU(\Gamma) \models \alpha$. ■

Usaremos a notação $\mid\!\approx \alpha$ em vez de $\emptyset \mid\!\approx \alpha$.

Corolário 8.8.12 $\models \alpha$ *se, e somente se,* $\mid\!\approx \alpha$

Demonstração: Direto da proposição 8.8.10 temos que se $\models \alpha$ então $\mid\!\approx \alpha$. Por outro lado, se $\mid\!\approx \alpha$, então direto da definição de $\mid\!\approx$ e de universalmente válida, temos que α é universalmente válida, isto é, $\models \alpha$. ■

Note que para a relação de conseqüência forte vale o teorema da compacidade, isto é, se $\Gamma \mid\!\approx \alpha$ então $\Gamma_0 \mid\!\approx \alpha$ para algum $\Gamma_0 \subseteq \Gamma$ finito. Mas, não vale o teorema da dedução irrestrito, pois trivialmente temos que $P(x) \mid\!\approx \forall x.P(x)$, mas não temos que $\mid\!\approx P(x) \rightarrow \forall x.P(x)$. Assim, deveríamos impor algumas condições, semelhantes ao caso do teorema da dedução para $\vdash$ (7.2.4), mas se tratando de interpretação isto resulta difícil.

Proposição 8.8.13 *Se* $\Gamma \vdash \alpha$ *então* $\Gamma \mid\!\approx \alpha$

DEMONSTRAÇÃO: Seja $\alpha_1, \ldots, \alpha_n$ uma prova de $\Gamma \vdash \alpha$ junto com uma justificativa para cada passo da dedução. Mostraremos por indução que $\Gamma \mid\!\approx \alpha_i$ para cada $1 \leq i \leq n$.

Caso base: $i = 1$. α_1 ou é uma fórmula de Γ ou é um axioma. No primeiro caso, trivialmente temos que $\alpha \models \alpha$, e usando o teorema da compacidade (observação 3. do capítulo 6), temos que $\Gamma \models \alpha$. Logo, pela proposição 8.8.10, temos que $\Gamma \mid\!\approx \alpha$. No segundo caso, temos que $\vdash \alpha$. Logo, aplicando o teorema seguro temos que $\models \alpha$. Assim, novamente usando o teorema da compacidade e a proposição 8.8.10, temos que $\Gamma \mid\!\approx \alpha$.

Etapa indutiva: Assuma que $\Gamma \mid\!\approx \alpha_i$, para $i < k$. Devemos mostrar que $\Gamma \mid\!\approx \alpha_k$.

1. Se α_k é um axioma ou um elemento de Γ, o resultado segue como no caso base, acima.

2. Se α_k sai por MP de α_i e $\alpha_j = \alpha_i \to \alpha_k$ com $i, j < k$, então pela hipóteses indutiva $\Gamma \mid\!\approx \alpha_i$ e $\Gamma \mid\!\approx \alpha_i \to \alpha_k$. Isto é, para cada interpretação $\mathcal{D}$, tal que $\mathcal{D} \models \Gamma$, $\mathcal{D} \models \alpha_i$ and $\mathcal{D} \models \alpha_{\to}\alpha_k$. Seja $\rho : X \longrightarrow D$ uma atribuição de valores às variáveis arbitrária. Então, pela definição 6.5.10, $\mathcal{V}_\rho(\alpha_i) = 1$ e $\mathcal{V}_\rho(\alpha_i \to \alpha_k) = 1$. Pela definição 6.5.4, $\mathcal{V}_\rho(\alpha_i \to \alpha_k) = 1$ se, e somente se, $\mathcal{V}_\rho(\alpha_i) \Rightarrow \mathcal{V}_\rho(\alpha_k) = 1$. Como $\mathcal{V}_\rho(\alpha_i) = 1$, então $\mathcal{V}_\rho(\alpha_i) \Rightarrow \mathcal{V}_\rho(\alpha_k) = 1$ se, e somente se, $1 \Rightarrow \mathcal{V}_\rho(\alpha_k) = 1$. Mas, pela definição do operador booleano $\Rightarrow$, $1 \Rightarrow \mathcal{V}_\rho(\alpha_k) = 1$ se, e somente se, $\mathcal{V}_\rho(\alpha_k) = 1$. Logo, se $\mathcal{D} \models \Gamma$ então $\mathcal{D} \models \alpha_k$. Portanto, $\Gamma \mid\!\approx \alpha_k$.

3. Se α_k sai por Gen de α_i com $i < k$, então $\alpha_k = \forall x.\alpha_i$ para alguma variável x. Seja $\mathcal{D}$ uma interpretação tal que $\mathcal{D} \models \Gamma$. Então, pela hipóteses indutiva ($\Gamma \mid\!\approx \alpha_i$), temos que $\mathcal{D} \models \alpha_i$, isto é para qualquer atribuição de valores às variáveis $\rho : X \longrightarrow D$, $\mathcal{V}_\rho(\alpha_i) = 1$. Como a atribuição ρ é arbitrária, então $\mathcal{V}_{\rho'}(\alpha_i) = 1$ para qualquer atribuição $\rho' : X \longrightarrow D$ a qual varia de ρ, no máximo, na atribuição dada à variável x. Logo, pela definição 6.5.4, $\mathcal{V}_\rho(\forall x.\alpha_i) = 1$. Portanto, $\Gamma \mid\!\approx \forall x.\alpha_i$, isto é, $\Gamma \mid\!\approx \alpha_k$.

Portanto, $\Gamma \mid\!\approx \alpha_i$ para cada $1 \leq i \leq n$. Como $\alpha_n = \alpha$, então $\Gamma \mid\!\approx \alpha$. ■

Proposição 8.8.14 *Seja* Γ *uma conjunto de cláusulas e* α *uma cláusula. Se* $\Gamma \mid\!\approx \alpha$ *então* $\Gamma \xrightarrow{*} \alpha$. *Ou seja, se* α *é conseqüência lógica forte de* Γ*, então existe uma dedução por resolução de* α *a partir de* Γ.

DEMONSTRAÇÃO: Se $\Gamma \mid\!\approx \alpha$ então $FU(\Gamma) \models \alpha$. Pelo teorema da compacidade existe $\Gamma_0 \subseteq FU(\Gamma)$ finito tal que $\Gamma_0 \models \alpha$. Seja $\Gamma_0 = \{\alpha_1, \ldots, \alpha_n\}$. Aplicando o teorema da dedução temos que $\models \alpha_1 \to (\ldots \to (\alpha_n \to \alpha)\ldots)$. Pelo teorema da completude, $\neg(\alpha_1 \to (\ldots \to (\alpha_n \to \alpha)\ldots) \xrightarrow{*} \square$. Como, $\neg(\alpha_1 \to (\ldots \to (\alpha_n \to \alpha)\ldots) \equiv \alpha_1 \wedge \ldots \alpha_n \wedge \neg\alpha$, então $\Gamma_0, \neg\alpha \xrightarrow{*} \square$ (observe que cada α_i é uma cláusula). Logo, pelo corolário 5.3.11, temos que $\Gamma_0 \xrightarrow{*} \alpha$. Portanto, pela propriedade de monotonicidade à esquerda (proposição 5.3.7), $\Gamma \xrightarrow{*} \alpha$. ■

Proposição 8.8.15 $\Gamma \xrightarrow{*} \beta$ *implica* $\Gamma \vdash \beta$.

DEMONSTRAÇÃO: $\Gamma \xrightarrow{*} \beta$ implica que uma dedução por resolução de β a partir de Γ existe.

Mostremos, por indução que para todo i, $\Gamma \vdash \alpha_i$.

Caso base: $i = 1$. α_1 é uma cláusula de Γ. Então, trivialmente, a seqüência α_1 é uma prova de $\Gamma \vdash \alpha_1$.

Etapa indutiva: Assuma que $\Gamma \vdash \alpha_i$, para cada $i < k$. Devemos mostrar que $\Gamma \vdash \alpha_k$.

1. Se α_k é uma cláusula de Γ, então, como no caso para $i = 1$, temos que trivialmente, $\Gamma \vdash \alpha_k$.

2. Se α_k é um resolvente de duas cláusulas anteriores na seqüência, suponha que de α_i e α_j, onde $i, j < k$, então $\alpha_k = (\alpha_i - \mathtt{p}) \cup (\alpha_j - \mathtt{p}^{op})$ para algum literal $\mathtt{p}$. Pela hipóteses indutiva temos que $\Gamma \vdash \alpha_i$ e $\Gamma \vdash \alpha_j$. Observe que $\mathtt{p} \vee \alpha \equiv \mathtt{p}^{op} \rightarrow \alpha$. Assim, $\alpha_i \equiv \mathtt{p}^{op} \rightarrow (\alpha_i - \mathtt{p})$ e $\alpha_j \equiv \mathtt{p} \rightarrow (\alpha_j - \mathtt{p}^{op})$. Portanto $\Gamma \vdash \mathtt{p}^{op} \rightarrow (\alpha_i - \mathtt{p})$ e $\Gamma \vdash \mathtt{p} \rightarrow (\alpha_j - \mathtt{p}^{op})$. Pelo exercício 2.f. e 2.g. do capítulo 4, temos que $\Gamma \vdash \neg(\alpha_i - \mathtt{p}) \rightarrow (\alpha_j - \mathtt{p}^{op})$. Ou seja $\Gamma \vdash (\alpha_i - \mathtt{p}) \cup (\alpha_j - \mathtt{p}^{op})$. Portanto, $\Gamma \vdash \alpha_k$.

Para $i = n$, temos $\Gamma \vdash \alpha_n$. Como $\alpha_n = \beta$, então $\Gamma \vdash \beta$. Portanto, $\Gamma \xrightarrow{*} \beta$ implica $\Gamma \vdash \beta$. ■

Teorema 8.8.16 (Teorema da Completude Forte) *Na lógica de predicados*

$$\Gamma \vdash \alpha \text{ se, e somente se, } \Gamma \models\!\!\approx \alpha \text{ se, e somente se, } \Gamma \xrightarrow{*} \alpha$$

DEMONSTRAÇÃO: Direto das proposições 8.8.13, 8.8.14 e 8.8.15. ■

8.9 Exercícios

1. Transforme as seguintes fbf's na forma normal prenex, elimine os quantificadores e transforme-as na forma normal conjuntiva reduzida.

   (a) $\forall x.(\neg P(x, y) \rightarrow \exists y.P(y, x))$

   (b) $\forall x.(P(z, x) \vee R(x, y)) \rightarrow (\exists x.P(z, x) \vee \forall x.R(x, y))$

   (c) $\exists x.P(x, y) \rightarrow \exists y.(P(y, x) \wedge R(a, f(x, y)))$

   (d) $\forall x.\exists y.(P(x, f(a, y)) \wedge \exists x.R(x))$

   (e) $\neg(\forall x.(P(x, y) \rightarrow \exists y.R(y, z)) \wedge \exists z.P(z, x))$

2. Determine o unificador mais geral para cada um dos seguintes pares de termos. Se estes não são unificáveis diga o porque.

a.	$P(x, f(a, x))$	$P(b, y)$
b.	$P(x, f(g(a, y), z))$	$P(b, f(g(a, f(w, c)), h(y, x)))$
c.	$f(a, f(b, f(c, x)))$	$f(a, y)$
d.	$f(x, f(a, f(y, c)))$	$f(z, f(z, f(f(a, c), w)))$

3. Exibir uma refutação por resolução para cada uma dos seguintes conjuntos de cláusulas. Algumas dicas:

   - Faça a separação padrão um par de cláusulas antes de unificar.
   - Indique cada substituição unificadora claramente.
   - Realize a substituição unificadora em ambas as cláusulas.
   - Não existe limite para a quantidade de vezes que se usa uma cláusula em particular.

   a. $P(a, x, x)$
   $P(f(y, x), w, f(x, z)) \vee \neg P(y, w, z)$
   $\neg P(f(a, f(b, a)), f(c, a), x)$

   b. $Q(a, b)$
   $Q(f(y, x), g(z)) \vee \neg Q(y, z)$
   $\neg Q(f(x, f(c, f(d, a))), w)$

   c. $R(x, a, a)$
   $R(x, f(x, y), z) \vee \neg R(x, y, z)$
   $R(x, f(y, z), f(y, w)) \vee \neg R(x, z, w)$
   $\neg R(b, f(b, f(c, f(b, a))), x)$

   d. $P(x, e, x)$
   $\neg P(y, z, v) \vee \neg P(y, v, w) \vee \neg P(e, z, w)$
   $P(a, f(u, v), e)$
   $\neg P(e, f(f(b, c), a), a)$

   e. $P(c, a, f(x))$
   $P(f(y), a, f(z)) \vee \neg P(y, x, z)$
   $\neg P(f(f(y)), z, f(f(f(z))))$

   f. $P(x, f(x, y), g(a))$
   $\neg P(a, x, z) \vee P(g(y), f(a, x), z)$
   $\neg P(g(z), f(a, f(a, a)), z) \vee \neg P(a, f(a, z), z)$

4. Mostre que $\Gamma \xrightarrow{*} \alpha$, quando:

   (a) $\Gamma = \{\neg P(x) \vee R(x, y), P(x)\}$ e $\alpha = R(x, y)$.

(b) $\Gamma = \{P(x,x), \neg P(x,z) \vee P(x,y) \vee P(y,z)\}$ e $\alpha = P(a,a)$

5. Dê uma refutação da negação da fórmula

$$\forall x.\forall y.\neg(P(y,x) \rightarrow \exists z.\neg(P(f(x),f(z)) \rightarrow \forall x_1.\neg P(x_1,x))) \rightarrow \forall x_2.\neg P(f(f(x_2)),x))$$

Capítulo 9

Programação em Lógica e Prolog

Os paradigmas funcional e imperativo de programação se baseiam na noção de que um programa implementa um mapeamento ou função. Numa linguagem imperativa, o programa tipicamente é um comando que lê entradas de arquivos e imprime saídas de arquivos. As saídas são funcionalmente dependentes das entradas, desse modo o programa pode ser visto abstratamente como implementando um mapeamento de entradas em saídas. Em linguagens funcionais, um programa é tipicamente uma função, e portanto obviamente implementa uma aplicação. Já a programação em lógica se baseia na noção de que um programa implementa uma relação, em vez de um mapeamento. Como relações são mais gerais que funções, programação em lógica é potencialmente de mais alto nível que programação funcional e imperativa.

A lógica, além de ser um formalismo propício para representar de maneira declarativa o conhecimento sobre alguma realidade, pode ser executada, no sentido que podemos escrever programas que representem o conhecimento usando o formalismo de um subconjunto da lógica de predicados (notação clausal) como linguagem de representação, podem-se inferir novos fatos e conhecimentos usando deduções através de alguns métodos de prova automática de teoremas.

Quando se usa lógica como formalismo de representação, a resolução de problemas se encara como demonstração de teoremas. O principio de resolução e o algoritmo de unificação de Robinson, descrito no capítulo anterior, fornecem a base para execução de programas em lógica. Um programa em lógica é um modelo ou descrição de um problema ou situação através de um conjunto finito de sentenças lógicas. Assim, ao contrário de programas procedurais ou imperativos, por exemplo escritos em C++ ou Java, um programa em lógica não descreve um procedimento (uma seqüência de passos ou comandos computacionais) para resolver um problema, mas o próprio problema. Este tipo de abordagem, conhecido como programação declarativa, não é exclusivo do paradigma lógico de programação, pois o paradigma funcional descreve um problema em termos de definições e avaliações de funções, pelo que também pode ser considerado como programação declarativa.

Um programa em lógica pode ser visto como uma base de dados mais complexa, onde

não só se têm dados do tipo: "Luiz esta cursando a disciplina lógica aplicada à computação", mas também regras indicando como deduzir novos fatos, por exemplo "todo aluno que esta cursando lógica aplicada à computação já cursou álgebra universal". Claramente, de essa informação e dessa regra podemos deduzir que "Luiz já cursou algebra universal", sem precisar incluir explicitamente esta informação no programa. A execução de um programa em lógica é realizada através de consultas ao programa, isto é, afirmações exprimindo condições que devem ser satisfeitas por um resposta correta, com respeito à informação contida no programa. A maioria dos sistemas desenvolvidos para programação em lógica encontrão as respostas das consultas ao programa através de uma pesquisa de refutações da negação da consulta.

Assim, o formalismo lógico permite emular sistemas baseado em regras, com encadeamento para adiante ou para atrás, e objetos estruturados. Pode-se também, com maior dificuldade, manipular valores por *default* e exceções, assim como quantificar existencial e universalmente a expressões. Fundamentalmente, um programa em lógica pode ser visto como uma teoria de verdade que dá fundamento sólido a todo o sistema. Sintetizando, programação em lógica é lógica em ação, onde as computações são realizadas como deduções.

Na atualidade existem diversas linguagens de programação com esta filosofia, chamada paradigma da programação em lógica. As mais conhecidas são Prolog, Gödel [HL94] e Isabelle [Pau94, NP01].

Aqui descreveremos a linguagem de programação Prolog, o qual é uma *ferramenta* usada para escrever programas baseados no raciocínio lógico (clássico) que são executáveis pelo computador. Prolog é um acrônimo de *PROgramming in LOGic* (programação em lógica). Assim, Prolog, é uma linguagem de programação baseada em lógica formal. Muitos programas de inteligência artificial são escritos em Prolog, pois esta linguagem permite, de maneira natural, a representação e dedução de conhecimento.

O Prolog foi implementado e projetado por Alain Colmerauer e seu Grupo de Inteligência Artificial [CHRP73], na França em 1972, fruto dos trabalhos e idéias de Robert Kowalski e outros autores [Kow72, Kow74]. Mas esta linguagem só começou a cativar adeptos a nível mundial, quando Pereira, Pereira e Warren implementaram na Universidade de Edinburgh [PPW79] o primeiro compilador Prolog escrito em Prolog.

Prolog é uma linguagem de programação interativa baseado em lógica com facilidades para a manipulação simbólica e com um motor de inferência, que lhe permite responder automaticamente questões, não só referentes ao conhecimento que contém explicitamente cada programa, más também ao que se pode deduzir de ele.

Prolog utiliza um subconjunto da lógica de predicados de primeira ordem, chamado cláusulas de Horn, que se caracteriza por que cada regra pode conter nenhum, hum ou mais antecedentes unidos conjuntivamente, mais no máximo só pode ter um conseqüente.

As inferências se realizam mediante o princípio da resolução proposto por Robinson em [Rob65], utilizando o algoritmo da unificação.

Existem predicados predefinidos que implementam funções extra-lógicas. Estes predicados, embora fogem da proposta original de programação declarativa, são necessários para que Prolog seja uma linguagem de programação de utilização prática real.

Uma característica desta linguagem, assim como de diversas linguagens funcionais, é que programas nesta linguagem não são eficientes quando rodados em um hardware convencional, que foram projetados para linguagens procedurais, que são mais orientadas ao cálculo numérico.

Muitos programas de inteligência artificial são escritos em Prolog, pois esta linguagem permite, de maneira natural, a representação e dedução de conhecimento.

Neste capítulo daremos uma introdução a esta linguagem, cobrindo as bases do Prolog, de modo que, o estudante, com ajuda de um manual do usuário, seja capaz de escrever e executar programas em algum dos tantos compiladores Prolog.

9.1 Conceitos Básicos

9.1.1 Cláusulas de Horn

Vimos que toda fbf do cálculo de predicado é equivalente a uma fbf na forma normal prenex e a parte livre de quantificadores dela é equivalente a uma fórmula na forma normal conjuntiva reduzida. A eliminação de quantificadores não gera uma fórmula equivalente, mas a fórmula original é uma contradição se, e somente se, a fórmula resultante do processo de skolemização também é uma contradição. Consideramos então, esta fórmula bem formada como um conjunto (entendido, por convenção, como sendo uma conjunção) de cláusulas, cada uma das quais é um conjunto (entendido como uma disjunção) de literais.

Definição 9.1.1 *Uma* **cláusula de Horn** *é uma cláusula com no máximo um literal positivo.*

É conveniente, de um ponto de vista de programação, escrever cláusulas como implicações em vez de disjunções. Assim a cláusula

$$\neg\alpha_1 \vee \cdots \vee \neg\alpha_n \vee \beta_1 \vee \cdots \vee \beta_k$$

seria escrita

$$(\alpha_1 \wedge \cdots \wedge \alpha_n) \rightarrow (\beta_1 \vee \cdots \vee \beta_k)$$

E uma cláusula de Horn, como

$$\neg\alpha_1 \vee \cdots \vee \neg\alpha_n \vee \beta$$

seria escrito

$$(\alpha_1 \wedge \cdots \wedge \alpha_n) \rightarrow \beta$$

Portanto, uma cláusula de Horn é uma implicação cujo antecedente é uma conjunção de fórmula atômicas e cujo conseqüente consiste de no máximo uma fórmula atômica. Existem quatro tipos de cláusulas de Horn

1. Aquelas com uma conclusão mas sem nenhuma condição;
2. Aquelas com condições e conclusão;
3. Aquelas com condições mas sem nenhuma conclusão;
4. A cláusula vazia, escrito $\square$, como anteriormente.

No que segue sobre programação em lógica, usaremos convenções da sintaxe Prolog. Isto nos permitirá usar nomes para predicados, funções, constantes e variáveis mais significativos (mnemônicos).

Símbolos de predicados, de funções e constantes iniciarão com letras minúsculas (número podem também ser usados como símbolos de constantes), variáveis começarão com letras maiúsculas. Escreveremos, também, implicações na cláusula de Horn para trás (isto é, a conclusão será colocada primeiro, e usaremos :-, em vez de $\leftarrow$ para separar a conclusão das condições das implicações).

9.1.2 Características Básica do Prolog

Comparada com as linguagens algorítmicas clássicas, tais como Pascal, C e JAVA, que são antes de tudo projetadas para computações numéricas, escrever sistemas, etc., uma linguagem para programação em lógica tem duas principais diferenças.

- Ela é orientada para "computação simbólica": os dados básicos são "símbolos" os quais somente representam eles próprios e não são interpretados na linguagem.
- Ela não especifica o algoritmo diretamente para ser executado e produzir resultados, mas especifica objetos, propriedades dos objetos e relações entre eles. Fala de *fatos*, *regras* para derivar novos fatos, e *consultas* (queries em inglês) para constatar se um fato foi estabelecido ou não.

9.2 Estrutura Básica do Prolog

As três espécies de *sentenças*: fatos, regras e questões podem ser lidas como predicados em lógica e, neste caso, a linguagem é um subconjunto da lógica formal. Ao mesmo tempo, Prolog é uma linguagem de programação: para obter a resposta a uma *consulta* é executado um algoritmo numa ordem precisa. No que segue, apresentaremos esses dois aspectos: o *declarativo* da lógica formal e o *procedural* das linguagens de programação.

Prolog é interessante por ser, de alguma maneira, lógica em ação. Ela executa fórmulas lógicas. Considere, por exemplo, os dois axiomas

"0 é um número natural"

"se n é um número natural, então $s(n)$ é um número natural"

Um programa Prolog pode implementar esses axiomas da seguinte maneira

```
natural(0).
natural(s(n)) :- natural(n).
```

Este programa é capaz de gerar e reconhecer números naturais, dependendo da questão:

```
?- natural(s(s(s(0)))).
        ⟶ sim
?- natural(–).
        ⟶ sim
?- natural(Prolog).
        ⟶ não
?- natural(X).
        ⟶ X=0;
        ⟶ X=s(0);
        ⟶ X=s(s(0));
        ⟶ X=s(s(s(0)));
          ⋮
```

A primeira consulta é "s(s(s(0))) é um número natural"?, a segunda é "existe um número natural"?, a terceira "Prolog é um número natural? e a última é "O que é um número natural"?. Se quisermos que s(s(s(0))) seja escrito 3, temos de programá-lo para tal.

Um programa realizando diferenciação simbólica de polinômios pode conter o fato

```
deriv(X,X,1)
```

para representar que a derivada de x com respeito a x é 1. Ele pode conter as regras:

```
deriv(U+V,X,A+B) :- deriv(U,X,A),deriv(V,X,B).
deriv(U-V,X,A-B) :- deriv(U,X,A),deriv(V,X,B).
```

A primeira dessas duas regras diz que "a derivada de U+V com respeito a X é A+B se A é a derivada de U e B é a derivada de V". Tais sentenças definem o ponto de vista Prolog da relação ternária "a derivada de $\cdots$ com respeito a $\cdots$ é $\cdots$". Este é o aspecto lógico da linguagem, ou sua leitura declarativa. Já no aspecto procedural de Prolog, se descrevem computações a serem executadas. Assim, a primeira regra diz que "para obter a derivada de U+V, primeiro se deve computar a derivada de U e V, dando A e B respectivamente, e então construir o termo A+B".

As razões para essa dupla leitura é a seguinte: Um sistema Prolog é um provador de teoremas explorando os aspectos lógicos da linguagem. O sistema tem de mostrar uma contradição entre a negação da questão e o conjunto de fatos e regras. Esta refutação, quando bem sucedida, exibe contra-exemplos que são as respostas à questão. A estratégia usada para achar é simples.

9.2.1 Termos e objetos

Em Prolog, assim como em lógica, objetos num "universo de discurso" (que é o conjunto dos objetos considerados) são representados por *termos*. Como a linguagem não interpreta esses termos, podemos dizer, também, que os objetos tratados por Prolog são termos, cada termo sendo um objeto simbólico pertencente a uma das seguintes categorias:

- constantes individuais ou átomos,
- variáveis,
- funções ou termos funcionais consistindo de um nome de função e uma lista de argumentos, que são termos.

A sintaxe deve permitir classificarmos um termo na sua categoria, especialmente variáveis como distintas de constantes. A sintaxe usada depende de cada implementação do Prolog. Aqui usaremos

- Um átomo pode ser escrito de uma das três maneiras:
  - como um identificador começando com uma letra minúscula, podendo conter subtraços:

pedro henrique_iv.

– como um número

321 3.21

– como uma seqüência de caracteres sob aspas simples

"quem é voce?" "eu não sei"

- Uma variável é um identificador começando com uma letra maiúscula, podendo conter sub-traço

X Nome Rei_da_Bélgica

- Uma função é um nome de função seguido por uma lista de termos entre parênteses, onde um nome de função tem a forma de um átomo não-numérico:

autor_de(scott) f(x,y,z) "oi!"(pedro)

Um programa em Prolog é usualmente um modelo de alguma parte da realidade cujos termos são usados para representar objetos reais. Por exemplo, para modelar uma biblioteca, os átomos podem representar nomes de autores e títulos de livros, termos funcionais podem representar livros e edições:

edição(wiley,1987)

livro(davis,the_undecidable,edição(raven,1965))

Portanto, os átomos representam objetos que são considerados indivíduos elementares. É prática normal usar identificadores mnemônicos, isto é, nomes que lembram os objetos que eles representam, tais como wiley, davis, em vez de nomes anônimos como x_{22} ou bbb. O mesmo podemos dizer para termos funcionais: edição(wiley,1987) e não e(w,1987). Deve-se observar que as funções de Prolog são vistas como estruturas de dados compostas, similares aos registros em Pascal, e conseqüentemente, elas podem ser construídas e analisadas. Elas não são funções aplicadas a argumentos para denotar o resultado correspondente.

Termos em que não há variável são os dados constantes em Prolog. Em lógica eles são os **termos básicos** ou **termos constantes**. A menos dos átomos numéricos, os termos não são interpretados por Prolog, isto é, eles não representam os dados, eles *são* os dados.

9.2.2 O Escopo dos Identificadores

A menos de átomos numéricos, que podem ser interpretados como números nas computações numéricas, e tirando funções ou predicados construídos, que tem um significado pré-definido, os nomes em Prolog, para constantes, variáveis, funções e predicados, não tem nenhum significado intrínseco e podem ser escolhidos livremente pelo programador. No entanto, distinguimos variáveis de constantes em virtude da primeira letra (maiúscula no caso de variável e minúscula no caso de constante). Em geral duas notações distintas denotarão, ou serão, objetos distintos. Existem mais ou menos exceções óbvias, por exemplo, para números, onde 133.1 e 133.10 denotam o mesmo número.

Com em qualquer linguagem de programação, a cada nome tem de ser dado um *escopo*, que é a parte do programa onde o nome pode ser escrito para significar um objeto. Em Prolog, as regras de escopo são:

- O escopo de um variável é a sentença (fato, regra, ou consulta) na qual aparece.
- O escopo de qualquer outro nome (constante, nome de função, ou nome de predicado) é todo o programa.

Isto significa que um nome de variável pode ser usado e reusado a vontade no programa para denotar variáveis diferentes, enquanto qualquer outra notação representa, ou é, o mesmo objeto para o programa todo.

No fragmento da base de dados de uma família reproduzida abaixo, as ocorrências de mãe são o mesmo predicado (uma vez que existem dois argumentos nos quatro casos). Por outro lado, três variáveis aparecem duas vezes em cada regra, mas, como a intuição sugere, as variáveis de uma regra são diferentes daquelas da outra.

```
mãe(vera,lúcia).
mãe(lúcia,alice).
avô(X,Y) :- pai(X,Z),mãe(Z,Y).
avó(X,Y) :- mãe(X,Z),pai(Z,Y).
```

9.3 Fatos Elementares

Os predicados simples (ou fórmulas atômicas) da lógica denotam valores verdade. Por exemplo, o predicado

```
autor_de(pablo_neruda,canto_geral).
```

denota um valor verdade. Este, também, é o caso em Prolog, ao menos aproximadamente. Fatos e questões sem variáveis podem ser vistos como significando "verdade" ou "falso". Aproximadamente, porque Prolog, sendo uma linguagem de programação, tem

uma semântica operacional, na qual "verdade" e "falso" são antes de tudo *resultados* de programas executados.

Um predicado simples é escrito em Prolog como um termo funcional:

```
autor_de(pablo_neruda,canto_geral).
deriv(X,X,1).
```

É o contexto que determina se uma fórmula atômica é para ser tratada como um termo funcional ou como um predicado. Predicados simples são os blocos construtores usados para escrever fatos e questões.

O programa seguinte é uma lista de fatos

```
/* fatos */
tem(emília,bicicleta).              /* "Emília tem uma bicicleta" */
gosta(emília,maria).                /* "Emília gosta de Maria" */
tem(joão,bicicleta).
livro(davis,the_undecidable,edição(raven,1965)).
chove.
```

Os fatos são predicados simples seguido por um ponto. O par /*···*/ encerra um *comentário*, isto é, um texto que é ignorado quando o programa for executado.

Cada fato que aparece num programa estabelece um valor verdade, uma relação entre objetos estabelecida como verdadeira. Por exemplo, a relação binária (ou predicado) chamada *gosta* é estabelecida entre emília e maria, livro é uma relação 3-ária, isto é, um predicado com três argumentos, e chove uma relação 0-ária. Um conjunto de fatos pode ser entendido como um banco de dados relacional. De fato, o conjunto de fatos e regras que constituem um programa Prolog será denominado, aqui, uma *base de dados*.

9.4 Consultas

Predicados simples também servem na construção de *consultas* (ou questões, perguntas ou metas) tais como

```
?- tem(joão,bicicleta).
```

Isto não estabelece qualquer fato novo, mas é uma "pergunta ao sistema", indagando se ou não um fato foi estabelecido. Neste caso (uma questão simples, nenhuma variável, nenhuma regra), o valor é verdade se o fato feito de um predicado da pergunta está na base de dados, caso contrário o valor é falso. Isto é o que intuitivamente se espera e o sistema responde sim ou não de acordo com a base de dados. Na prática, a seqüência pergunta-resposta dá:

```
?- tem(joão,bicicleta)
      ⟶ sim
```

Assim, na sua forma mais simples, um programa Prolog registra fatos elementares e responde consultas simples sobre eles. A base de dados é a definição extensional das relações que ele contém, aqui, tem(-,-), gosta(-,-), livro(-,-,-), e chove.

Consultas simples contendo somente termos básicos (as constantes da linguagem) são perguntas do tipo sim-não, isto é, somente tem duas possíveis respostas: sim se o fato está na base de dados e não se o fato não está na base de dados.

É bom deixar claro que a semântica de uma consulta é definida com respeito a uma dada base de dados, num dado momento: uma consulta simples indaga se um fato foi estabelecido ou não, que pode ser o seu significado. Por exemplo, para a base de dados fatos, obteríamos a seguinte seqüência:

```
?- tem(joão,bicicleta)
      ⟶ sim
?- tem(pedro,bicicleta)
      ⟶ não
?- tem(joão,biiccleta,)
      ⟶ não
?- chove(agora)
      ⟶ não
```

Como Prolog não tem controle de declaração nem de tipo, as últimas duas perguntas não são erros Prolog, mesmo que eles sejam provavelmente um erro de programação. Um nome faltando ou mudando uma ou várias letras é exatamente um outro átomo simbólico e o mesmo átomo simbólico pode nomear relações que diferem pelo número de argumentos. Portanto, para Prolog, simplesmente não existem fatos na base de dados casando com as três últimas perguntas.

9.4.1 Conjunções

Fórmulas lógicas são escritas com predicados simples e conectivos. Este, também, é o caso em Prolog, sendo que a construção mais freqüente é a conjunção, denotada por vírgula:

```
?- tem(joão,bicicleta), tem(emília,bicicleta).
```

Essa conjunção usada, aqui, numa consulta, é mais ou menos equivalente à fórmula lógica

tem(joão,bicicleta)∧tem(emília,bicicleta).

Em lógica, uma conjunção é verdadeira se todas as suas componentes são verdadeiras. Isto, também, acontece em Prolog, com uma diferença importante: a ordem na qual as componentes são consideradas, da esquerda para a direita, faz parte da semântica.

9.5 Variáveis

O uso das variáveis em Prolog é análogo, mas não equivalente a seu uso em lógica. As consultas em Prolog vistas até aqui contém somente termos básicos, isto é, não contém variáveis, e tem somente uma de duas possíveis respostas: sim ou não. Consultas mais interessantes são aquelas cujas respostas são listas de objetos satisfazendo uma dada relação. Por exemplo, "quais as pessoas que possuem bicicleta?" é uma pergunta cuja resposta é uma lista de objetos. Relativamente à base de dados "fatos", a resposta completa para esta questão seria a lista [emília,joão]. Em Prolog, este tipo de consultas são expressas com a ajuda de variáveis. Um pronome como "quais" é expresso por uma variável como X, e a pergunta acima pode ser escrita

```
?- tem(X,bicicleta).
```

Intuitivamente, obteríamos uma resposta substituindo a variável X por uma constante que torna o termo assim obtido igual a um fato da base de dados. Por exemplo, emília é uma tal constante.

```
?- tem(X,bicicleta).
      ⟶ X=emília
```

O sistema responde com um valor para a variável que transforma a consulta em um predicado verdadeiro. Por que emília e não joão ou [emília,joão]?. Foi feita uma escolha mais ou menos arbitrária: somente é produzida a primeira resposta, o significado de "primeira" depende do algoritmo que computa a resposta.

A base de dados não é um conjunto, mas uma *lista ordenada* de sentenças (fatos e regras). Para responder a uma consulta, o sistema pesquisará a base de dados nesta ordem, usualmente a ordem textual na qual sentenças entraram, para achar a primeira sentença que satisfaz o predicado na pergunta. Este predicado, que aqui é tem(X,bicicleta), é chamado o *meta*. O meta é bem sucedido se uma sentença na base de dados satisfaz seu predicado. Um fato que satisfaz um simples predicado como tem(X,bicicleta) deve ter a forma tem(–,bicicleta), isto é, deve ter

- o mesmo nome de predicado
- o mesmo número de argumentos, dois,
- as mesmas constantes nos mesmo lugares, bicicleta como primeiro argumento.

Quando essas três condições são satisfeitas, dizemos que o fato e o predicado se *casam*. Tal fato aqui, é

tem(emília,bicicleta)

Então a variável X da questão, toma como valor (é *instanciado* por) o termo básico que está no lugar no fato. Este dá o resultado esperado:

⟶ emília.

Uma outra maneira de expressar a semântica desta pergunta é "pode uma atribuição a X ser encontrada tal que a fórmula resultante está na base de dados?". A atribuição

X=emília

atende esse requisito.

Prolog produz somente, uma resposta, mesmo se várias são possíveis, e a escolha dela é uma decisão arbitrária que dependerá ao modo iterativo que nosso Prolog trabalha. O usuário que quer mais tem de dizê-lo, digitando um ponto e vírgula. Isto diz ao sistema que uma outra resposta está sendo esperada:

```
?- tem(X,bicicleta).
      ⟶ X=emília;
      ⟶ X=joão;
      ⟶ não
```

A última resposta, não, significa que não existe mais casamento na base de dados.

Se considerarmos a pergunta

?- tem(X,bicicleta).

como uma chamada de procedimento que tem de produzir um resultado, então a constante bicicleta é o argumento da chamada (o parâmetro de entrada) e a variável X representa o resultado (o parâmetro de saída). Este ponto de vista é do usuário e não do Prolog. Na sentença na base de dados, não existe diferença entre os argumentos. Quais são os valores que devem aparecer na resposta é indicado somente pelas variáveis presentes na consulta. Por exemplo:

```
?- tem(joão,X).
      ⟶ X=bicicleta.
?- tem(X,Y).
      ⟶ X=emília, Y=bicicleta;
      ⟶ X=joão, Y=bicicleta;
      ⟶ não
```

Variáveis podem ser usadas, também, em fatos:

gosta(X,pera).

poderia ser uma tradução em Prolog da sentença "todos gostam de pera" e da fórmula lógica

$$\forall x.gosta(x, pera).$$

Qualquer consulta da forma ?- gosta(–,pera) será satisfeita se a base de dados contém o fato gosta(X,pera). Quando usado numa conjunção, uma variável estabelece uma ligação entre os predicados na conjunção (tomando os mesmos valores em todas suas ocorrências). A pergunta "quem são os proprietários de bicicleta que gostam de Maria?" torna-se:

?- tem(X,bicicleta), gosta(X,maria).

A resposta a tal pergunta é computada satisfazendo as duas partes (submetas) da conjunção. A primeira submeta,

tem(X,bicicleta),

é bem sucedido para o fato

tem(emília,bicicleta)

com X tomando o valor emília (no caso de falha, a resposta a toda a questão teria sido "não"). Então, um casamento para o segunda submeta é transformado em

gosta(emília,maria)

Um casamento para a meta é achado e a primeira resposta é

⟶ X=emília

Se o usuário digitar ponto e vírgula para perguntar por um outro valor para X, a pesquisa começará no lugar onde o último casamento foi encontrado, a meta sendo o mesmo, isto é,

gosta(emília,maria),

que não pode ser satisfeito uma segunda vez. O sistema então reverte para a submeta imediatamente precedente que é

tem(X,bicicleta).

Isto é, *backtracking*. A submeta é bem sucedida uma segunda vez, por

tem(X,bicicleta),

dando a X o valor joão. Agora, a segunda submeta torna-se

gosta(joão,maria)

e isto falha. A próxima resposta é não.

9.5.1 Variáveis Anônimas

Todas as variáveis usadas até aqui tiveram um nome como X ou Y. Existem, também, variáveis *anônimas*, todas representadas por um subtraço. Se desejamos saber na base de dados fatos se existem possuidores de bicicleta, a pergunta

?- tem(X,bicicleta).

vista anteriormente, produzirá sucessivamente emília e joão como respostas, embora gostássemos de um simples sim ou não. Isto pode ser obtido trocando X por uma variável anônima. A pergunta

?- tem(–,bicicleta).

produz a resposta

⟶ sim

Do ponto de vista de casar uma meta, uma variável anônima comporta-se como qualquer variável e pode casar com qualquer termo. A diferença é que o termo casado não é atribuído a ela. Uma primeira conseqüência é que diversas ocorrências do símbolo – numa consulta denotam diferentes variáveis anônimas: eles podem casar diferentes termos. Uma segunda conseqüência é que nenhum valor para variáveis anônimas aparecerá na saída.

A seguinte pergunta produzirá uma lista de possuidores de algum objeto

```
?- tem(-,X).
     ⟶ X=emília;
     ⟶ X=joão;
     ⟶ não
```

9.6 Regras

A terceira e última espécie de sentença Prolog é a regra, que generaliza fatos e dá potência à linguagem, tornando a base de dados dedutiva.

Por exemplo, para construir uma base de dados sobre famílias, podemos modelar algumas relações familiares por fatos, tais como:

```
mãe(rossana,gabriela).
mãe(gabriela,aline).
pai(roberto,gabriela).
pai(edmundo,aline).
```

para representar as relações familiares "Rossana é mãe de Gabriela", "Gabriela é mãe de Aline", "Roberto é pai de Gabriael" e "Edmundo é pai de Aline".

Não parece razoável continuar e representar, por exemplo, os fatos

```
avó(rossana,aline).
avô(roberto,aline).
```

uma vez que tais relações podem ser deduzidas da relação de parentesco: "X é avó de Y se existe Z cuja mãe é X e que é mãe ou pai de Y" e "X é avô de Y se existe Z cuja pai é X e que é mãe ou pai de Y" Tais sentenças são modeladas em Prolog pelas regras:

```
avó(X,Y) :- mãe(X,Z),mãe(Z,Y).
avó(X,Y) :- mãe(X,Z),pai(Z,Y).
avô(X,Y) :- pai(X,Z),mãe(Z,Y).
avô(X,Y) :- pai(X,Z),pai(Z,Y).
```

onde o símbolo :- pode ser lido com "se". É claro que embora tenhamos introduzidos quatro regras para deduzir os dois fatos acima, estamos economizando memória. Pois, generalizando os parentescos "avô" e "avó" em bases de fatos onde existam diversos fatos envolvendo este parentesco, ainda só precisaremos destas quatro regras para deduzir todos estes parentescos.

A forma geral para uma regra é

```
C :- P1,P2,···,Pn.
```

onde C,P_1,P_2,$\cdots$,P_n são predicados simples. Isto pode ser lido como "C se (P_1 e P_2 e $\cdots$ e P_n)". O predicado C é a *cabeça* ou conclusão da regra, enquanto a seqüência de

predicados P_1,P_2,$\cdots$,P_n é o *corpo* da regra ou conjunto de premissas. Tal regra reflete uma cláusula de Horn.

Essas são maneiras diferentes de escrever cláusulas de Horn. Em termos de linguagem de programação, uma regra é análogo à declaração de um procedimento. A cabeça de uma regra é estabelecida como significando o mesmo que o corpo das regra. Por exemplo, nas regras para representar o parentesco "avó", o corpo

avó(X,Y),

significa

mãe(X,Z),mãe(Z,Y).

ou

mãe(X,Z),pai(Z,Y).

Em outras palavras, regras como esta indicam que uma questão tal como

?- avó(ana,amanda).

é para ser transformada nas questões

?- mãe(ana,Z),mãe(Z,amanda).

?- mãe(ana,Z),pai(Z,amanda).

Numa base de dados, regras são consideradas do mesmo modo que fatos: suas ocorrências são para estabelecer suas verdades, e são usadas pelo sistema para dedução. De um ponto de vista lógico, variáveis em regras são vistas como quantificadas universalmente. Assim, a regra

avó(X,Y) :- mãe(X,Z),mãe(Z,Y).

pode ser interpretada como a fórmula lógica

$$\forall x \forall y \forall z (mãe(x,z) \wedge mãe(z,y) \longrightarrow avó(x,y))$$

Regras são as sentenças mais gerais. Um fato pode ser considerado uma regra sem o lado direito, enquanto uma consulta pode ser considerada uma regra sem um lado esquerdo. Elas podem ser lidas, respectivamente, como C é verdadeira se P_1 e P_2 e $\cdots$ e P_n são verdadeiras (regra com cabeça e corpo), F é verdadeira (fato ou regra sem corpo), são P_1 e P_2 e $\cdots$ e P_n verdadeiras? (consulta ou regra sem cabeça).

Consideremos a seguinte base de dados:

```
        /* Base de Dados Família */

/* mãe(X,Y) denota "X é mãe de Y" */
/* pai(X,Y) denota "X é pai de Y" */
/* avô(X,Y) denota "X é avô de Y" */
/* avó(X,Y) denota "X é avó de Y" */
/* feminino(X) denota "X é do sexo feminino" */
/* masculino(X) denota "X é do sexo masculino" */

mãe(claudilene,amanda).
mãe(rossana,gabriela).
mãe(gabriela, aline).
pai(roberto,amanda).
pai(roberto,gabriela).
pai(mario,rossana).
pai(mario,joão).
avô(X,Y) :- pai(X,Z),mãe(Z,Y).
avô(X,Y) :- pai(X,Z),pai(Z,Y).
avó(X,Y) :- mãe(X,Z),pai(Z,Y).
avó(X,Y) :- mãe(X,Z),mãe(Z,Y).
feminino(X) :- mãe(X,–).
feminino(aline).
feminino(amanda).
masculino(X) :- pai(X,–).
masculino(joão).
```

Nesta base de dados, as relações mãe e pai são definidas (em extensão) por fatos, enquanto as relações avô e avó são definidas (em intensão) por regras. Uma resposta à pergunta

```
?- avô(roberto,aline).
```

não pode ser encontrada entre os fatos na base de dados, de vez que nenhum fato casa com ele, mas é encontrada entre as regras onde algumas cabeças de regra casa com ela.

Uma pergunta pode ser casada pela cabeça de uma regra do mesmo modo que ela casa com um fato: mesmo nome de predicado, mesmo número de argumentos, mesmas constantes nos mesmos lugares. Assim, o predicado

```
?- avô(roberto,aline).
```

é casado com

avô(X,Y).

Este casamento associa valores às variáveis na cabeça da regra:

X=roberto, Y=aline.

Então o corpo da regra, na qual as variáveis tomaram novos valores, tornam-se a nova meta a ser satisfeita, neste caso:

?- pai(roberto,Z),mãe(Z,aline).

Este meta é uma conjunção de predicados, a ser tratada como uma seqüência de submetas. Eles são satisfeitos pela atribuição

Z=gabriela

e a resposta à pergunta é sim.

O valor de uma variável interna, isto é, uma variável tal como X, Y e Z, que não aparecem na pergunta, nunca são impressos. Aqui a pergunta pede uma resposta sim-não, as variáveis internas servem somente para computar esta resposta.

Observe que no caso da pergunta

?- avó(rossana,aline)

será casado com

avó(X,Y).

na regra

avó(X,Y) :- mãe(X,Z),pai(Z,Y).

Este casamento associará os seguintes valores às variáveis X e Y:

X=rossana, Y=aline.

em seguida o corpo da regra com os valores de X e Y já instanciados será a nova meta, isto é

?-mãe(rossana,Z),pai(Z,aline).

como não existe em família nenhum valor para Z satisfazendo esta pergunta, o Prolog casará a meta original com a regra

```
avó(X,Y) :- mãe(X,Z),mãe(Z,Y).
```

Assim, a ordem em que aparecem as regras e fatos num programa Prolog é importante na execução do programa.

9.6.1 Regras Recursivas

Uma base de dados sobre relações familiares deveria ser capaz de responder questões sobre ancestralidade. O predicado "X é ancestral de Y" tem, necessariamente, uma definição recursiva. Em Prolog, ancestralidade pode ser definido pelas regras

```
ancestral(X,Y) :- mãe(X,Y).
ancestral(X,Y) :- pai(X,Y).
ancestral(X,Y) :- mãe(X,Z),ancestral(Z,Y).
ancestral(X,Y) :- pai(X,Z),ancestral(Z,Y).
```

Regras recursivas como estas devem ser permitidas, caso contrário a linguagem não seria útil para muitas aplicações.

Qualquer predicado definido por uma regra recursiva deve, é claro, ter no mínimo uma definição não recursiva. Se isso não acontecesse, a definição seria logicamente mal-formada e o programa correspondente ficaria em círculo (laço infinito). Porém isto não é suficiente para evitar que o programa fique em laço infinito. Devemos também tomar cuidado com a ordem na qual casamentos são procurados para metas. Por exemplo se invertermos a ordem nas regras recursivas do predicado ancestral, isto é

```
ancestral(X,Y) :- ancestral(Z,Y),mãe(X,Z).
ancestral(X,Y) :- ancestral(Z,Y),pai(X,Z).
```

o programa entrará em um laço.

9.7 Disjunções e Negações

Usando somente conjunções para compor consultas e regras (cláusulas de Horn) nós dá uma espécie de Prolog *puro*, que pode não ser muito prático. Em muitos casos, são fornecidas, também, disjunções e negações, com semânticas refletindo a lógica.

9.7.1 Disjunção

A disjunção em regras pode ser usada implicitamente em Prolog puro usando várias regras com a mesma conclusão. Por exemplo para dizer que uma pessoa é filho de alguém se esse alguém é o seu pai ou sua mãe, pode ser expressado através das regras:

```
filho(X,Y) :- pai(Y,X).

filho(X,Y) :- mae(Y,X).
```

Já em consultas uma disjunção não poderia ser desdobrada, pois consultas são feitas através de uma única cláusula sem conseqüente. Assim, para facilitar as consultas um Prolog não puro permite o uso de disjunção em consultas, e usa o ponto e vírgula para representar disjunção. Por exemplo a consulta "João é pai de Maria ou de Pedro" pode ser feita da seguinte forma:

```
?- pai(joão,maria); pai(joão,pedro).
```

onde o ponto e vírgula representa disjunção, é aproximadamente o mesmo que a fórmula lógica

$$pai(joão, maria) \vee pai(joão, pedro)$$

Quando uma consulta é uma disjunção, o sistema primeiro tenta satisfazer a parte esquerda e, se este não foi bem sucedido, ele tenta então satisfazer a parte direita. é claro, podemos sempre procurar todos os casamentos possíveis. Com o exemplo fatos, isto dá:

```
?- tem(X,bicicleta); gosta(X,maria).
      ⟶ X=emília;
      ⟶ X=joão;
      ⟶ X=emília;
      ⟶ não
```

Observe a ordem na qual as respostas são produzidas para concluir que uma vez que a disjunção não está num predicado composto, as duas partes da disjunção se comportam como duas questões sucessivas.

Embora, não seja necessário incluir disjunções em regras, para facilitar a programação Prolog permitiremos o uso desta (também através do ponto e vírgula) em regras. Porém devemos ter cuidados especiais quando uma disjunção envolva conjunções. Nestes casos assumiremos que a ordem de precedência é a usual em lógica (conjunção é mais forte que disjunção). Caso desejarmos quebrar esta precedência devemos usar parênteses. Por exemplo uma regra para expressar a relação "X é avô de Y" é a seguinte

```
avô(X,Y) :- pai(X,Z),(pai(Z,Y);mae(Z,Y)).
```

Se não usarmos os parênteses, isto é

```
avô(X,Y) :- pai(X,Z),pai(Z,Y);mae(Z,Y).
```

qualquer pessoa (X) seria avô de qualquer pessoa (Y), desde que esse alguém (Y) tenha mãe.

9.7.2 Negação por Falha e Hipóteses do Mundo Fechado

Quando fazemos uma consulta a um programa Prolog, por exemplo

```
?- pai(X,maria).
```

e ele da como resposta "Não", não significa que Maria não tenha pai, mas que o programa não tem informações sobre o pai de Maria. Tal raciocínio é baseado na "hipóteses do mundo fechado", segundo a qual o mundo é fechado, no sentido que "tudo o que existe está ou pode ser deduzido do programa". Assim, se alguma coisa não está nem explicitamente (fato) nem implicitamente (derivada através de regras) no programa, então não é verdadeira. Portanto devemos ter isto em consideração ao momento de fazer consultas e fazer nosso programa Prolog. Uma maneira de aliviar isto seria usar um conectivo de negação para indicar que algo não é verdadeiro, mas isto tornaria o programa muito tedioso, pois deveríamos por todo o que um objeto não é, por exemplo na base família deveríamos pôr que Roberto não é pai de André, Vera, Alice, e até dele mesmo.

Nos usaremos um operador lógico de negação em Prolog, mas não no sentido acima, mas para auxiliar as consultas e regras (não em fatos!!) e assumindo a hipóteses de mundo fechado.

Negação é escrita como um predicado chamado *not* cujo único argumento é o predicado a ser negado. Se P é um predicado, a meta not(P) é bem sucedida se a meta P falhar e falha se o predicado P é bem sucedido. *Nenhuma variável é instanciada quando* not(P) *é bem sucedida*. De fato, instanciação de variáveis pode acontecer somente se P é bem sucedido.

```
?- not(tem(camilo,bicicleta)).
      ⟶ sim
?- not(tem(X,bicicleta)).
      ⟶ não
```

A primeira resposta diz que Camilo não tem uma bicicleta, já a segunda diz que existe alguém que tem uma bicicleta.

Este é somente uma explicação procedural do predicado not: uma meta negada é bem sucedida se seu complemento não pode ser satisfeito com respeito à base de dados. Esta

visão pragmática não é entretanto equivalente ao ponto de vista lógico teórico. De um ponto de vista lógico, Prolog manipula a negação de uma meta como uma falha de sua prova.

Em regras podemos usar este operador somente nos antecedentes. Por exemplo uma regra para descrever irmãos somente por parte de pai na base família seria a seguinte:

```
irmaosP(X,Y) :- pai(Z,X),pai(Z,Y),mae(M,X),not(mae(M,Y)).
```

No caso de se ter uma consulta ou regra envolvendo negações, conjunções e disjunções, o compilador Prolog usa a seguinte ordem de prioridade nos conectivos: primeiro negação, depois conjunção e por último disjunção. Caso deseje alterar a ordem deve usar parênteses.

9.8 Operadores de Controle

Até o momento nos conseguimos interferir no controle de execução de um programa Prolog através da reordenação das cláusulas ou dos antecedentes de uma regra. No entanto este controle é muito limitado. Nesta seção introduziremos dois operadores que interferiram no próprio processo de execução do programa abortando o processo de *backtracking*, aumentando a eficiência do programa.

9.8.1 Backtracking

Ao se fazer uma consulta a um programa Prolog, ela terá como respostas um simples "Sim", um "Não" ou valores para as variáveis da consultas. Estas respostas dependem se é possível deduzir no programa a cláusula em questão. Por simplicidade assumamos que o Prolog é puro. Assim uma consulta seria da forma: $? - p_1, \ldots, p_n$. Para o compilador Prolog determinar a resposta a essa pergunta tenta unificar p_1 à primeira regra ou fato do programa possível. Se for um fato então tenta agora responder p_2 e assim por diante. Se casar com o conseqüente de uma regra, digamos $p'_1 : -q_1, \ldots, q_k$ então o objetivo se transforma em $? - q_1\sigma, \ldots, q_k\sigma, p_2\sigma \ldots, p_n\sigma$, onde σ é o unificador mais geral entre p_1 e p'_1. Quando um objetivo falha, isto é, não conseguimos casar um dos sub-objetivos, então retornamos a ponto imediatamente anterior ao último casamento e avançamos no programa. Ao retornar pelo caminho já percorrido todo o trabalho executado é desfeito. A resposta só será "Não" quando todos os caminhos falharem.

Vejamos um exemplo para esclarecer melhor o processo de *backtracking*.

Seja a base de dados família. façamos a ela a seguinte consulta:

```
?- pai(roberto,X),mãe(rossana,X)
```

O compilador Prolog tenta primeiro satisfazer o primeiro objetivo, isto é pai(roberto,X), desencadeando a seguinte execução *top-down*.

1. Encontra que Roberto é pai de amanda.
2. Instância X com "amanda".
3. Tenta satisfazer o segundo objetivo com a variável X já instanciada, isto é mãe(rossana,amanda).
4. Como não consegue satisfazer este objetivo, realiza um *backtracking*, isto é volta à consulta original ?- pai(roberto,X),mãe(gabriela,X) a partir do último fato casado (pai(roberto,lúcia).).
5. Encontra que Roberto é pai de gabriela.
6. Instância X com "gabriela".
7. Encontra que mãe(rossana,gabriela).
8. Foi bem sucedido, pelo que a resposta é: X=gabriela.

Mas como o compilador Prolog acha que Roberto é pai de amanda? fazendo refutação. Primeiro negamos pai(roberto,X) e tentamos refutar o conjunto de cláusulas de Horn que compõem o programa Prolog acrescido desta cláusula. Existem basicamente duas refutações

$$\dfrac{\dfrac{\neg pai(roberto, X) \qquad pai(roberto, amanda)}{\sigma_1 = [(amanda, X)]}}{\neg pai(roberto, amanda) \qquad\qquad pai(roberto, amanda)}$$

$$\square$$

Figura 9.1: Árvore para a refutação da pergunta ?- pai(roberto,X).

$$\dfrac{\dfrac{\neg pai(roberto, X) \qquad pai(roberto, gabrela)}{\sigma_1 = [(gabriela, X)]}}{\neg pai(roberto, gabriela) \qquad\qquad pai(roberto, gabriela)}$$

$$\square$$

Figura 9.2: Segunda árvore para a refutação da pergunta ?- pai(roberto,X).

Mas, o compilador Prolog acha primeiro a refutação 9.1. Este mesmo processo é realizado para achar que Rossana é mãe de Gabriela.

Vejamos agora uma pergunta mais complexa:

?- avô(rossana,X),mãe(Y,X)

Primeiro tenta achar de quem Rossana é avó, no caso seria "Aline", através da refutação do programa Prolog acrescido da cláusula $\neg avoó(rossana, X)$. Este processo é ilustrado pela árvore 9.3 a seguir. Observe que foi feita uma separação padrão das variáveis antes de fazer a refutação.

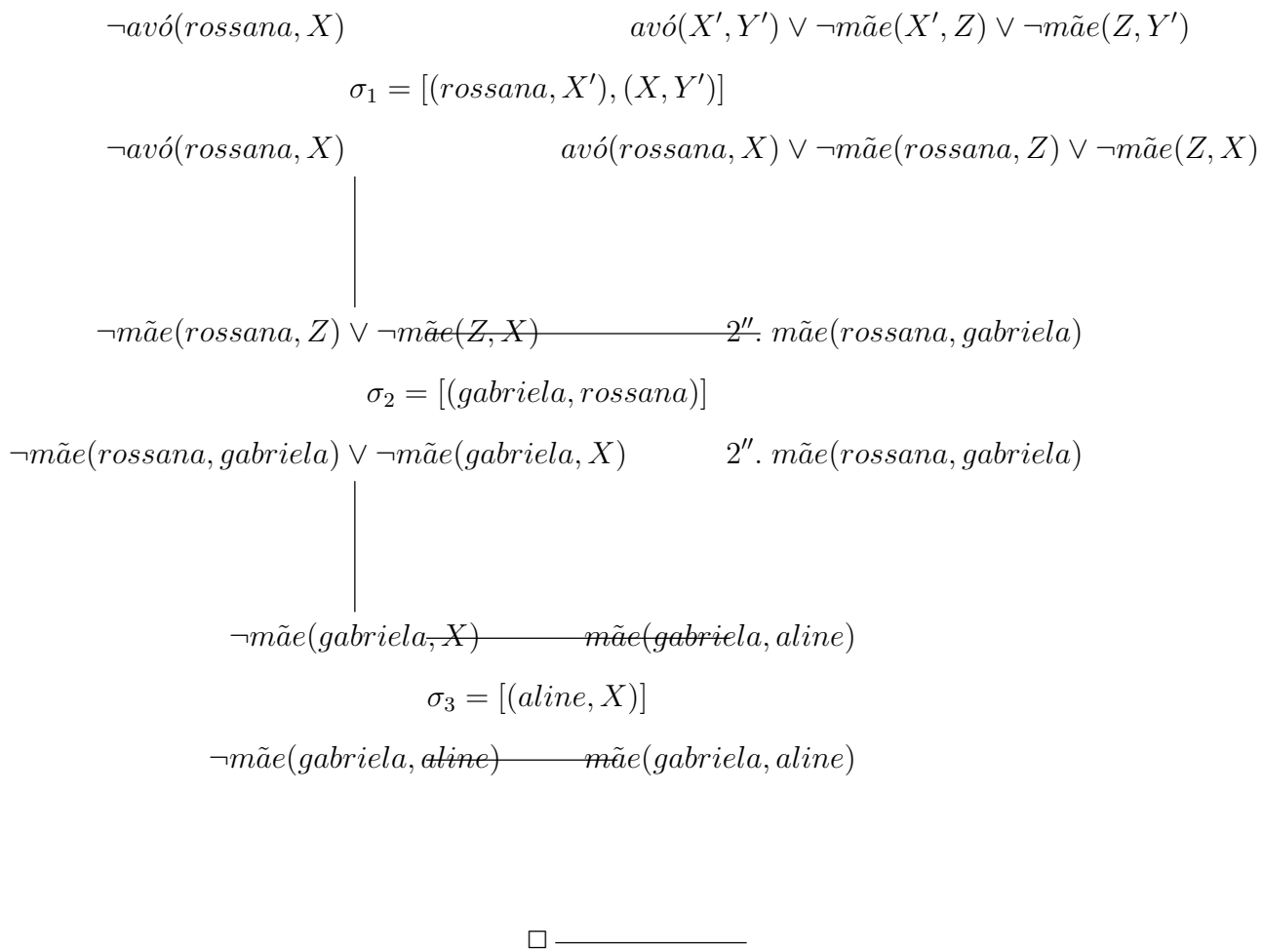


Figura 9.3: Árvore para a refutação da consulta ?- avô(rossana,X).

Assim, σ_3 diz que aline é atribuído à variável X, em seguida essa instância de X é considerada na segunda parte da pergunta, isto é em mãe(Y,X), ou seja agora seria indagado quem é mãe de Aline, cuja resposta seria "Gabriela". Esta resposta poderia ser achada usando a refutação apresentada na figura 9.4.

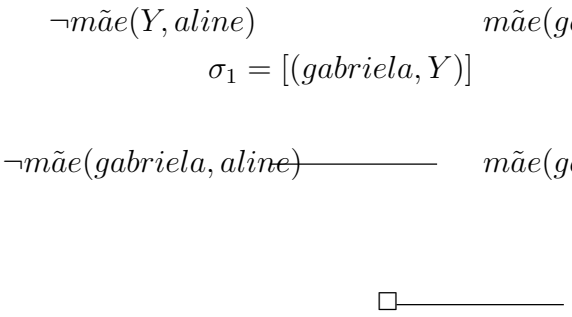


Figura 9.4: Árvore para a refutação da pergunta ?- mãe(Y,aline).

No caso de um Prolog não puro, é possível considerar também disjunções e negações, nesses casos procede-se da seguinte maneira: Um programa Prolog satisfaz uma consulta do tipo

?- $\alpha, (\beta_1; \beta_2), \gamma$

somente se ele satisfazer α, β_1 e γ ou caso isto não seja possível, se ele satisfazer α, β_2 e γ. Analogamente, se a consulta for do tipo

?- $\alpha, not(\beta_1), \gamma$

ele tenta satisfazer α e se for o caso, tenta satisfazer β_1 e se não for o caso de β_1 ser satisfeito então tenta satisfazer γ, se γ for satisfeito então o programa Prolog satisfaz a consulta $\alpha, not(\beta_1), \gamma$ como um todo.

O *backtracking* automático é uma ferramenta muito poderosa, pelo que um programador Prolog deve conhece-la bem para assim poder explora-la melhor. Porém, as vezes, ela pode tornar o processo de execução altamente ineficiente em termos de memória e tempo de execução.

9.8.2 Comando "Cut"

O comando "*cut*" (corte) é um comando extra-lógico sem argumento que quando empregado se comporta como uma fórmula atômica sem argumento que é sempre satisfeita. Assim ele permite indicar ao Prolog quais sub-metas já satisfeitas não necessitam ser consideradas de novo ao realizar um *backtracking*. Isto é, ele aborta o processo de *backtracking*.

Existem duas razões de porque isto pode ser importante:

1. Permite que seu programa possa rodar mais rápido, pois este comando pode evitar desperdiçar tempo tentando satisfazer sub-metas que a priori são sabidas que não contribuem nunca para determinar a resposta da meta.

2. Permite que seu programa possa ocupar menos memória, pois não será necessário armazenar todas as sub-metas analisadas (pontos do *backtracking*).

Em alguns casos, o uso do comando "*cut*" pode evitar que um programa fique em laço infinito para uma determinada consulta e retorne uma resposta.

A resposta a uma regra com comando cut depende da ordem em que estão dispostas as regras e fatos no programa.

Sintaticamente o *cut* é representado pelo símbolo de exclamação "!".

Suponha que uma regra da forma

C :- $P_1, \ldots, P_m$,!,$P_{m+1}, \ldots, P_n$

foi ativada por algum objetivo O que unifica com C, então no momento em que o *cut* é encontrado o sistema já possui alguma solução para as sub-metas $P_1, \ldots, P_m$. Ao se executar o cut essa solução é tomada enquanto as demais soluções são descartadas. Além disso o sistema fica impossibilitado de unificar o objetivo O com outra cláusula. Entretanto o *backtracking* ainda é possível para as sub-metas $P_{m+1}, \ldots, P_n$.

Por exemplo, o comando cut no programa

```
        /* Base de Dados Família 2 */

/* pai(X,Y) denota "X é pai de Y" */
/* primogênito(X,Y) denota "X é o primogênito de Y" */
/* feminino(X) denota "X é do sexo feminino" */
/* masculino(X) denota "X é do sexo masculino" */

pai(joão,ana).
pai(joão,carlos).
pai(joão,sofia).
pai(joão,pedro).
feminino(X) :- mãe(X,–).
feminino(sofia).
feminino(ana).
masculino(X) :- pai(X,–).
masculino(carlos).
masculino(pedro).
primogênito(X,Y) :- pai(Y,X), masculino(X),!.
```

A inclusão do comando *cut* na última regra garante que seja retornado só uma única resposta (a primeira, isto é "Carlos´´) para a consulta

```
?- primogênito(X,joão)
```

Sem o *cut* o sistema retornaria duas respostas ("Carlos´´ e "Pedro´´) o que não faz sentido, pois o conceito de primogênito pressupõe ser único.

9.8.3 Comando "Fail"

O comando "*fail*" (falha) é um comando extra-lógico sem argumento que quando empregado se comporta como uma fórmula atômica sem argumento que nunca é satisfeita, ou seja equivalente a cláusula vazia. Ele força ao sistema um retrocesso no processo de *backtracking* para a procura de respostas alternativas. Assim, este comando pode ser usado para solicitar respostas corretas alternativas ou provocar um laço que possibilite a execução de uma seqüência repetidamente.

Por exemplo como dizer em Prolog que "Ana gosta de todas as flores menos de cravo"? uma possibilidade seria escrever para cada flor diferente de cravo um fato indicando que Ana gosta dessa flor. Porém isto seria extremamente tedioso e ineficiente, tendo em vista a quantidade de flores que existem. A outra possibilidade seria usar a comando extra-lógico "fail", da seguinte forma

```
é_cravo(cravo).

gosta(ana,X) :- é_cravo(X),!,fail.

gosta(ana,X) :- flor(X).
```

Na primeira regra se X é "cravo" então o comando cut evita o *backtracking* excluindo a segunda regra e o fail ocasionará a falha da cláusula. Desta maneira as duas regras tem o comportamento:

Se X é um cravo
então não é verdade que Ana goste de X
senão se X é uma flor então Ana gosta de X.

9.9 Algumas Ferramentas Práticas de Programação

Nesta seção introduziremos algumas ferramentas de programação que são úteis na prática. Assim, veremos como lidar com tipos de dados numéricos e com algumas estruturas de lista. Como já foi observado, as definições Prolog apresentam operadores e funções como predicados e seus predicados como termos.

9.9.1 Operadores

Para Prolog, um operador é, antes que tudo, o nome de uma função que, no caso de ser binária, pode ser escrita entre seus argumentos e que no caso de ser unária pode ser escrita antes ou depois de seu argumento. Já dissemos que um termo funcional é composto de um nome de função seguido por uma lista de argumentos entre parênteses. Além disso, um átomo simbólico é, sempre que isso fizer sentido, considerado como um termo funcional com zero argumentos.

Cada termo pode, portanto, ser representado como uma árvore cuja raíz é o nome da função e cujos ramos representam os argumentos. Por exemplo, a representação de $f(x, g(y, z))$ seria a ilustrada na árvore da figura 9.5

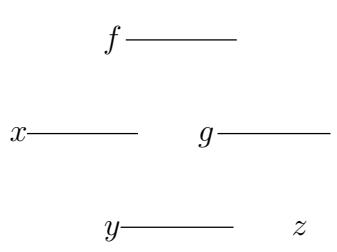


Figura 9.5: Árvore que representa o termo $f(x, g(y, z))$

Esta representação é útil pelo fato de que existem termos que podem ser escritos em mais de uma maneira. A árvore é uma forma canônica que não depende da maneira como os termos são escritos. Por exemplo, no caso de termos com um ou dois argumentos, a função pode ser escrita como uma *operação* com o nome da função escrito como um operador binário, ou como um prefixo, ou como um sufixo unário:

$$a\ op_1\ b \qquad op_2\ c \qquad d\ op_3$$

em vez de

$$op_1(a, b) \qquad op_2(c) \qquad op_3(d)$$

É suficiente que os nomes de funções se tornem *operadores* com uma prioridade. Quando um nome de função é um operador, ambas as notações podem coexistir. Assim, por exemplo, $a\ op_1\ b$ e $op_1(a, b)$ constituem o mesmo termo. A figura 9.6 ilustra a representação de árvore deste termo.

Nomes de função ou de predicado são predefinidos como operadores. Por exemplo, $a + b * c$ é o mesmo termo que $+(a, *(b, c))$, representado pela árvore da figura 9.7.

Como usual, o operador + tem menor prioridade que o operador *. Mas precisamos estar consciente que $a + b * c$ é um termo funcional em Prolog e não representa uma computação numérica a ser executada para produzir um resultado.

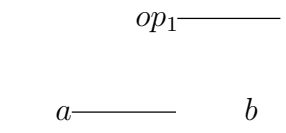


Figura 9.6: Árvore que representa o termo $op_1(a, b)$.

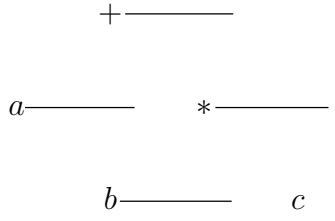


Figura 9.7: Árvore que representa o termo $+(a, *(b, c))$.

9.9.2 Computações Numéricas

Em Prolog, como usual em linguagem de programação, temos as operações numéricas usuais sobre os tipos de dados inteiros e ponto flutuante. Os operadores aritméticos e unários tem as mesmas prioridades usuais a qualquer linguagem de programação. Assim, por exemplo,

```
letícia*natalia, -ricardo, letícia*natalia+joana
```

são termos Prolog equivalentes, respectivamente, a

```
*(letícia,natalia), -(ricardo), +(*(letícia,natalia),joana).
```

Embora estes termos não tenham nada de computação numérica, elas podem ser usadas para gerar um programa válido em Prolog. Por exemplo, a base de dados

```
+(letícia, joana).
letícia*natalia+joana.
9+16.
x+10*3.
log(4).
sin(2).
```

As seguintes perguntas e respostas à base de dados obedecem às regras Prolog e não às da aritmética:

```
?- X+joana.
      ⟶ X = letícia;
      ⟶ X = letícia*natalia.
?- 12+5.
      ⟶ não
?- X+Y.
      ⟶ X=letícia , Y=joana;
      ⟶ X=letícia*natalia , Y=joana;
      ⟶ X=9 , Y=16;
      ⟶ X=x , Y=10*3;
      ⟶ não
?- log(X).
      ⟶ X=3
```

pois Prolog realiza computações simbólicas. No entanto, para auxiliar as computações simbólicas assim como para Prolog poder realizar computações usuais em outras paradigmas de programação, precisamos de facilidades de computações numéricas. Embora, teoricamente, seja possível, não podemos usar o mecanismo de unificação assim como devemos incorporar operações predefinidas. Em Prolog, computações numéricas são baseadas em três aspectos

1. Existem átomos numéricos (números) os quais se diferenciam dos simbólicos pela sua sintaxe.

2. Existe um conjunto de operadores e funções definidos como executáveis, no sentido numérico, de tal modo que termos formados somente por estas funções, operadores, números e variáveis cujos valores são termos executáveis, são também executáveis.

3. O operador binário predefinido, is, transforma um termo executável num átomo numérico que representa o resultado da computação e unifica este resultado com outros termos, normalmente uma variável.

O termo

X is T

onde X é uma variável e T um termo executável, é uma atribuição de T a X. Assim, a computação numérica definida pelo termo T, interpretado como uma expressão aritmética, é realizada e o resultado, um átomo numérico, é unificado com X, se possível.

A maioria das funções e operações numéricas disponíveis nas linguagens de programação tradicionais estão predefinidas em Prolog. No entanto, não podemos em Prolog definir nossa própria biblioteca de operações numéricas, pois não podemos criar novos predicados executáveis.

Exemplo 9.9.1 *O seguinte programa define o fatorial.*

```
fatorial(1,1).
fatorial(X,Y) :- integer(X), X≥2, X1 is X-1, fatorial(X1,Y1), Y is X*Y1.
```

Para unificar fatorial(X,Y) *é necessário que* X *seja unificado com inteiro positivo e que* Y *seja unificado com uma variável ou com o número o qual é o fatorial de* X*. Isto dá o resultado esperado, porém* fatorial *jamais será "executável" e portanto esta função não poderá aparecer numa expressão aritmética. Assim, para obter o resultado de 5!*3, devemos escrever*

```
fatorial(5,Y), Z is 3*Y
```

o resultado esperado estará em Z*.*

9.9.3 Computações com Listas

Em Prolog listas são definidas a partir de um símbolo atômico escrito por [ ], representando a lista vazia, e de pares ordenados precedidos por um ponto. Formalmente,

Definição 9.9.2 *Uma lista é uma seqüência de termos, possivelmente vazia, separados por vírgulas entre parênteses quadrado,* [*e*]*.*

Exemplo 9.9.3 *Os termos* [ ]*,* [a,b,c] *e* [1,suc(X),[a,b,c]] *são exemplos de listas, onde a primeira representa a lista vazia.*

As vezes é mais prático distinguir, em listas não vazias, o primeiro elemento da lista, chamado **cabeça**, dos restantes, chamados de **cauda**. A cabeça é um termo da lista enquanto que a cauda é uma lista. A seguinte notação é usada em Prolog para este fim: [C|T] para denotar a lista cuja cabeça é o termo C e cuja cauda é a lista T.

Exemplo 9.9.4 *As duas listas, não vazias, do exemplo 9.9.3 podem ser reescritas em termos de cabeça e cauda, da seguinte maneira:* [a|[b|c]] *e* [1|[suc(X)|[[a|[b|c]]]]]*.*

Este tipo de notação é mais utilizada quando queremos definir certas operações ou regras sobre listas. As operações são definidas por predicados cujos argumentos representam parâmetros tanto como resultados das operações.

Exemplo 9.9.5 *Uma operação usual sobre listas é dado um elemento e uma lista, determinar se o elemento está ou não na lista. O seguinte par de regras descrevem esta operação através do predicado* elem*.*

```
elem(X,[X|–]).
elem(X,[–|Y]) :- elem(X|Y).
```

Este predicado será bem sucedido se o segundo argumento é uma lista da qual o primeiro argumento é um elemento.

Uma possível sessão pergunta e resposta é

```
?- elem(X,[a,b,c]).
      ⟶ X=a;
      ⟶ X=b;
      ⟶ X=c;
?- elem(b,[a,b,c]).
      ⟶ sim
?- elem(b,[a,[b],c]).
      ⟶ não
```

Exemplo 9.9.6 *Uma outra operação usual entre listas é a concatenação de listas. O seguinte par de regras definem esta operação.*

```
concat([ ],X,X).
concat([X|Y],Z,[X|W]) :- concat(Y,Z,W).
```

Uma possível sessão de perguntas e respostas é

```
?- concat([a,b],[c,d],L).
      ⟶ L=[a,b,c,d];
      ⟶ não
?- concat(X,Y,[a,b,c,d]).
      ⟶ X=[ ] , Y=[a,b,c,d];
      ⟶ X=[a] , Y=[b,c,d];
      ⟶ X=[a,b] , Y=[c,d];
      ⟶ X=[a,b,c] , Y=[d];
      ⟶ X=[a,b,c,d] , Y=[ ];
      ⟶ não
```

9.10 Exercícios

1. Crie uma base de dados de livros em prolog, onde cada livro deve conter informações do autor principal, título e ano de publicação. Faça as seguintes perguntas:

   (a) Quais os livros de um determinado autor?

   (b) Existe um livro na sua base publicado este ano?

   (c) Quais os livros escritos nos anos 1997 e 1998?

   (d) Quais os livros escritos nos anos anteriores?

2. Defina regras e fatos no seu programa em lógica Família que lhe permita inferir as seguintes relações

   (a) cônjuges(X,Y) - X é um cônjuge de y se tem algum filho em comum.

   (b) avós(X,Y) - X é um dos avós de Y.

   (c) irmãos(X,Y) - X e Y são irmãos (tem ambos pais em comum).

   (d) irmã(X,Y) - X é irmã de Y.

   (e) tia(X,Y) - X é um tia de Y.

   (f) primo(X,Y) - X é um primo ou prima em primeiro grau de Y.

   (g) parentes(X,Y) - X e Y são parentes (são da mesma família).

3. Escreva um programa PROLOG que descreva as seguintes situações:

   (a) Marcos era um homem;

   (b) Marcos nasceu em Pompéia;

   (c) Todos os que nasceram em Pompéia são romanos;

   (d) César era um soberano;

   (e) Todos os romanos eram leais a César ou então odiavam-no;

   (f) Todo mundo é leal a alguém;

   (g) As pessoas só tentam assassinar soberanos aos quais não são leais;

   (h) Marcos tentou assassinar César;

   (i) Todos os homens são pessoas.

   Em seguida faça a seguinte pergunta ao seu programa: Marcos era leal ao César? qual seria a resposta de seu programa? Porque?

4. Escreva programas em lógica que computem as seguintes funções numéricas. Assuma o tipo de dados inteiro munido das operações aritméticas e as relações de igualdade e de ordem ($\leq$ e $<$) como primitivo

(a) valor absoluto.

(b) raiz quadrada.

(c) quadrado de um número.

(d) $\Sigma n = 1 + 2 + \cdots + n$ para cada $n \geq 0$ e $\Sigma n = 0$ caso $n < 0$..

(e) $f(x)$ dê o x-ésimo valor da série de Fibonacci (1,1,2,3,5,8,13,$\cdots$).

(f) determinar se um número é primo ou não.

5. Escreva programas em lógica para os seguintes procedimentos que manipulam com estruturas de listas (de números naturais).

(a) insira1(X,L1,L2) - L2 é o resultado de inserir X no inicio da lista L1.

(b) insira2(X,L1,L2) - L2 é o resultado de inserir X (em qualquer parte) na lista L1.

(c) mesma(L1,L2) - Diz se L1 e L2 tem os mesmos elementos, independente da ordem em que estão dispostos.

(d) retira(X,L1,L2) - L2 é o resultado de retirar a primeira ocorrência (de esquerda para direita) do elemento X da lista L1. Se X não faz parte da lista L1, então L2 é o próprio L1.

(e) retira2(X,L1,L2) - L2 é o resultado de retirar todas as ocorrências do elemento X da lista L1. Se X não faz parte da lista L1, então L2 é o próprio L1.

(f) ordenada(L) - Diz se L está ordenada (ascendentemente).

(g) ordena(L1,L2) - L2 é o resultado de ordenar L1 ascendentemente.

(h) apaga(X,L1,L2) - L2 é o resultado de apagar todas as ocorrências de X em L1.

(i) sublista(L1,L2) - L1 é uma sublista de L2, isto é, a lista L1 inteira está (contiguamente) em L2. Assim, isto não significa que cada elemento de L1 está em L2. Por exemplo

- sublista([1,2],[2,1,2,3]) é verdadeira
- sublista([1,2],[2,1,3]) é falsa

(j) resto(X,L1,L2) - L2 é a sublista de L1 que começa na primeira ocorrência de X em L1. Se X não ocorre em L1 então L2=[ ].

(k) max(L,X) - X é o máximo valor da lista de números L. (Assuma que você dispõe do relacionamento "maior ou igual que", denotado por $\geq$).

(l) inverte(L1,L2) - L2 é a lista L1 invertida, isto é, se L1=[1,2,3] então L2=[3,2,1].

Capítulo 10

Outras Apresentações da Lógica Clássica

Tanto em lógica proposicional quanto em lógica de predicados vimos três abordagens ou estilos diferentes de se lidar com elas. Essas abordagens captam diferentes aspectos da lógica:

- semântico: no qual o importante é o conceito de verdade e conseqüência lógica,
- formal: no qual desenvolvemos uma teoria de provas (no caso do estilo axiomático isto compreende uma linguagem formal, um conjunto de axiomas e regras de inferências) para construir provas para os teoremas da lógica,
- operacional: no qual apresentamos um algoritmo para refutar a negação de um teorema (no caso usamos o método de eliminação de literais complementares).

O primeiro é de maior interesse filosófico e matemático no sentido clássico (teoria dos modelos), o segundo de maior interesse matemático construtivo e o terceiro computacional. No entanto, estas três abordagens da lógica clássica são complementares, do ponto de vista epistemológico, e são essenciais para o bom entendimento da lógica para ciências da computação. Observe que os dois últimos estilos são ambos de natureza sintática, sendo que o terceiro é mais operacional, pois é factível de ser implementado computacionalmente, por exemplo definindo uma linguagem de programação baseada nos princípios lógicos desse estilo.

Neste capítulo veremos mais três maneiras de apresentar a lógica proposicional e de predicados clássica. Sendo que as duas primeiras, dedução natural e cálculo de seqüentes de Gentzen, são de natureza formal, isto é, descrevem um conjunto de regras formais para construir provas dos teoremas e deduções da lógica proposicional e de predicados. O terceiro estilo que veremos aqui, o estilo tableaux, é uma maneira alternativa de se realizar provas por refutação e, portanto, é mais um método computacional de se provar teoremas.

10.1 Dedução Natural

Provas axiomáticas são difíceis de construir além de serem muito longas. Assim, na prática não se constrói uma tal prova, mas mostra-se que existe uma prova. Uma maneira na qual temos usados esta técnica é quando, numa prova, consideramos qualquer teorema que já tenha sido provado anteriormente. Outra maneira, é quando usamos o teorema da dedução, o qual nos permite provar a partir de assunções (hipóteses) e não uma prova axiomática como originalmente definida. Uma terceira maneira é introduzir novos símbolos por definição, e derivar novas regras para os novos símbolos as quais serão, usualmente, usadas em provas a partir de assunções. Assim, a partir de um certo momento, o desenvolvimento de um sistema axiomático dificilmente será um prova axiomática real, mas baseado sobre uma variedade de pequenas partes. Usando uma analogia computacional, uma prova é baseada num conjunto de "macros" de provas.

A principal idéia do que chamamos "*dedução natural*" é abandonar o ponto de partida axiomático e, em vez disso, começar com o que chamamos acima de "pequenas partes". O ponto fundamental em dedução natural é considerar a noção de prova a partir de assunções como noção básica, e trabalhar simplesmente com estas. Tais provas não são pensadas como abreviando alguma outra, nem como um tipo mais básico de prova, mas são os objetos primários de estudo. Logo, as regras básicas iniciais serão regras para usar em provas a partir de assunções. Axiomas, como tradicionalmente entendidos, não terão nenhum papel preponderante (inclusive nos sistemas de dedução natural que veremos aqui não teremos a figura do "axioma"). Esta é a característica crucial de todo sistema de dedução natural. Mas, existem diversas outras características que também são hoje em dia esperadas e desejadas. Por exemplo, o conectivo lógico $\rightarrow$ não tem qualquer *status* especial, com respeito aos demais conectivos, como era o caso no sistema axiomático.

Desse modo no estilo de dedução natural a lógica é apresentada apenas por regras. Este estilo é caracterizado por dois fatos básicos: primeiro, o uso, em certas regras, de assunções (hipóteses). Essas hipóteses são internas à derivação e não devem ser confundidas com **premissas**. Uma hipótese é somente para ser usada com o propósito de derivar um resultado particular determinado pela regra que motivou sua introdução. Cada hipótese tem um *escopo* que vai da linha na qual ela é introduzida até a linha antes da regra na qual é descartada. A hipótese não deve ser usada fora do escopo. Por isso a definição linear de prova (como uma seqüência finita) deve ser ligeiramente modificada de modo a permitir o acomodamento de hipóteses e de premissas. Uma segunda característica do estilo de dedução natural consiste no fato de que as regras aparecem em pares. Uma regra para introduzir o operador e outra para eliminá-lo.

Assim, sistemas de dedução natural constituem uma outra família de mecanismos de prova, que tenta formalizar o tipo de raciocínio nos argumentos informais das pessoas. Eles são baseados nas idéias de *provas subordinadas*, nas quais derivamos conclusões a partir das premissas, então *descartamos* estas premissas para produzir resultados livres de assunções.

Uma típica regra de um sistema dedução natural é: Se podemos derivar β, assumindo α, então podemos descartar a assunção α e concluir que temos provado $\alpha \to \beta$. Isto é esquematizado na figura 10.1.

$$\dfrac{\begin{array}{c}\overline{\alpha} \\ \vdots \\ \beta\end{array}}{\alpha \to \beta}$$

Figura 10.1: Uma regra de dedução natural para a implicação

Observe que, analogamente a provas em teorias formais, não necessariamente usamos a premissa α para obter β.

Um conjunto de regras devem satisfazer algumas propriedades para ser considerado um "sistema de dedução natural".

1. Para cada conectivo lógico, um sistema de dedução natural deve considerar somente regras que concernem especificamente a esse conectivo e mais nenhum outro conectivo. Isto é mais um requerimento de elegância que uma condição que possa ser considerada como "dedução natural".

2. Para cada conectivo existem uma ou duas regras de *introdução* as quais permitem introduzir o conectivo, e uma ou duas regras de *eliminação* que possibilitem eliminar o conectivo.

3. Deve-se ter uma certa completude entre as regras de introdução e eliminação de um conectivo lógico, no sentido de levarem a um sistema completo para as fórmulas contendo só este conectivo. Isto é, para cada fbf logicamente válida que contenha somente esse conectivo se deve ter uma dedução natural para ela que use somente as regras de introdução e eliminação do conectivo.

4. As regras para cada conectivo lógico devem ser "naturais", no sentido que inferências usando essas regras ilustrem argumentos e inferências de maneira "natural".

5. Se a fórmula a ser provada não é "muito complexa", então deve haver uma prova razoavelmente curta que use somente as regras inicialmente adotadas.

Existem diversas maneiras de ilustrar uma prova em dedução natural. A que usaremos aqui não vê uma prova como uma seqüência linear de fórmulas, mas como um vetor bidimensional destas, arranjados numa estrutura de árvore. A estrutura tem uma fórmula como sua raiz, a qual fica na parte inferior da árvore. A fórmula na raiz é a fórmula que é provada e os ramos que se espalham para acima da raíz representam a série de raciocínio que necessitamos para chegar até a conclusão, e cada ramo tem como sua fórmula mais

acima uma assunção da prova. Em uma prova de dedução natural existem dois tipos de assunções, uma na qual ela é considerada uma premissa da prova, semelhante a α, num prova de $\alpha \vdash \beta$ e outra na qual a assunção é usada temporariamente, em cujo caso dizemos que ela é descartada na prova ao se usar uma determinada regra.

Devemos ter cuidado em não usar, numa derivação, uma assunção que já tenha sido descartada na prova. Por exemplo na regra da figura 10.1, a assunção α deve ser descartada após usar essa regra. Indicaremos isto colocando um traço acima da assunção e pondo no lado um número, sinalizando a derivação na prova pela qual ela foi descartada (o mesmo número estará no lado direito da derivação).

10.1.1 Lógica Proposicional

Um sistema de dedução natural para a lógica proposicional é dado pelo seguinte conjunto de regras.

Introdução do $\wedge$:

$$\mathsf{I}\wedge : \frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

Eliminação do $\wedge$:

$$\mathsf{E}\wedge_1 : \frac{\alpha \wedge \beta}{\alpha} \qquad \mathsf{E}\wedge_2 : \frac{\alpha \wedge \beta}{\beta}$$

Exemplo 10.1.1 *Uma dedução natural para mostrar que a conjunção é associativa, isto é que $\alpha \wedge (\beta \wedge \gamma) \vdash (\alpha \wedge \beta) \wedge \gamma$, é ilustrada pela seguinte árvore de derivação:*

$$\dfrac{\dfrac{\mathsf{E}\wedge_1 : \dfrac{\alpha \wedge (\beta \wedge \gamma)}{\alpha} \qquad \mathsf{E}\wedge_1 : \dfrac{\mathsf{E}\wedge_2 : \dfrac{\alpha \wedge (\beta \wedge \gamma)}{\beta \wedge \gamma}}{\beta}}{\mathsf{I}\wedge : \dfrac{\alpha \qquad \beta}{\alpha \wedge \beta}} \qquad \mathsf{E}\wedge_2 : \dfrac{\mathsf{E}\wedge_2 : \dfrac{\alpha \wedge (\beta \wedge \gamma)}{\beta \wedge \gamma}}{\gamma}}{\mathsf{I}\wedge : \dfrac{\alpha \wedge \beta \qquad \gamma}{(\alpha \wedge \beta) \wedge \gamma}}$$

Introdução do $\vee$:

$$\mathsf{I}\vee_1 : \frac{\alpha}{\alpha \vee \beta} \qquad \mathsf{I}\vee_2 : \frac{\beta}{\alpha \vee \beta}$$

Eliminação $\vee$:

$$\mathsf{E}\vee : \frac{\alpha \vee \beta \quad \begin{array}{c}\overline{\alpha}(n)\\ \vdots \\ \gamma\end{array} \quad \begin{array}{c}\overline{\beta}(n)\\ \vdots \\ \gamma\end{array}}{\gamma}\ (\mathsf{n})$$

Embora a regra de eliminação do $\vee$, aparentemente, não seja tão imediata como as anteriores, ela é bastante natural. Para ver isso pense da seguinte maneira: Suponha que temos uma premissa $\alpha \vee \beta$, e queremos saber o que podemos deduzir dela, então vemos o que podemos deduzir de α e o que podemos deduzir de β, se γ é dedutível de ambas então, independente de qual seja verdadeira (α ou β) poderemos sempre deduzir γ de $\alpha \vee \beta$. Este processo implica que tivemos inicialmente que introduzir a assunção α e a assunção β, pelo que devemos descartar essas assunções quando derivamos γ.

Exemplo 10.1.2 *Uma árvore de derivação para a propriedade de comutatividade do conectivo $\vee$, isto é que $\beta \vee \alpha \vdash \alpha \vee \beta$, é mostrada a seguir:*

$$\mathsf{E}\vee : \frac{\beta \vee \alpha \qquad \mathsf{I}\vee_2 : \dfrac{\overline{\beta}(1)}{\alpha \vee \beta} \qquad \mathsf{I}\vee_1 : \dfrac{\overline{\alpha}(1)}{\alpha \vee \beta}}{\alpha \vee \beta}\ (1)$$

Note que nesta dedução natural de $\beta \vee \alpha \vdash \alpha \vee \beta$, β e α são premissas, enquanto que $\beta \vee \alpha$ é uma hipótese.

Também podem haver deduções que combinem regras de diferentes conectivos.

Exemplo 10.1.3 *Uma árvore de derivação para a propriedade de distributividade do conectivo $\vee$ sobre $\wedge$, isto é que $\alpha \vee (\beta \wedge \gamma) \vdash (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$, é mostrada a seguir:*

$$\mathsf{E}\vee:\ \dfrac{\alpha\vee(\beta\wedge\gamma)\qquad \dfrac{\mathsf{I}\vee_1:\ \dfrac{\overline{\alpha}(1)}{(\alpha\vee\beta)}\quad \mathsf{I}\vee_1:\ \dfrac{\overline{\alpha}(1)}{(\alpha\vee\gamma)}}{\mathsf{I}\wedge:\ \dfrac{}{(\alpha\vee\beta)\wedge(\alpha\vee\gamma)}}\qquad \dfrac{\mathsf{E}\wedge_1:\ \dfrac{\overline{\beta\wedge\gamma}(1)}{(\alpha\vee\beta)}\quad \mathsf{E}\wedge_2:\ \dfrac{\overline{\beta\wedge\gamma}(1)}{(\alpha\vee\gamma)}}{\mathsf{I}\wedge:\ \dfrac{}{(\alpha\vee\beta)\wedge(\alpha\vee\gamma)}}}{(\alpha\vee\beta)\wedge(\alpha\vee\gamma)}(1)$$

Introdução do $\rightarrow$:

$$\mathsf{I}\rightarrow:\ \dfrac{\begin{array}{c}\overline{\alpha}(n)\\ \vdots\\ \beta\end{array}}{\alpha\rightarrow\beta}\ (\mathsf{n})$$

Eliminação do $\rightarrow$:

$$\mathsf{E}\rightarrow:\ \dfrac{\alpha\quad \alpha\rightarrow\beta}{\beta}$$

Exemplo 10.1.4 *Uma dedução natural para o axioma $\alpha\rightarrow(\beta\rightarrow\alpha)$ é ilustrada pela seguinte árvore:*

$$\mathsf{I}\rightarrow:\ \dfrac{\mathsf{I}\rightarrow:\ \dfrac{\overline{\beta}(1)\qquad \overline{\alpha}(2)}{\beta\rightarrow\alpha}\ (1)}{\alpha\rightarrow(\beta\rightarrow\alpha)}\ (2)$$

Exemplo 10.1.5 *Uma dedução natural de $\beta,\alpha\rightarrow(\beta\rightarrow\gamma)\vdash\alpha\rightarrow\gamma$ é ilustrada pela seguinte árvore:*

$$\mathsf{I}\rightarrow:\ \dfrac{\mathsf{E}\rightarrow:\ \dfrac{\mathsf{E}\rightarrow:\ \dfrac{\overline{\alpha}(1)\qquad \alpha\rightarrow(\beta\rightarrow\gamma)}{\beta\rightarrow\gamma}\qquad \beta}{\gamma}}{\alpha\rightarrow\gamma}\ (1)$$

Na literatura não há um consenso claro de quais as regras mais "naturais" para a negação. Aqui, consideramos uma que capte a idéia de que se na assunção de α derivamos uma contradição, então podemos derivar $\neg\alpha$ e descartar a assunção α.

Introdução do $\neg$:

$$\mathsf{I}\neg : \frac{\begin{array}{cc}\overline{\alpha}(\mathsf{n}) & \overline{\alpha}(\mathsf{n}) \\ \vdots & \vdots \\ \beta & \neg\beta\end{array}}{\neg\alpha}\ (\mathsf{n})$$

Eliminação do $\neg$:

$$\mathsf{E}\neg : \frac{\neg\neg\beta}{\beta}$$

Exemplo 10.1.6 *O único axioma, dos sistema visto no capítulo 4, que considera a negação é $(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$. Uma dedução natural para esse axioma é dada pela seguinte árvore.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\neg\beta}(1) \quad \overline{\neg\beta \to \alpha}(2)}{\alpha}\ \mathsf{E}\to: \quad \dfrac{\overline{\neg\beta}(1) \quad \overline{\neg\beta \to \neg\alpha}(3)}{\neg\alpha}\ \mathsf{E}\to:}{\neg\neg\beta}\ \mathsf{I}\neg : (1)}{\beta}\ \mathsf{E}\neg :}{(\neg\beta \to \alpha) \to \beta}\ \mathsf{I}\to: (2)}{(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)}\ \mathsf{I}\to: (3)$$

Exemplo 10.1.7 *A seguir uma dedução de $\neg(\alpha \vee \beta) \vdash \neg\alpha \wedge \neg\beta$.*

$$\dfrac{\dfrac{\dfrac{\neg(\alpha \vee \beta) \quad \dfrac{\overline{\alpha}(1)}{\alpha \vee \beta}\ \mathsf{I}\vee :}{\neg\alpha}\ \mathsf{E}\to: (1) \quad \dfrac{\neg(\alpha \vee \beta) \quad \dfrac{\overline{\beta}(2)}{\alpha \vee \beta}\ \mathsf{I}\vee :}{\neg\beta}\ \mathsf{E}\to: (2)}{\neg\alpha \wedge \neg\beta}\ \mathsf{I}\wedge : (1)$$

Teorema 10.1.8 *Se α é um teorema de T_P, então existe uma dedução natural para α.*

DEMONSTRAÇÃO: Se $\vdash \alpha$ então existe uma prova $\alpha_1, \alpha_2, \cdots, \alpha_n$ de α. Provaremos por indução no tamanho da prova que existe uma dedução natural para α.

Caso base: Se $i = 1$, então α é um dos axiomas.

1. Uma dedução natural para $\alpha \to (\beta \to \alpha)$ é mostrado no exemplo 10.1.4.

2. Uma dedução natural para $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$ é dada pela seguinte árvore:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\alpha}(1) \quad \overline{\alpha \to \beta}(2)}{\beta}\mathsf{E}\to: \quad \dfrac{\overline{\alpha}(1) \quad \overline{\alpha \to (\beta \to \gamma)}(3)}{\beta \to \gamma}\mathsf{E}\to:}{\gamma}\mathsf{E}\to:}{\alpha \to \gamma}\mathsf{I}\to:(1)}{(\alpha \to \beta) \to (\alpha \to \gamma)}\mathsf{I}\to:(2)}{(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))}\mathsf{I}\to:(3)$$

3. Uma dedução natural para o axioma $(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$ é mostrado no exemplo 10.1.6.

Etapa indutiva: Suponha que existe uma dedução natural para cada α_i na prova tal que $i < k$. Então por definição de prova

- α_k é um axioma, ou
- α_k segue por MP de α_i e α_j, onde $i, j < k$ e α_j é $\alpha_i \to \alpha_k$.

O primeiro caso é exatamente como no caso base mostrado acima. No segundo caso, pela etapa indutiva, existe uma dedução natural para α_i e outra para $\alpha_i \to \alpha_k$. Logo, juntando ambas deduções e aplicando a regra $(\mathsf{E}\to)$ obteremos uma dedução natural para α_k.

Portanto segue o teorema. ■

Em dedução natural também podemos provar conseqüências lógicas.

Exemplo 10.1.9 *Uma dedução natural para* $\neg\beta \to \neg\alpha \vdash \alpha \to \beta$ *é dada pela seguinte árvore.*

$$\mathsf{E}\to:\ \dfrac{\mathsf{I}\neg:\dfrac{\dfrac{\dfrac{\dfrac{\overline{\neg\beta}\,(1)\quad \neg\beta\to\neg\alpha}{\neg\alpha}\qquad \overline{\alpha}\,(2)}{\neg\neg\beta}\,\mathsf{E}\neg}{\beta}}{\alpha\to\beta}\,\mathsf{I}\to\,(2)}{}\,(1)$$

10.1.2 Lógica de Predicados

Analogamente ao estilo axiomático, o sistema de dedução natural para a lógica proposicional está incluído no correspondente da lógica de predicados. Assim, só devemos incorporar as respectivas regras de introdução e eliminação dos quantificadores.

Introdução do $\forall$:

$$\mathsf{I}\forall:\ \frac{\alpha}{\forall x.\alpha}$$

Eliminação do $\forall$:

$$\mathsf{E}\forall:\ \frac{\forall x.\alpha}{\alpha[t/x]}\ \text{, onde } t \text{ é um termo livre para } x \text{ em } \alpha$$

Exemplo 10.1.10 *Deduções naturais para os axiomas 5 e 4, respectivamente, são dadas pelas árvores seguintes:*

$$\mathsf{I}\to:\ \dfrac{\mathsf{E}\forall:\ \dfrac{\overline{\forall x.\alpha}\,(1)}{\alpha[t/x]}}{\forall x.\alpha\to\alpha[t/x]}\,(1)$$

$$\mathsf{I}\to:\ \dfrac{\mathsf{E}\to:\ \dfrac{\overline{\alpha}(1)\qquad \mathsf{E}\forall:\ \dfrac{\overline{\forall x.(\alpha\to\beta)}(1)}{\alpha\to\beta}}{\beta}}{\forall x.\beta}\ (1)}{\alpha\to\forall x.\beta}\ (2)$$

$$\forall x.(\alpha\to\beta)\to(\alpha\to\forall x.\beta)$$

Exemplo 10.1.11 *Uma árvore de dedução natural para* $\forall x.\forall y.P(x,y)\to\forall x.P(x,x)$ *é a seguinte:*

$$\mathsf{I}\forall:\ \dfrac{\mathsf{E}\forall:\ \dfrac{\mathsf{E}\forall:\ \dfrac{\mathsf{I}\forall:\ \dfrac{\overline{\forall x.\forall y.P(x,y)}(1)}{\forall y.P(x,y)}}{P(x,x)}}{\forall x.P(x,x)}}{\forall x.\forall y.P(x,y)\to\forall x.P(x,x)}$$

Introdução do $\exists$:

$$\mathsf{I}\exists:\ \frac{\alpha[t/x]}{\exists x.\alpha}\ \text{, onde } t \text{ é um termo livre para } x \text{ em } \alpha$$

Eliminação do $\exists$:

$$\mathsf{E}\exists:\ \frac{\exists x.\alpha}{\alpha[a/x]}\ \text{, onde } a \text{ é uma constante nova na prova de } \exists x.\alpha$$

Exemplo 10.1.12 *Uma dedução de* $\exists x.(P(a)\wedge R(x))\vdash P(a)\wedge\exists x.R(x)$ *no estilo de dedução natural é ilustrada na seguinte árvore.*

$$\dfrac{\dfrac{\dfrac{\exists x.(P(a) \wedge R(x))}{P(a) \wedge R(b)}\,\mathsf{E}\exists}{P(a)}\,\mathsf{E}\wedge_1 \qquad \dfrac{\dfrac{\dfrac{\exists x.(P(a) \wedge R(x))}{P(a) \wedge R(b)}\,\mathsf{E}\exists}{R(b)}\,\mathsf{E}\wedge_2}{\exists x.R(x)}\,\mathsf{I}\exists}{P(a) \wedge \exists x.R(x)}\,\mathsf{I}\wedge$$

Exemplo 10.1.13 *Uma dedução de* $\neg\forall x.\alpha \rightarrow \exists x.\neg\alpha$ *no estilo de dedução natural é ilustrada na seguinte árvore.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\alpha}\,(1)}{\forall x.\alpha}\,\mathsf{I}\forall \qquad \overline{\neg\forall x.\alpha}\,(2)}{\neg\alpha}\,\mathsf{I}\neg\,(1)}{\exists x.\neg\alpha}\,\mathsf{I}\exists}{\neg\forall x.\alpha \rightarrow \exists x.\neg\alpha}\,\mathsf{I}\rightarrow$$

No caso do estilo axiomático, temos algumas restrições no uso do teorema da dedução. Por outro lado, o uso da regra de introdução da implicação, do jeito que está, incorporaria, implicitamente, na lógica de predicados um teorema da dedução irrestrito. Portanto, para ser coerente com a lógica de predicado vista aqui, devemos introduzir estas restrições. Para isto, devemos observar que a regra de inferência Gen, que era a que trazia os problemas apontados no capítulo 6, é exatamente igual à regra de introdução do quantificador universal ($\mathsf{I}\forall$). A incorporação da generalização, também está implícita no uso de regra $\mathsf{I}\exists$, pois para provar, na teoria formal de primeira ordem TP, que $\alpha[t/x] \vdash \exists x.\alpha$ (ou equivalentemente que $\alpha[t/x] \vdash \neg\forall x.\neg\alpha$) é necessário aplicar a regra Gen. Assim, a regra $\mathsf{I}\rightarrow$ só poderá ser aplicada, no sistema de dedução natural da lógica de predicados, quando no ramo que deriva β de α, não se aplicou nenhuma regra $\mathsf{I}\forall$ ou $\mathsf{I}\exists$ introduzindo uma variável x que é livre em α. A maioria dos autores da apresentação da lógica clássica no estilo de dedução natural [Tho87, Cos92, RS92, Bos97] incorporam restrições no uso das regras de introdução e eliminação dos quantificadores para tentar compensar a ausência de restrições no uso da regra $\mathsf{I}\rightarrow$. Mas, estas restrições nas regras dos quantificadores são severas, impedindo provar conseqüências lógicas do tipo $P(x) \vdash \forall x.P(x)$.

Como vimos no teorema 10.1.8, para todo teorema da lógica proposicional existe uma dedução natural (proposicional). Assim, é natural pensarmos num teorema equivalente para a lógica de predicados.

Teorema 10.1.14 *Se α é um teorema de T^P então existe uma prova no estilo de dedução natural para α.*

Demonstração: A demonstração deste teorema é análoga à demonstração do teorema 10.1.8. Assim, só devemos acrescentar uma prova de que os axiomas A4 e A5, da teoria formal de 1^a ordem, são dedutíveis no estilo de dedução natural, o qual foi mostrado no exemplo 10.1.10. Por outro lado como a regra de inferência Gen é exatamente a mesma que a regra I$\forall$, podemos afirmar que para todo teorema de T^P existe uma prova no estilo de dedução natural. ■

10.2 Cálculo de seqüente de Gentzen

Pensemos, de modo geral, no que acontece numa prova de dedução natural. Uma prova é uma seqüência de fórmulas, as quais estabelecem algum seqüente[1] com o lado esquerdo sendo composto de todas as hipóteses que não foram descartadas na prova e com o lado direito sendo composto pela fórmula que foi provada ao final da prova. Também, consideraremos as regras de inferências como regras sobre seqüentes. Uma prova sempre iniciará com uma assunção, por exemplo α, e se não acrescentamos nenhuma outra, então ela mesma é considerada como a prova do seqüente $\alpha \vdash \alpha$. Assim, a regra que nos permite começar é uma regra a qual nos leva diretamente a que todos os seqüentes deste tipo sejam corretos. As outras regras são todas condicionais, elas refletem a idéia de que se certos seqüentes são corretos, então os seqüentes que podemos concluir deles também são corretos. Por exemplo, a regra Modus Ponens pode ser escrita em forma de seqüentes como segue

$$\text{Se } \Gamma \vdash \alpha \text{ e } \Delta \vdash \alpha \to \beta \text{ então } \Gamma, \Delta \vdash \beta$$

O que acontece numa prova é que começamos com certos seqüentes conhecidos como corretos e então deduzimos que outros seqüentes devem, portanto, também ser corretos.

A idéia de um cálculo de seqüente é manter um registro explícito de que seqüente é estabelecido em cada ponto da prova. Isto é possível introduzindo um novo tipo de prova na qual cada linha, por si mesmo, é um seqüente provado até esse ponto da prova. Logo, analogamente às provas de dedução natural, uma prova no cálculo de seqüentes não é uma seqüência linear ou outro vetor de fórmulas, mas uma combinação de vetores de seqüentes.

Podemos pensar nas listas que compõem um seqüente como tacitamente fechadas por parênteses de chave, pelo que seu papel seria especificar um conjunto de fórmulas. Quando

[1] Um seqüente é um par de ordenado de listas finitas de fórmulas, que pode ser interpretado como que uma das fórmulas da segunda lista "segue" (pode ser deduzida) da primeira lista. Na prática usamos a notação sugerida por Gentzen: O seqüente $\langle \Gamma, \Delta \rangle$ será denotado por $\Gamma \vdash \Delta$.

um conjunto é especificado por uma lista de seus membros, então a ordem na qual os membros são listados não é relevante, e qualquer repetição na lista pode automaticamente ser desconsiderada. Isto é devido a que dois conjuntos são iguais se, e somente se, tem os mesmos membros e diferentes listas dos membros podem ainda listar os mesmos membros. Mas se não quisermos pensar em termos de conjuntos, e trabalharmos diretamente com listas, então não podemos prosseguir sem dizer que ordens e repetições são relevantes. Para isso precisamos considerar duas novas regras de inferência que reflitam estas propriedades intrínsecas a conjuntos, a regra de intercâmbio (Int) e a regra de contração (Con), respectivamente.

Analogamente à dedução natural, podemos apresentar o cálculo de seqüentes como um sistema cujas provas têm a estrutura de uma árvore. Na posição superior sobre cada ramo existirá um seqüente o qual, de acordo com as regras, pode ser descartado. Este portanto será uma instância de uma regra com assunções. As outras posições na prova serão ocupadas por um seqüente o qual é deduzido de outros seqüentes, os seqüentes do qual este é deduzido são escritos imediatamente acima deste separando-os por uma linha horizontal. Podemos, portanto, escrever nossas regras básicas de inferência da mesma maneira. Diferentemente do estilo de dedução natural, no estilo de seqüentes de Gentzen teremos dois tipos de regras: aquelas que dependem de um dos conectivo lógicos (usadas para introduzir este conectivo ou no lado esquerdo ou direito do seqüente) e que por isso são chamadas de **regras lógicas** e aquelas que não dependem dos conectivos lógicos, mas da disposição ou estrutura do seqüente, denominadas **regras estruturais**.

Temos cinco regras estruturais ,algumas das quais tem sua versão para o lado direito (que denotaremos por d) e outra para o lado esquerdo (denotado por e), as quais se denominam: inclussão (Inc), emfraquecimento (Emf), corte (Cor), intercâmbio (Int) e contração (Con).

Regras Estruturais

$$\mathsf{Inc}: \frac{}{\Gamma, \alpha \vdash \alpha, \Delta} \qquad \mathsf{Cor}: \frac{\Gamma \vdash \alpha \quad \Gamma, \alpha \vdash \Delta}{\Gamma \vdash \Delta}$$

$$\mathsf{Emf_e}: \frac{\Gamma \vdash \Delta}{\Gamma, \alpha \vdash \Delta} \qquad \mathsf{Emf_d}: \frac{\Gamma \vdash \Delta}{\Gamma \vdash \alpha, \Delta}$$

$$\mathsf{Int_e}: \frac{\Gamma, \alpha, \beta, \Delta \vdash \Theta}{\Gamma, \beta, \alpha, \Delta \vdash \Theta} \qquad \mathsf{Int_d}: \frac{\Gamma \vdash \Delta, \alpha, \beta, \Theta}{\Gamma \vdash \Delta, \beta, \alpha, \Theta}$$

$$\mathsf{Con_e}: \frac{\Gamma, \alpha, \alpha \vdash \Delta}{\Gamma, \alpha \vdash \Delta} \qquad \mathsf{Con_d}: \frac{\Gamma \vdash \alpha, \alpha\Delta}{\Gamma \vdash \alpha, \Delta}$$

A regra Cor pode ser omitida, uma vez que existe uma maneira de obtê-la a partir das outras. Como esta demonstração é complexa não a incluímos neste texto. Uma demonstração de que a eliminação do corte é possível pode ser encontrada em [GLT89, Fit96, Bos97].

10.2.1 Lógica Proposicional

No cálculo de seqüentes, também existem regras para os operadores lógicos, chamadas regras lógicas. Para cada um dos operadores lógicos teremos uma regra para introduzir o conectivo no lado esquerdo e outra para o lado direito.

Regras Lógicas

$$\wedge_{\mathsf{d}} : \frac{\Gamma \vdash \alpha, \Delta \quad \Gamma \vdash \beta, \Delta}{\Gamma \vdash \alpha \wedge \beta, \Delta} \qquad \wedge_{\mathsf{e}} : \frac{\Gamma, \alpha, \beta \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta}$$

$$\vee_{\mathsf{d}} : \frac{\Gamma \vdash \alpha, \beta, \Delta}{\Gamma \vdash \alpha \vee \beta, \Delta} \qquad \vee_{\mathsf{e}} : \frac{\Gamma, \alpha \vdash \Delta \quad \Gamma, \beta \vdash \Delta}{\Gamma, \alpha \vee \beta \vdash \Delta}$$

$$\rightarrow_{\mathsf{d}} : \frac{\Gamma, \alpha \vdash \beta, \Delta}{\Gamma \vdash \alpha \rightarrow \beta, \Delta} \qquad \rightarrow_{\mathsf{e}} : \frac{\Gamma \vdash \alpha, \Delta \quad \Gamma, \beta \vdash \Delta}{\Gamma, \alpha \rightarrow \beta \vdash \Delta}$$

$$\neg_{\mathsf{d}} : \frac{\Gamma, \alpha \vdash \Delta}{\Gamma \vdash \neg\alpha, \Delta} \qquad \neg_{\mathsf{e}} : \frac{\Gamma \vdash \alpha, \Delta}{\Gamma, \neg\alpha \vdash \Delta}$$

Exemplo 10.2.1 *Uma dedução no cálculo de seqüentes para* $\vdash \neg(\alpha \wedge \beta) \rightarrow (\neg\alpha \vee \neg\beta)$ *é dada pela seguinte árvore de dedução.*

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,\mathsf{Inc}}{\alpha, \beta \vdash \alpha}\,\mathsf{Emf_e} \quad \dfrac{}{\beta \vdash \beta}\,\mathsf{Inc}}{\beta, \alpha \vdash \alpha}\,\mathsf{Int_e} \quad \dfrac{}{\beta, \alpha \vdash \beta}\,\mathsf{Emf_e}}{\beta, \alpha \vdash \alpha \wedge \beta}\,\wedge_{\mathsf{d}}}{\beta \vdash \neg\alpha, \alpha \wedge \beta}\,\neg_{\mathsf{d}}}{\beta \vdash \alpha \wedge \beta, \neg\alpha}\,\mathsf{Int_d}}{\vdash \neg\beta, \alpha \wedge \beta, \neg\alpha}\,\neg_{\mathsf{d}}}{\vdash \alpha \wedge \beta, \neg\beta, \neg\alpha}\,\mathsf{Int_d}}{\neg(\alpha \wedge \beta) \vdash \neg\beta, \neg\alpha}\,\neg_{\mathsf{e}}}{\neg(\alpha \wedge \beta) \vdash \neg\alpha, \neg\beta}\,\mathsf{Int_d}}{\neg(\alpha \wedge \beta) \vdash \neg\alpha \vee \neg\beta}\,\vee_{\mathsf{d}}}{\vdash \neg(\alpha \wedge \beta) \to (\neg\alpha \vee \neg\beta)}\,\to_{\mathsf{d}}
$$

Teorema 10.2.2 *Se $\alpha \in L_P$ tem uma prova por dedução natural então existe uma dedução no cálculo de seqüentes de Gentzen para α.*

Demonstração: Primeiro expressaremos cada uma das regras da dedução natural como seqüentes.

$$\mathsf{I}\wedge : \frac{\Gamma \vdash \alpha \quad \Gamma \vdash \beta}{\Gamma \vdash \alpha \wedge \beta} \qquad \mathsf{E}\wedge_1 : \frac{\Gamma \vdash \alpha \wedge \beta}{\Gamma \vdash \alpha} \qquad \mathsf{E}\wedge_2 : \frac{\Gamma \vdash \alpha \wedge \beta}{\Gamma \vdash \beta}$$

$$\mathsf{I}\vee_1 : \frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha \vee \beta} \qquad \mathsf{I}\vee_2 : \frac{\Gamma \vdash \beta}{\Gamma \vdash \alpha \vee \beta} \qquad \mathsf{E}\vee : \frac{\Gamma \vdash \alpha \vee \beta \quad \Gamma, \alpha \vdash \gamma \quad \Gamma, \beta \vdash \gamma}{\Gamma \vdash \gamma}$$

$$\mathsf{I} \to : \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \qquad \mathsf{E} \to : \frac{\Gamma \vdash \alpha \quad \Gamma \vdash \alpha \to \beta}{\Gamma \vdash \beta}$$

$$\mathsf{I}\neg : \frac{\Gamma, \alpha \vdash \beta \quad \Gamma, \alpha \vdash \neg\beta}{\Gamma \vdash \neg\alpha} \qquad \mathsf{E}\neg : \frac{}{\Gamma, \neg\neg\alpha \vdash \alpha}$$

A seguir mostraremos que para cada regra de dedução natural existe uma dedução no estilo Gentzen. Claramente as regras $\mathsf{I}\wedge$ e $\mathsf{I} \to$ são casos particulares das regras $\wedge_{\mathsf{d}}$ e $\to_{\mathsf{d}}$,

respectivamente, onde $\Delta = \emptyset$. Já as outras regras (as regras de eliminação e as regras $I\vee_2$ e $I\neg$) devem ser obtidas de uma maneira indireta.

- $E\wedge_1$:

$$\textsf{Cor}: \dfrac{\Gamma \vdash \alpha \wedge \beta \qquad \dfrac{\textsf{Inc}: \dfrac{}{\Gamma, \alpha \vdash \alpha}}{\textsf{Emf}_e: \dfrac{\Gamma, \alpha \vdash \alpha}{\Gamma, \alpha, \beta \vdash \alpha}}}{\wedge_e: \dfrac{\Gamma, \alpha, \beta \vdash \alpha}{\Gamma, \alpha \wedge \beta \vdash \alpha}}}{\Gamma \vdash \alpha}$$

- $I\vee_1$:

$$\vee_d: \dfrac{\textsf{Emf}_d: \dfrac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha, \beta}}{\Gamma \vdash \alpha \vee \beta}$$

- $E\vee$:

$$\textsf{Cor}: \dfrac{\Gamma \vdash \alpha \vee \beta \qquad \vee_e: \dfrac{\Gamma, \alpha \vdash \gamma \qquad \Gamma, \beta \vdash \gamma}{\Gamma, \alpha \vee \beta \vdash \gamma}}{\Gamma \vdash \gamma}$$

- $E \rightarrow$:

$$\textsf{Cor}: \dfrac{\Gamma \vdash \alpha \rightarrow \beta \qquad \rightarrow_e: \dfrac{\textsf{Emf}_d: \dfrac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha, \beta} \quad \textsf{Inc}: \dfrac{}{\Gamma, \beta \vdash \beta}}{\Gamma, \alpha \rightarrow \beta \vdash \beta}}{\Gamma \vdash \beta}$$

- $I\neg$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \beta, \neg\alpha}\,\neg_{\mathsf{d}}}{\Gamma, \neg\beta \vdash \neg\alpha}\,\neg_{\mathsf{e}} \qquad \dfrac{\Gamma, \alpha \vdash \neg\beta}{\Gamma \vdash \neg\beta, \neg\alpha}\,\neg_{\mathsf{d}}}{\Gamma \vdash \neg\alpha}\,\mathsf{Cor}$$

- $\mathsf{E}\neg$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \alpha \vdash \alpha}\,\mathsf{Inc}}{\Gamma \vdash \neg\alpha, \alpha}\,\neg_{\mathsf{d}}}{\Gamma, \neg\neg\alpha \vdash \alpha}\,\neg_{\mathsf{e}}$$

As regras $\mathsf{E}\wedge_2$ e $\mathsf{I}\vee_2$ são obtidas de maneira análogas às regras $\mathsf{E}\wedge_1$ e $\mathsf{I}\vee_1$, respectivamente.

Considerando a familiaridade das regras formuladas dessa maneira e a idéia básica de que uma prova no cálculo de seqüente registra, a cada etapa, o seqüente que foi provado até essa etapa, fica fácil de ver como uma prova, originalmente escrita como uma prova de dedução natural, pode agora ser reescrita como uma prova no cálculo de seqüentes. Portanto, para uma dada dedução natural de um $\alpha \in L_P$, substituindo cada regra de dedução natural usada na prova pela árvore de dedução de Gentzen correspondente, sempre poderemos construir uma dedução no cálculo de seqüentes de Gentzen para $\vdash \alpha$. ■

Exemplo 10.2.3 *Seja a prova do exemplo 10.1.2. Podemos escrever uma prova equivalente para ela no cálculo de seqüentes, é dada pela seguinte árvore.*

$$\dfrac{\Gamma \vdash \beta \vee \alpha \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \beta \vdash \beta}\,\mathsf{Inc}}{\Gamma, \beta \vdash \alpha, \beta}\,\mathsf{Emf}_{\mathsf{d}}}{\Gamma, \beta \vdash \alpha \vee \beta}\,\vee_{\mathsf{d}} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, \alpha \vdash \alpha}\,\mathsf{Inc}}{\Gamma, \alpha \vdash \alpha, \beta}\,\mathsf{Emf}_{\mathsf{d}}}{\Gamma, \alpha \vdash \alpha \vee \beta}\,\vee_{\mathsf{d}}}{\Gamma, \beta \vee \alpha \vdash \alpha \vee \beta}\,\vee_{\mathsf{e}}}{\Gamma \vdash \alpha \vee \beta}\,\mathsf{Cor}$$

10.2.2 Lógica de Predicados

Analogamente a dedução natural, pelo fato da lógica de predicados ser uma extensão da lógica proposicional, só incorporaremos as respectivas regras lógicas para os quantificadores universal e existencial, assim como uma restrição ao uso da regra $\rightarrow_{\mathsf{d}}$ e à regra $\neg_{\mathsf{d}}$, que é equivalente a aplicar o teorema da dedução.

Regras Lógicas

$$\forall_{\mathsf{d}} : \frac{\Gamma \vdash \alpha[t/x], \Delta}{\Gamma \vdash \forall x.\alpha, \Delta} \qquad \forall_{\mathsf{e}} : \frac{\Gamma, \alpha[t/x] \vdash \Delta}{\Gamma, \forall x.\alpha \vdash \Delta}$$

$$\exists_{\mathsf{d}} : \frac{\Gamma \vdash \alpha[t/x], \Delta}{\Gamma \vdash \exists x.\alpha, \Delta} \qquad \exists_{\mathsf{e}} : \frac{\Gamma, \alpha[a/x] \vdash \Delta}{\Gamma, \exists x.\alpha \vdash \Delta}$$

Em todos os casos, t é um termo livre para x em α. Em $\exists_{\mathsf{e}}$, a constante a não deve ocorrer em Γ, Δ nem α.

As restrições impostas ao uso do teorema da dedução aqui também são necessárias, uma vez que a regra $\rightarrow_{\mathsf{d}}$ é uma versão de seqüentes de este teorema e a regra $\neg_{\mathsf{d}}$ incorpora implicitamente este teorema. Logo, devemos restringir o uso de esta regra: a regra $\rightarrow_{\mathsf{d}}$ só poderá ser aplicada, no cálculo de seqüentes de Gentzen da lógica de predicados, quando no ramo que deriva $\Gamma, \alpha \vdash \beta, \Delta$ não se aplicou nenhuma regra $\forall_{\mathsf{d}}$ ou $\exists_{\mathsf{e}}$, introduzindo uma variável x que é livre em α.

Exemplo 10.2.4 *A seguir daremos uma prova do seqüente* $\forall x.\alpha \vdash \exists x.\alpha$

$$\begin{array}{ll} \mathsf{Inc}: & \dfrac{}{\alpha \vdash \alpha} \\ \forall_{\mathsf{e}}: & \dfrac{\alpha \vdash \alpha}{\forall x.\alpha \vdash \alpha} \\ \exists_{\mathsf{d}}: & \dfrac{\forall x.\alpha \vdash \alpha}{\forall x.\alpha \vdash \exists x.\alpha} \end{array}$$

Considerando que x é livre para x em α, para qualquer variável x e qualquer fórmula α, temos que $\alpha[x/x]$ é α. Assim, as aplicações das regras $\forall_e$ e $\exists_d$ nesta prova são corretas.

Note que se desejarmos provar a reversa deste teorema, isto é a fórmula $\exists x.\alpha \rightarrow \forall x.\alpha$, não conseguiríamos. Uma tentativa de prova seria a seguinte

$$\begin{array}{ll} \mathsf{Inc}: & \dfrac{}{\alpha[a/x] \vdash \alpha[a/x]} \\ \exists_{\mathsf{e}}: & \dfrac{\alpha[a/x] \vdash \alpha[a/x]}{\exists x.\alpha \vdash \alpha[a/x]} \\ \forall_{\mathsf{d}}: & \dfrac{\exists x.\alpha \vdash \alpha[a/x]}{\exists x.\alpha \vdash \forall x.\alpha} \end{array}$$

Mas ao aplicar a regra $\exists_e$ nesta prova, estamos transgredindo a restrição de a não ocorrer em Δ.

Como visto no teorema 10.2.2, para cada dedução natural de uma fórmula bem formada de L_P temos uma dedução no cálculo de seqüentes de Gentzen. Assim, resulta natural pensarmos que o mesmo é válido para deduções naturais de fórmulas bem formadas de LP.

Teorema 10.2.5 *Seja $\alpha \in LP$. Se existe uma dedução natural para α então existe uma dedução no cálculo de seqüente de Gentzen para α.*

Demonstração: Como a lógica predicados estende a lógica proposicional e, como demonstrado no teorema 10.2.2, se $\alpha \in L_P$ tem uma dedução natural, então α tem uma dedução no cálculo de seqüentes de Gentzen, aqui só realizaremos a prova da parte não incluída na prova do teorema 10.2.2. Analogamente à demonstração do teorema 10.2.2, primeiro expressaremos as regras de dedução natural para os quantificadores em termos de seqüentes.

$\mathsf{I}\forall : \dfrac{\Gamma \vdash \alpha}{\Gamma \vdash \forall x.\alpha}$

$\mathsf{E}\forall : \dfrac{\Gamma \vdash \forall x.\alpha}{\Gamma \vdash \alpha[t/x]}$

$\mathsf{I}\exists : \dfrac{\Gamma \vdash \alpha[t/x]}{\Gamma \vdash \exists x.\alpha}$, onde t é livre para x em α

$\mathsf{E}\exists : \dfrac{\Gamma \vdash \exists x.\alpha}{\Gamma \vdash \alpha[a/x]}$, onde a é uma constante nova na dedução de $\Gamma \vdash \exists x.\alpha$

Claramente, as regras de introdução são instâncias das regras $\forall_\mathsf{d}$ e $\exists_\mathsf{d}$, respectivamente. Assim, só precisamos provar as regras $\mathsf{E}\forall$ e $\mathsf{E}\exists$.

- $\mathsf{E}\forall$:

$$\mathsf{Cor}: \dfrac{\Gamma \vdash \forall x.\alpha \qquad \forall_\mathsf{e}: \dfrac{\mathsf{Inc}: \dfrac{}{\Gamma, \alpha[t/x] \vdash \alpha[t/x]}}{\Gamma, \forall x.\alpha \vdash \alpha[t/x]}}{\Gamma \vdash \alpha[t/x]}$$

- $\mathsf{E}\exists$:

$$\mathsf{Cor}: \dfrac{\Gamma \vdash \exists x.\alpha \qquad \exists_\mathsf{e}: \dfrac{\mathsf{Inc}: \dfrac{}{\Gamma, \alpha[a/x] \vdash \alpha[a/x]}}{\Gamma, \exists x.\alpha \vdash \alpha[a/x]}}{\Gamma \vdash \alpha[a/x]}$$

Como a restrição sobre o uso da regra $\mathsf{I} \rightarrow$ em dedução natural e da regra $\rightarrow_{\mathsf{d}}$ no cálculo de seqüente de Gentzen é a mesma, então podemos transformar cada dedução natural de uma fórmula $\alpha \in LP$ numa dedução no cálculo de seqüentes de Gentzen para α. ■

10.3 Estilo Tableaux

O método de tableau para a lógica clássica, devido a Jaako Hintikka (1929-) e Evert Beth (1908-1964) em [Hin55, Bet59], é semelhante ao método de resolução, no sentido que uma prova tableau é uma prova por redução ao absurdo. Começamos com uma hipótese e se a partir dessa hipótese derivamos uma conseqüência impossível, então podemos concluir que a hipótese original é impossível. Se a prova não foi bem sucedida, e nenhuma conseqüência impossível é vislumbrada, então em alguns casos podemos concluir que não existe erro com a hipótese aberta, isto é, a hipótese pode ser verdadeira para alguma interpretação. Existem casos nos quais podemos mostrar que a procura por uma conseqüência impossível foi exaustiva, e poderíamos ter encontrado uma conseqüência impossível se uma existisse. Mas nem todos os casos são assim. Algumas vezes tudo o que podemos dizer sobre a prova é que ainda não chegamos a nenhuma conseqüência impossível e que não sabemos se isto irá acontecer caso continuemos a prova. As assunções que são pesquisadas dessa maneira são assunções de certas fórmulas com determinados valores verdades. O caso mais simples, e que será usado neste texto, é quando assumimos que todas as fórmulas em questão são verdadeiras. Se estas assunções levam a uma conseqüência impossível, então temos mostrado que as fórmulas são inconsistentes. Porém, podemos também considerar a assunção que todas as fórmulas são falsas ou que algumas são falsas e outras verdadeiras (por exemplo como em [Bos97]).

Embora o método tableau para prova de teoremas em lógica clássica tenha se mostrado bastante satisfatório no estudo de lógica, acreditava-se que ele não teria uma aplicação computacional satisfatória devido à dificuldade de se controlar as várias instanciações de uma variável quantificada universalmente e pelo espaço de busca necessário para essas instanciações. Entretanto, a experiência mostrou o contrário, e inclusive podemos incorporar a ele o algoritmo da unificação para se obter um provador automático de teoremas tão eficiente quanto os considerados mais eficientes usando o método de resolução [Cos92].

10.3.1 Lógica Proposicional

Notemos que as fórmulas do tipo $\alpha \wedge \neg\alpha$ são insatisfatíveis para qualquer interpretação e uma fórmula da forma $\alpha \vee \beta$ é insatisfatível se e somente se ambas, α e β, são insatisfatíveis. Assim, para verificarmos se uma fórmula α é insatisfatível, poderíamos reescrever α na sua forma normal disjuntiva e observar se cada conjunção contém um subconjunto insatisfatível (isto é, se contém uma subfórmula do tipo $\alpha \wedge \neg\alpha$), então α é insatisfatível.

Porém isto não será necessário, uma vez que podemos incluir regras tableau específicas para cada conectivo tendo presente as equivalências lógicas:

- $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$
- $\neg(\alpha \rightarrow \beta) \equiv \alpha \wedge \neg\beta$
- $\neg\neg\alpha \equiv \alpha$
- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

Assim, o primeiro a se fazer para se provar uma fórmula α usando o sistema tableau é colocar na forma normal disjuntiva a fórmula $\neg\alpha$, dispondo-a numa estrutura de árvore, onde cada conjunção da fórmula ocupe um ramo da árvore. Depois procura-se subfórmulas insatisfatíveis em todos os ramos da árvore. Se todos os ramos da árvore são insatisfatíveis, então $\neg\alpha$ é insatisfatível e, portanto, α é logicamente válida.

O sistema tableau tem como base um conjunto de **regras de expansão**, as quais são divididas em duas categorias:

Regras de Expansão do tipo A

$$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{\neg(\alpha \vee \beta)}{\neg\alpha} \qquad \frac{\neg(\alpha \rightarrow \beta)}{\alpha} \qquad \frac{\neg\neg\alpha}{\alpha}$$

$$\beta \qquad\qquad \neg\beta \qquad\qquad \neg\beta$$

Regras de Expansão do tipo B

$$\frac{\alpha \vee \beta}{\alpha \mid \beta} \qquad \frac{\alpha \rightarrow \beta}{\neg\alpha \mid \beta} \qquad \frac{\neg(\alpha \wedge \beta)}{\neg\alpha \mid \neg\beta}$$

A idéia é pôr as duas subfórmulas imediatas de uma conjunção no mesmo ramo, o que é feito com a aplicação de uma regra do tipo A. Analogamente, as subfórmulas imediatas de uma disjunção devem bifurcar o ramo original, o que é conseguido com a aplicação de uma regra do tipo B.

Exemplo 10.3.1 *Vamos provar o teorema $\alpha \rightarrow (\beta \rightarrow \alpha)$ usando o método tableau.*

1)	$\neg(\alpha \to (\beta \to \alpha))$	*(o teorema negado)*
2)	α	*(1, Regra A)*
3)	$\neg(\beta \to \alpha)$	*(1, Regra A)*
4)	β	*(3, Regra A)*
5)	$\neg\alpha$	*(3, Regra A)*

Os números não são parte oficial tableau, mas foram acrescentados para melhor falar sobre a prova. Em particular, podemos dizer que no único ramo do tableau acima encontramos duas fórmulas contraditórias (2 e 5). Assim, a negação do teorema nos leva a um absurdo e portanto provamos o teorema.

Agora definiremos a noção de prova tableau formalmente.

Definição 10.3.2 *Seja α uma fórmula.*

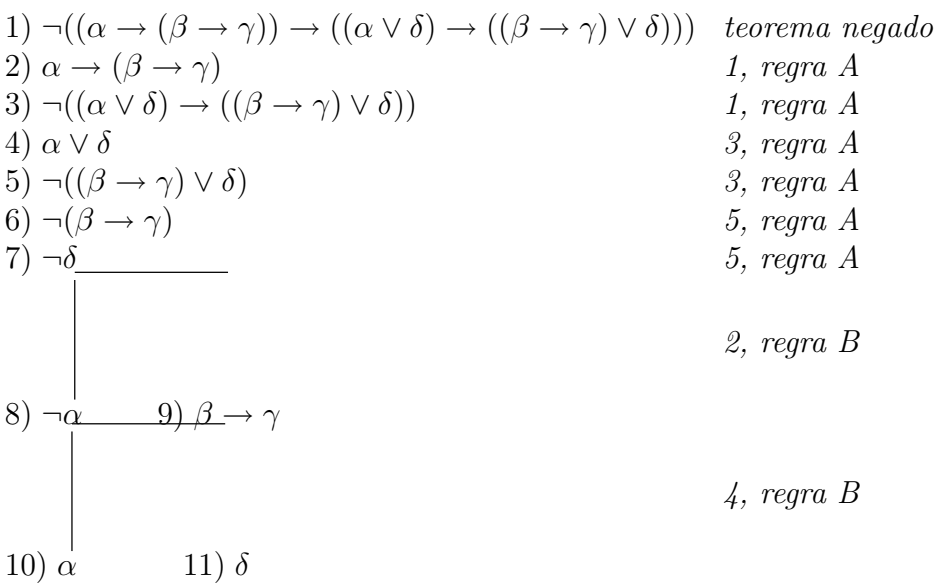
1. *α é um* **tableau**.
2. *Se $\mathcal{T}$ é um tableau e $\mathcal{T}^*$ resulta de $\mathcal{T}$ ao aplicar uma das regra de expansão tableau, então $\mathcal{T}^*$ também é um* **tableau**.

Definição 10.3.3 *Um ramo Δ de um tableau $\mathcal{T}$ é um* **ramo fechado** *se ele contiver α e $\neg\alpha$, para alguma fórmula α.*

Definição 10.3.4 *Um tableau $\mathcal{T}$ é um* **tableau fechado** *se cada um de seus ramos for fechado.*

Definição 10.3.5 *Uma* **prova tableau** *de α é um tableau fechado cuja raiz é $\neg\alpha$. α é um* **teorema** *do sistema tableau, se α tem uma prova tableau.*

Exemplo 10.3.6 *Uma prova tableau (com rótulos adicionados para referenciar) de $(\alpha \to (\beta \to \gamma)) \to ((\alpha \vee \delta) \to ((\beta \to \gamma) \vee \delta))$ é dada abaixo.*



Nesta prova, 1. é a negação da fórmula a ser provada; 2. e 3. saem de 1. pela regra do tipo A; 4. e 5. saem de 3 pela regra do tipo A; 6 e 7 são obtidas de 5 pela aplicação da regra de tipo A, 8 e 9 de 2 pela aplicação da regra de tipo B e, finalmente, 10. e 11. são obtidas de 4. pela regra de tipo B. Lendo da esquerda para direita, os ramos são fechados uma vez que 8. e 10., 7. e 11. e 6. e 9. são contraditórias. Note que num dos ramos fechados temos uma fórmula não atômica.

A seguir estenderemos o conceito de prova tableau para seqüentes.

Definição 10.3.7 *Sejam Γ e Δ conjuntos de fórmulas. Uma* **prova tableau para o seqüente** $\Gamma \vdash \alpha$ *é uma prova tableau de $\alpha_1 \to (\alpha_2 \to (\cdots(\alpha_n \to \alpha)\cdots)$, onde $\alpha_1, \ldots, \alpha_n \in \Gamma$. Uma prova tableau para o seqüente $\Gamma \vdash \Delta$ é uma prova tableau de $\Gamma \vdash \alpha$ para algum $\alpha \in \Delta$.*

Exemplo 10.3.8 *Uma prova tableau para $\alpha \vdash \neg\neg\alpha$ é a seguinte prova tableau para $\alpha \to \neg\neg\alpha$.*

1)	$\neg(\alpha \to \neg\neg\alpha)$	*conclusão negada*
2)	$\neg\alpha$	*1, regra A.*
3)	$\neg\neg\alpha$	*1,regra A*

Exemplo 10.3.9 *Uma prova tableau de $\alpha \to \beta, \beta \to \gamma \vdash \alpha \to \gamma$ é dada abaixo.*

1) $\alpha \to \beta$ *hipóteses*
2) $\beta \to \gamma$ *hipótese*
3) $\neg(\alpha \to \gamma)$ *conclusão negada*
4) α *3, regra A*
5) $\neg\gamma$ *3, regra A*

1, regra B

6) $\neg\alpha$ 7) β

2, regra B

8) $\neg\beta$ 9) γ

Teorema 10.3.10 *Se existe uma derivação no estilo de Gentzen para $\alpha \in L_P$, então existe uma prova tableau para α.*

DEMONSTRAÇÃO: Para provar este teorema, é suficiente mostrar que cada regra no cálculo de seqüentes de Gentzen tem uma prova tableau, isto é, se a regra é da forma

$$\frac{\Gamma \vdash \Delta}{\Gamma' \vdash \Delta'}$$

supomos que contamos com uma prova tableau de $\Gamma \vdash \Delta$ e a partir dela construímos uma prova tableau para $\Gamma' \vdash \Delta'$.

Por simplicidade consideraremos em cada uma das regras que $\Gamma = \Theta = \emptyset$ e $\Delta = \emptyset$ ou $\Delta = \beta$ (ou ainda igual a γ), segundo seja o caso. Primeiro reescreveremos as regras usando esta simplificação e as considerações na definição de prova tableau para seqüentes (definição 10.3.7).

Regras Estruturais

$$\mathsf{Inc}: \frac{}{\vdash \alpha \to \alpha} \qquad \mathsf{Cor}: \frac{\vdash \alpha \quad \vdash \alpha \to \beta}{\vdash \beta}$$

$$\mathsf{Emf_e}: \frac{\vdash \beta}{\vdash \alpha \to \beta} \qquad \mathsf{Emf_d}: \frac{\vdash \beta}{\vdash \alpha \vee \beta}$$

$$\mathsf{Int_e}: \frac{\vdash \alpha \to (\beta \to \gamma)}{\vdash \beta \to (\alpha \to \gamma))} \qquad \mathsf{Int_d}: \frac{\vdash \alpha, \beta}{\vdash \beta, \alpha}$$

$$\mathsf{Con_e}: \frac{\vdash \alpha \to (\alpha \to \beta)}{\vdash \alpha \to \beta} \qquad \mathsf{Con_d}: \frac{\vdash \alpha, \alpha}{\vdash \alpha}$$

Regras Lógicas

$$\wedge_{\mathsf{d}}: \frac{\vdash \alpha \quad \vdash \beta}{\vdash \alpha \wedge \beta} \qquad \wedge_{\mathsf{e}}: \frac{\vdash \alpha \to (\beta \to \gamma))}{\vdash (\alpha \wedge \beta) \to \gamma}$$

$$\vee_{\mathsf{d}}: \frac{\vdash \alpha, \beta}{\vdash \alpha \vee \beta} \qquad \vee_{\mathsf{e}}: \frac{\vdash \alpha \to \gamma \quad \vdash \beta \to \gamma}{\vdash (\alpha \vee \beta) \to \gamma}$$

$$\to_{\mathsf{d}}: \frac{\vdash \alpha \to \beta}{\vdash \alpha \to \beta} \qquad \to_{\mathsf{d}}: \frac{\vdash \alpha, \gamma \quad \vdash \beta \to \gamma}{\vdash (\alpha \to \beta) \to \gamma}$$

$$\neg_{\mathsf{d}}: \frac{\vdash \alpha \to \beta}{\vdash \neg\alpha, \beta} \qquad \neg_{\mathsf{e}}: \frac{\vdash \alpha, \beta}{\vdash \neg\alpha \to \beta}$$

- Inc :

  1) $\neg(\alpha \to \alpha)$ teorema negado
  2) α 1, regra A
  3) $\neg\alpha$ 1, regra A

- Cor : Seja $\Delta_{\neg\alpha}$ uma prova tableau de α. Por outro lado, uma prova tableau de $\alpha \to \beta$ tem a seguinte forma:

  1) $\neg(\alpha \to \beta)$ teorema negado
  2) α 1, regra A
  3) $\neg\beta$ 1, regra A
     Δ

  Como Δ precedido de α não pode ser uma prova de algum γ tal que $\alpha = \neg\gamma$, pois isso contradiz a hipótese de que temos uma prova para α ($\Delta_{\neg\alpha}$). Logo,

  1) $\neg\beta$
     Δ

  é uma prova tableau de β.

- $\mathsf{Emf_e}$: Se $\Delta_{\neg\beta}$ é uma prova tableau de β então o seguinte tableau é uma prova de $\alpha \to \beta$.

  1) $\neg(\alpha \to \beta)$ teorema negado
  2) α 1, regra A
     $\Delta_{\neg\beta}$

- $\mathsf{Emf_d}$: Se $\Delta_{\neg\beta}$ é uma prova tableau de β então, pela definição 10.3.7, ela também é uma prova tableau de $\vdash \alpha, \beta$.

- $\mathsf{Int_e}$: Uma prova de $\alpha \to (\beta \to \gamma)$ deve ter a seguinte forma

  1) $\neg(\alpha \to (\beta \to \gamma))$ teorema negado
  2) α 1, regra A
  3) $\neg(\beta \to \gamma)$ 1, regra A
  4) β 3, regra A
  5) $\neg\gamma$ 3, regra A
     Δ

  Claramente, o tableau

  1) $\neg(\beta \to (\alpha \to \gamma))$ teorema negado
  2) β 1, regra A
  3) $\neg(\alpha \to \gamma)$ 1, regra A
  4) α 3, regra A
  5) $\neg\gamma$ 3, regra A
     Δ

  é uma prova de $\beta \to (\alpha \to \gamma)$.

- $\mathsf{Int_d}$: Direta da definição 10.3.7.

- $\mathsf{Con_e}$: Uma prova de $\alpha \to (\alpha \to \beta)$ deve ter a seguinte forma:

  1) $\neg(\alpha \to (\alpha \to \beta))$ teorema negado
  2) α 1, regra A
  3) $\neg(\alpha \to \beta)$ 1, regra A
  4) α 3, regra A
  5) $\neg\beta$ 3, regra A

  Δ

  Logo, o seguinte tableau é uma prova de $\alpha \to \beta$:

  1) $\neg(\alpha \to \beta)$ teorema negado
  2) α 1, regra A
  3) $\neg\beta$ 1, regra A

  Δ

- $\mathsf{Con_d}$: Se Δ é uma prova tableau de $\vdash \alpha, \alpha$ então, pela definição 10.3.7, ela é uma prova de α.

- $\wedge_\mathsf{d}$: Se $\Delta_{\neg\alpha}$ e $\Delta_{\neg\beta}$ são provas tableau de α e β, respectivamente, então o seguinte tableau é uma prova tableau de $\alpha \wedge \beta$.

  1) $\neg(\alpha \wedge \beta)$ teorema negado

  $\Delta_{\neg\alpha}$ $\quad$ $\Delta_{\neg\beta}$

- $\wedge_\mathsf{e}$: Uma prova tableau de $\alpha \to (\beta \to \gamma)$ deve ter a seguinte forma

  1) $\neg(\alpha \to (\beta \to \gamma))$ teorema negado
  2) α 1, regra A
  3) $\neg(\beta \to \gamma)$ 1, regra A
  4) β 3, regra A
  5) $\neg\gamma$ 3, regra A

  Δ

  Logo, o seguinte tableau é uma prova para $(\alpha \wedge \beta) \to \gamma$.

  1) $\neg((\alpha \wedge \beta) \to \gamma)$ teorema negado
  2) $\alpha \wedge \beta$ 1, regra A
  3) $\neg\gamma$ 1, regra A
  4) α 2, regra A
  5) β 2, regra A

  Δ

- $\vee_\mathsf{d}$: Se Δ é uma prova de $\vdash \alpha, \beta$ então pela definição 10.3.7, Δ é uma prova de α ou de β. Se Δ é uma prova de α então

1) $\neg(\alpha \vee \beta)$ teorema negado
2) $\neg\beta$ 1, regra A
Δ

é uma prova tableau de $\alpha \vee \beta$. Analogamente, se Δ é uma prova tableau de β, então

1) $\neg(\alpha \vee \beta)$ teorema negado
2) $\neg\alpha$ 1, regra A
Δ

é uma prova tableau de $\alpha \vee \beta$.

- $\vee_{\mathsf{e}}$: Uma prova tableau de $\alpha \to \gamma$ tem a seguinte forma:

1) $\neg(\alpha \to \gamma)$ teorema negado
2) α 1, regra A
3) $\neg\gamma$ 1, regra A
Δ_1

e uma prova tableau para $\beta \to \gamma$ deve ter a seguinte forma

1) $\neg(\beta \to \gamma)$ teorema negado
2) β 1, regra A
3) $\neg\gamma$ 1, regra A
Δ_2

Se

1) $\neg\gamma$ teorema negado
Δ_i

para algum $i = 1, 2$, então, trivialmente,

1) $\neg((\alpha \vee \beta) \to \gamma)$ teorema negado
2) $\alpha \vee \beta$ 1, regra A
3) $\neg\gamma$ 1,regra A
Δ_i

é uma prova tableau de $(\alpha \vee \beta) \to \gamma$. Se não, isto é, se

1) α 1) β
Δ_1 Δ_2

são tableau fechado, então o seguinte tableau é uma prova de $(\alpha \vee \beta) \to \gamma$

1) $\neg((\alpha \vee \beta) \to \gamma)$ teorema negado
2) $\alpha \vee \beta$ 1, regra A
3) $\neg\gamma$ 1, regra A
1, regra A

4) α 5) β
Δ_1 Δ_2

onde 4) e 5) saíram de 2 aplicando a regra do tipo B.

- $\rightarrow_{\mathsf{d}}$: Direta.

- $\rightarrow_{\mathsf{e}}$: Seja Δ_1 uma prova tableau de $\vdash \alpha, \gamma$. Uma prova tableau de $\beta \rightarrow \gamma$ tem a seguinte forma.

  1) $\neg(\beta \rightarrow \gamma)$ teorema negado
  2) β 1, regra A
  3) $\neg\gamma$ 1, regra A
  Δ_2

  Uma prova tableau de $(\alpha \rightarrow \beta) \rightarrow \gamma$ é

  1) $\neg((\alpha \rightarrow \beta) \rightarrow \gamma)$ teorema negado
  2) $\alpha \rightarrow \beta$ 1, regra A
  3) $\neg\gamma$ 1, regra A

  4) $\neg\alpha$ Δ_1 | 5) β Δ_2

- $\neg_{\mathsf{d}}$: Uma prova tableau de $\alpha \rightarrow \beta$ tem a seguinte forma,

  1) $\neg(\alpha \rightarrow \beta)$ teorema negado
  2) α 1, regra A
  3) $\neg\beta$ 1, regra A
  Δ

  Logo,

  1) $\neg\neg\alpha$ teorema negado
  2) α 1, regra A
  Δ

  é uma prova de $\neg\alpha$ ou

  1) $\neg\beta$ teorema negado
  Δ

  é uma prova tableau de β. Portanto, uma das duas é uma prova tableau de $\vdash \neg\alpha, \beta$

- $\neg_{\mathsf{e}}$: Se Δ é uma prova tableau de α ou β então o seguinte tableau é uma prova de $\neg\alpha \rightarrow \beta$.

  1) $\neg(\neg\alpha \rightarrow \beta)$ teorema negado
  2) $\neg\alpha$ 1, regra A
  3) $\neg\beta$ 1, regra A
  Δ

Assim, é fácil ver que podemos transformar qualquer dedução no estilo de Gentzen de um seqüente numa prova tableau. ■

10.3.2 Lógica de Predicados

Analogamente como fizemos com o estilo de dedução natural e com o cálculo de seqüentes de Gentzen, para definir o sistema tableau para a lógica de predicados só acrescentaremos a regras específicas dos quantificadores, pois, a menos do teorema da dedução, tudo que foi definido para a lógica proposicional vale aqui.

Antes de darmos as regras de extensão para os quantificadores, é bom lembrar as equivalências

$$\forall x.\alpha \equiv \neg\exists x.\neg\alpha \quad \text{e} \quad \exists x.\alpha \equiv \neg\forall x.\neg\alpha$$

Regras de Expansão do tipo C

$$\frac{\forall x.\alpha}{\alpha[t/x]} \qquad \frac{\neg\exists x.\alpha}{\neg\alpha[t/x]}$$

Regras de Expansão do tipo D

$$\frac{\exists x.\alpha}{\alpha[a/x],} \quad \frac{\neg\forall x.\alpha}{\neg\alpha[a/x],} \quad \text{onde } a \text{ é uma constante nova para o tableau}$$

Exemplo 10.3.11 *Uma prova tableau do teorema* $\forall x.P(x) \rightarrow \exists y.P(y)$ *é dada a seguir:*

1. $\neg(\forall x.P(x) \rightarrow \exists y.P(y))$ *teorema negado*
2. $\forall x.P(x)$ *1, regra A*
3. $\neg\exists y.P(y)$ *1, regra A*
4. $P(b)$ *2, regra C*
5. $\neg P(b)$ *3, regra C*

onde 1. é o teorema a provar; 2. e 3. saem deste pela aplicação da regra A; 4. é obtida de 2. pela aplicação da regra C e 5. pela aplicação da regra D a 3. Como 4. e 5. são contraditórios e temos um único ramo na árvore, concluímos que o tableau acima é fechado e portanto é uma prova do teorema desejado.

Observe que a definição de prova tableau para seqüentes (definição 10.3.7) admite o uso indiscriminado do teorema da dedução. Assim, as restrições impostas por este uso indiscriminado nos leva a introduzir uma definição mais forte de prova tableau.

Definição 10.3.12 *Seja* $\mathcal{T}$ *uma prova tableau para algum* $\alpha \in LP$. $\mathcal{T}$ *é uma* **prova tableau forte** *para* α *se a primeira regra de expansão usada em* $\mathcal{T}$ *é do tipo B, C, D, ou é a regra do tipo A*

$$\frac{\neg\neg\alpha}{\alpha}$$

A definição de prova tableau para seqüentes continua a mesma. Assim, teremos uma prova tableau para o seqüente $\alpha \vdash \forall x.\alpha$ mas não teremos uma prova tableau forte para $\vdash \alpha \rightarrow \forall x.\alpha$.

Teorema 10.3.13 *Se existe uma derivação no estilo de Gentzen para $\alpha \in LP$, então existe uma prova tableau forte para α.*

DEMONSTRAÇÃO: Para demonstrar este teorema, devemos considerar toda a prova do teorema 10.3.10 e acrescentar um prova tableau para as regras de Gentzen dos quantificadores. Antes, porém, simplificamos estas regras usando os mesmos critérios adotados na prova acima mencionada.

$$\forall_{\mathsf{d}} : \frac{\vdash \alpha[t/x]}{\forall x.\alpha} \qquad \forall_{\mathsf{e}} : \frac{\vdash \alpha[t/x] \rightarrow \beta}{\vdash \forall x.\alpha \rightarrow \beta}$$

$$\exists_{\mathsf{d}} : \frac{\vdash \alpha[a/x]}{\vdash \exists x.\alpha} \qquad \exists_{\mathsf{e}} : \frac{\vdash \alpha[a/x] \rightarrow \beta}{\vdash \exists x.\alpha \rightarrow \beta}$$

- $\forall_{\mathsf{d}}$: Se $\Delta_{\neg\alpha[a/x]}$ é uma prova tableau de $\alpha[a/x]$ (como t é qualquer termo livre para x em α, consideramos o termo constante a) então o seguinte tableau é uma prova de $\forall x.\alpha$

  1) $\neg\forall x.\alpha$ teorema negado
  $\Delta_{\neg\alpha[a/x]}$

- $\forall_{\mathsf{e}}$: Uma prova tableau de $\alpha \rightarrow \beta$ tem a seguinte forma,

  1) $\neg(\alpha \rightarrow \beta)$ teorema negado
  2) α 1,regra A
  3) $\neg\beta$ 1, regra A
  Δ

  Logo, o seguinte tableau é uma prova para $\forall x.\alpha \rightarrow \beta$.

  1) $\neg(\forall x.\alpha \rightarrow \beta)$ teorema negado
  2) $\forall x.\alpha$ 1, regra A
  3) $\neg\beta$
  4) α 2, regra C
  Δ

- $\exists_{\mathsf{d}}$: Se $\Delta_{\neg\alpha[a/x]}$ é uma prova de $\alpha[a/x]$ então o seguinte tableau é uma prova de $\exists x.\alpha$.

  1) $\neg\exists x.\alpha$ teorema negado
  $\Delta_{\neg\alpha[a/x]}$

- $\exists_{\mathsf{e}}$: Uma prova tableau de $\alpha[a/x] \rightarrow \beta$ tem a seguinte forma,

  1) $\neg(\alpha[a/x] \rightarrow \beta)$ teorema negado
  2) $\alpha[a/x]$ 1, regra A
  3) $\neg\beta$ 1, regra A
  Δ

  Uma prova tableau para $\exists x.\alpha \rightarrow \beta$ é a seguinte

  1) $\neg(\exists x.\alpha \rightarrow \beta)$ teorema negado
  2) $\exists x.\alpha$ 1, regra A
  3) $\neg\beta$ 1,regra A
  4) $\alpha[a/x]$ 2, regra D
  Δ

Assim, considerando estas provas tableau junto com as do teorema 10.3.10, podemos transformar qualquer dedução no estilo de seqüentes de Gentzen numa prova tableau do mesmo seqüente. ■

10.4 Sistema Tableau com Unificação

O procedimento de unificação de Robinson, apresentado no capítulo 7, pode ser adaptado para ser usado pelo sistema de tableau para lógica clássica de 1^a ordem, conforme mostrado por [Bow82, Ree85]. Nesta seção apresentamos o sistema tableau com unificação para a lógica de 1^a ordem e enunciamos sua equivalência com o sistema tableau sem unificação.

De aqui em diante só consideraremos fórmulas fechadas na forma normal Prenex livres de quantificadores existenciais. Assim, toda vez que comecemos aplicar o sistema tableau, começaremos com a forma normal de Skolem da negação da fórmula original. Em virtude desta restrição a Regra D não precisa ser mais considerada. Este sistema tableau é denominado com unificação porque usamos o procedimento de unificação de Robinson na obtenção de pares de fórmulas atômicas complementares. Assim aqui acharemos **tableau atomicamente fechados**, isto é tableau fechados onde cada ramo contém uma fórmula atômica e sua negação. Smullyan [Smu68] demonstrou que se uma fórmula é insatisfatível então existe um tableau atomicamente fechado para a fórmula. A maneira como podemos achar esse tableau atomicamente fechado é ilustrado no seguinte algoritmo:

Procedimento tableau com unificação:

1. Crie uma variável booleana para cada fórmula. Esta variável indicará se uma fórmula foi usada ou não (inicialmente todas as variáveis tem valor "falso").

2. A regra do tipo C é modificada para:

$$\frac{\forall x.\alpha}{\alpha[y/x]} \qquad \frac{\neg\exists x.\alpha}{\neg\alpha[y/x]}$$

onde a variável y é nova para o tableau. Esta nova regra será denominada de Regra C$'$.

3. A regra do tipo D é eliminada do sistema.
4. Comece o tableau escrevendo a negação da fórmula original.
5. Encontre a forma normal de Skolem (FNS) para a fórmula (negação da fórmula original).
6. ENQUANTO o tableau não for atomicamente fechado
   tente unificar as fórmula atômicas de cada ramo aberto com o seu complementar
   SE existir alguma fórmula não-atômica que não foi usada
   ENTÃO {
   faça α a primeira fórmula não-atômica (de cima para baixo) não usada
   PARA { cada ramo aberto Θ contendo α aplique a regra apropriada a α em Θ }
   marque α como uma fórmula usada
   }
   FIM-ENQUANTO
7. Pare com " Sucesso"
8. FIM

Para ilustrar este procedimento, vamos apresentar alguns exemplos.

Exemplo 10.4.1 *Vamos tentar provar a fórmula*

$$\forall x.\exists y.P(x,y) \rightarrow \exists z.\forall u.P(u,z) :$$

1.	$\forall x.\forall z.\neg(P(x,f(x)) \rightarrow P(g(x,z),z))$	*FNS da negação*
2.	$\forall z.\neg(P(v,f(v)) \rightarrow P(g(v,z),z))$	*1, Regra C$'$*
3.	$\neg(P(v,f(v)) \rightarrow P(g(v,w),w))$	*2, Regra C$'$*
4.	$P(v,f(v))$	*3, Regra A*
5.	$\neg P(g(v,w),w)$	*3, Regra A*

Claramente, os literais 4 e 5 não podem ser unificados. Portanto, a única regra que pode ser aplicada é a regra C em 1 ou em 2 obtendo-se uma cópia da fórmula 2 ou 3, respectivamente (mudando só o nome da variável) o qual só poderá gerar novas cópias das fórmulas 4 e 5. Logo, o tableau nunca poderá ser fechado.

Exemplo 10.4.2 *Provaremos usando o sistema tableau com unificação que a fórmula*

$$\forall x.(P(x,z) \wedge R(x,y)) \rightarrow (\exists x.P(x,z) \vee \forall x.R(x,y))$$

é um teorema.

1.	$\forall x_1.\forall x_2.((P(x_1,a) \wedge R(x_1,b)) \wedge (\neg P(x_2,a) \wedge \neg R(f(x_1,x_2),b)))$	*FNS da negação*
2.	$\forall x_2.((P(x,a) \wedge R(x,b)) \wedge (\neg P(x_2,a) \wedge \neg R(f(x,x_2),b)))$	*1, Regra C'*
3.	$((P(x,a) \wedge R(x,b)) \wedge (\neg P(y,a) \wedge \neg R(f(x,y),b)))$	*2, Regra C'*
4.	$P(x,a) \wedge R(x,b)$	*3, Regra A*
5.	$\neg P(y,a) \wedge \neg R(f(x,y),b)$	*3, Regra A*
6.	$P(x,a)$	*4, Regra A*
7.	$\neg P(y,a)$	*5, Regra A*
8.	$P(y,a)$	*de 6 aplicando* $\sigma[(y,x)]$

Como no caso do tableau tradicional (sem unificação) não temos a restrição de que a variável introduzida ao usar a regra C seja nova, poderíamos ter substituído em 3, ao aplicar a regra C, a variável x_2 por x. Assim, em 7, obteríamos $\neg P(x,a)$ em vez de $\neg P(y,a)$, fechando assim com 6 o único ramo do tableau. Assim, resulta natural pensarmos que podemos sempre construir uma prova tableau forte tradicional a partir de uma com unificação. A reversa também é razoável ser correta, isto é, sempre que tivermos uma prova tableau forte tradicional (de uma fórmula na forma normal de Skolem) também poderemos obter uma prova tableau com unificação. Por exemplo seja a seguinte prova tableau forte tradicional da fórmula

$$\alpha \equiv \forall x.(P(x,z) \wedge R(x,y)) \rightarrow (\exists x.P(x,z) \vee \forall x.R(x,y))$$

negando α e reescrevendo-o na sua forma normal de Skolem resulta na fórmula

$$\forall z.\forall y.\forall x_1.\forall x_2.(P(x_1,z) \wedge R(x_1,y) \wedge \neg(P(x_2,z) \vee R(f(z,y,x_1,x_2),y)))$$

Prova tableau forte tradicional:

1.	$\forall z.\forall y.\forall x_1.\forall x_2.((P(x_1,z) \wedge R(x_1,y)) \wedge \neg(P(x_2,z) \vee R(f(z,y,x_1,x_2),y)))$	FNS da negação de α
2.	$(P(a,z) \wedge R(a,y)) \wedge \neg(P(a,z) \vee R(f(z,y,a,a),y))$	1, Regra C, aplicada 4 vezes
3.	$P(a,z) \wedge R(a,y)$	2, Regra A
4.	$\neg(P(a,z) \vee R(f(z,y,a,a),y))$	2, Regra A
5.	$P(a,z)$	3, Regra A
6.	$\neg P(a,z)$	4, Regra A

Claramente esta prova pode ser modificada para uma prova tableau com unificação, simplesmente alterando todas as aplicações da regra C, na qual se substitua uma variável quantificada por um termo t por uma aplicação da Regra C modificada, isto é na qual se substitui a variável quantificada por uma nova variável. Assim, o tableau com unificação correspondente seria:

1.	$\forall z.\forall y.\forall x_1.\forall x_2.((P(x_1,z) \wedge R(x_1,y)) \wedge \neg(P(x_2,z) \vee R(f(z,y,x_1,x_2),y)))$	FNS da negação de α
2.	$(P(x_5,x_3) \wedge R(x_5,x_4)) \wedge \neg(P(x_6,x_3) \vee R(f(x_3,x_4,x_5,x_6),x_4))$	1, Regra C, aplicada 4 vezes
3.	$P(x_5,x_3) \wedge R(x_5,x_4)$	2, Regra A
4.	$\neg(P(x_6,x_3) \vee R(f(x_3,x_4,x_5,x_6),x_4))$	2, Regra A
5.	$P(x_5,x_3)$	3, Regra A
6.	$\neg P(x_6,x_3)$	4, Regra A
7.	$P(x_6,x_3)$	de 5 aplicando $\sigma[(x_6,x_5)]$

Teorema 10.4.3 *Seja α uma fórmula na forma normal de Skolem e $\mathcal{T}$ um tableau forte para α. Se $\mathcal{T}$ contém a fórmula $\gamma[t/x]$ obtida ao aplicar uma Regra C, então existe um tableau com unificação $\mathcal{T}'$ para α que contém $\gamma[y/x]$, obtida ao aplicar a Regra C', e tal que t é livre para y em γ.*

Demonstração: Para demonstrar este teorema usaremos indução nas fórmulas do tableau, $\mathcal{T} = \alpha_1, \alpha_2, \ldots, \alpha_n$ para construir um tableau (provavelmente não fechado) $\mathcal{T}_1 = \beta_1, \ldots, \beta_n$:

Base da indução: α_1 é forma normal de Skolem da negação do teorema a ser provado. Logo, $\beta_1 = \alpha_1$.

Etapa indutiva: Assuma que para $i < n$, já temos construído o tableau $\beta_1, \ldots, \beta_j$, com $j < m$. Se α_{i+1} não foi obtida pela aplicação da Regra C, então $\beta_{j+1} = \alpha_{i+1}$. Senão, isto é, $\alpha_{i+1} = \gamma[t/x]$ obtido pela aplicação da regra C, então faça $\beta_{j+1} = \gamma[y,x]$ como sendo obtido pela aplicação da regra C' (y é uma variável nova em $\beta_1, \ldots, \beta_j$).

Finalmente, para tornar este tableau atomicamente fechado, aplique o algoritmo da unificação para todos pares de fórmulas atômicas em cada um dos ramos do tableau. ■

Dessa maneira todo o que pode ser provado usando tableau tradicional também pode ser provado usando tableau com unificação. Observe que a restrição no teorema da fórmula α ser a forma normal de Skolem da negação da fórmula a ser provada é supérflua, uma vez que uma fórmula é um teorema se e somente se sua forma normal de Skolem também é.

10.5 Correspondência entre Tableau e Resolução

Robinson [Rob79] utilizou o sistema tableau para explicar o princípio da resolução. Por outro lado, Gallier em [Gal86] apresentou um algoritmo para transformar provas da lógica proposicional usando o estilo de seqüentes de Gentzen em refutações por resolução e vice-versa. Nesta seção apresentaremos uma adaptação, realizada em [Cos92], do método apresentado por Gallier, para obter um algoritmo para transformar provas tableau com unificação em refutações por resolução.

Teorema 10.5.1 *Existe um algoritmo para transformar uma refutação por tableau com unificação de um conjunto Γ de cláusulas da lógica clássica de 1^a ordem em uma refutação por resolução para Γ.*

DEMONSTRAÇÃO: A demonstração será por indução no número de ramos do tableau. Para verificar se o resultado está correto, será necessário verificar em cada etapa da indução se as seguintes condições são satisfeitas:

1. Existe uma refutação por resolução para Γ obtida a partir da prova tableau com unificação de Γ e ambas provas tem o mesmo unificador.
2. O número de resolventes é menor ou igual ao número de ramos fechados no tableau.
3. O conjunto de pares de literais usados nas etapas da resolução é um subconjunto do conjunto de pares de literais usados para se fechar o tableau.

Base da indução: O tableau tem só um ramo. Portanto, Γ contém cláusulas atômicas C_i e C_j tais que existe um unificador mais geral σ_0 para C_i e o complemento de C_j. Então a seguinte árvore é uma resolução para Γ

$$\begin{array}{cc} C_i & C_j \\ C_i\sigma_0 \text{———} & C_j\sigma_0 \end{array}$$

$$\square \text{———}$$

Assim, ambos tableau e resolução têm o mesmo unificador σ_0, a refutação por resolução tem um resolvente e o par de literais usados é o mesmo para usado para fecharmos o tableau. Logo, esta etapa da indução satisfaz as condições exigidas.

Etapa indutiva: O tableau tem mais de um ramo. Assim o tableau que foi fechado com um unificador digamos σ_0, é da forma

$$\begin{array}{c} C_1 \\ \vdots \\ C_k \\ \vdots \\ C_n \text{———} \end{array}$$

$$\begin{array}{cc} B_1 \text{———} & B_2 \end{array}$$

onde a cláusula C_k é da forma $C_i \vee C_j$, com $i, j < k$, e os ramos B_1 e B_2 são gerados por aplicação da regra B a C_k, de tal modo que C_i está em B_1 e C_j está em B_2. Como o tableau é fechado, cada um dos ramos deve também ser fechado. Suponha que os ramos

B_1 e B_2 sejam fechados e seus unificadores mais gerais sejam σ_1 e σ_2, respectivamente. Então, $\sigma_0 = \sigma_1 \circ \sigma_2$.

Pela hipótese indutiva,

1. Existem refutações por resolução D_1 e D_2 derivadas de B_1 e B_2 com unificadores mais gerais σ_1 e σ_2, respectivamente.

2. O número de resolventes de D_1 e D_2 são menores ou iguais ao número de ramos para se fechar B_1 e B_2, respectivamente.

3. O conjunto de pares de literais usados nos passos de D_1 e D_2 são subconjuntos dos conjuntos de pares de literais usados para se fechar B_1 e B_2, respectivamente.

Se cada cláusula de entrada de D_1 é um membro de Γ, então

1. Pela hipótese indutiva, D_1 é uma refutação para Γ com σ_1 como unificador mais geral. Desde que o conjunto de pares de literais usados nos passos da resolução de D_1 é um subconjunto do conjunto de pares de literais usados para fechar B_1, o unificador σ_0 para o tableau é também um unificador para D_1. Agora, só é preciso refazer D_1 usando o unificador mais geral σ_0 ao invés de σ_1.

2. O número de resolventes é, pela hipótese indutiva, menor ou igual ao número de ramos para se fechar B_1.

3. O conjunto de pares de literais usados nos passos de D_1 é subconjunto do conjunto de pares de literais usados para se fechar B_1, e portanto é um subconjunto dos pares de literais usados para se fechar o tableau inteiro.

Se cada cláusula de entrada de D_2 é um membro de Γ, então a situação é análoga à descrita acima para D_1.

Caso contrário, substituímos D_1 por D_1', onde D_1' é similar a D_1, substituindo a cláusula C_i por $C_i \vee C_k$ e conseqüentemente, todos os resolventes obtidos recursivamente a partir de C_i. Se no final o resolvente for a cláusula vazia, então a refutação está feita e o resultado é verificado da mesma maneira que acima. Caso contrário, o último resolvente deve ser uma instância de C_j. Assim, construímos uma refutação por resolução para Γ concatenando D_2 a D_1' e usando o unificador mais geral σ_0. Note que o número de resolventes é igual à soma dos números de resolventes de D_1 e D_2, o qual é menor ou igual que a soma de ramos necessários para fechar B_1 e B_2, ou seja o número de ramos necessários para fechar o tableau inteiro. ■

10.6 Completude do Sistema

O teorema da completude (teorema 8.8.8) indica que cada uma das maneiras como abordamos a lógica de 1^a ordem são potencialmente equivalentes. Portanto, para mostrar, por exemplo, que uma fbf de 1^a ordem é universalmente válida podemos mostrar que ela é um teorema usando uma prova no estilo axiomático ou uma refutação por resolução da negação da fórmula. Assim, os conceitos de conseqüência lógica, provabilidade e refutabilidade se correspondem, embora eles sejam de natureza diferentes (semântico, sintático e computacional). Neste capítulo, por outro lado, vimos uma série de novas maneiras de apresentar lógica (proposicional e de 1^a ordem). Assim, resulta natural que queríamos que estes estilos de apresentar lógica sejam equivalentes entre se. Mas os resultados que os teoremas 10.1.14, 10.2.5, 10.3.13, 10.4.3 e 10.5.1 nos garantem é mostrar que toda prova no estilo axiomático de um teorema α pode ser transformada num dedução natural para α, e que toda dedução natural de um α pode ser transformada numa dedução no estilo de seqüentes de Gentzen para α, etc. A figura 10.2 mostra como se relacionam os diversos modos de abordar lógica vistos aqui, na figura o sentido da seta indica que é possível transformar uma prova no estilo descrito na origem da seta no estilo descrito no final da seta.

Claramente devido à circularidade dos estilos apresentados neste capítulo com o estilo axiomático, e pelo teorema da completude 8.8.8, podemos estender este mesmo teorema da completude, para um que englobe estes novos estilos.

Teorema 10.6.1 *Para o cálculo de predicado as seguintes sentenças são equivalentes:*

1. $\models \alpha$, *isto é* α *é universalmente válida.*
2. $\vdash \alpha$, *isto é* α *é um teorema de* TP.
3. $\neg\alpha \xrightarrow{*} \square$, *isto é existe uma refutação por resolução de* $\neg\alpha$.
4. *existe uma dedução natural para* α.
5. *existe uma dedução no estilo de seqüentes de Gentzen para* α.
6. *existe uma prova tableau forte de* α.
7. *existe uma prova tableau com unificação de* α.

Demonstração: Direto do teorema da completude 8.8.8 e dos teoremas 10.1.14, 10.2.5, 10.3.13, 10.4.3 e 10.5.1. ■

Figura 10.2: Representação esquemática de como se relacionam os diversos estilos de apresentar lógica de 1^a ordem.

10.7 Exercícios

1. Prove as proposições 4.3.2 e 4.3.3 e os exercícios do capítulo 4 e capítulo 7 no estilo de dedução natural

2. Prove os seguintes teoremas usando dedução natural

   (a) $\neg(\alpha \wedge \beta) \rightarrow (\neg\beta \vee \neg\alpha)$
   (b) $\alpha \vee (\beta \vee \gamma) \rightarrow ((\neg\beta \rightarrow (\alpha \vee \gamma)) \vee \alpha)$
   (c) $\alpha \vee (\beta \vee \gamma) \rightarrow (\beta \vee (\alpha \vee \gamma))$
   (d) $\forall x.(\neg P(x) \vee R(x)) \rightarrow \neg(\exists x.P(x) \wedge \neg\exists x.R(x))$
   (e) $(\forall x.(P(x) \rightarrow (R(x) \vee S(x))) \wedge \neg\forall x.(P(x) \rightarrow R(x))) \rightarrow \exists x.(P(x) \wedge S(x))$

3. Prove de maneira direta que todo teorema de L^P pode ser provado usando cálculo de seqüente de Gentzen.

4. Prove no estilo de Gentzen os seguintes seqüentes:

   (a) $\alpha \rightarrow \beta, \beta \rightarrow \gamma \vdash \alpha \rightarrow \gamma$
   (b) $\neg\alpha \vee (\beta \rightarrow \gamma), \beta \vdash \neg(\alpha \wedge \neg\gamma)$
   (c) $\neg\beta \vee \alpha, \neg\beta \rightarrow \alpha \vdash \alpha$
   (d) $\forall x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)$
   (e) $\forall x.\forall y.P(x,y) \vdash \forall x.\forall y.P(y,x)$
   (f) $\forall x.(P(x) \rightarrow (R(x) \vee S(x))) \wedge \neg\forall x.(P(x) \rightarrow R(x)) \vdash \exists x.(P(x) \wedge S(x))$

5. De uma prova tableau para cada uma das fórmulas bem formadas dos exercícios do capítulo 5.

6. De uma prova tableau para

   (a) $\neg(\alpha \vee \beta) \rightarrow (\neg\alpha \wedge \neg\beta)$
   (b) $\alpha \rightarrow (\beta \rightarrow (\alpha \wedge \beta))$
   (c) cada um dos axiomas de L^P
   (d) $\exists x.(P(x) \rightarrow \forall y.P(y))$
   (e) $\forall x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)$
   (f) $\forall x.\forall y.P(x,y) \vdash \forall x.\forall y.P(y,x)$

Bibliografia

[Acy98] B.M. Acióly. *Programação como Matemática ou Lógica Construtiva*. Anais do I Workshop em Lógica, Banco de Dados e Inteligência Computacional, Natal-RN, 9 e 10 de novembro de 1998. Páginas 13 a 22.

[ABL02] B.M. Acióly; B.R.C. Bedregal; A. Lyra. *Introdução à Teoria das Linguagens Formais, dos Autômatos e da Computabilidade*. Edições UnP, 2002.

[BRA98] J.M. Barreto; M. Roisenberg; M.A.F. Almeida; K. Collazos. *Fundamentos de Matemática Aplicada à Informática*. (Apostila) PPgCC-UFSC, Florianópolis, 1998.

[BSR97] J. Barwise; J. Seligman; C.J. van Rijsbergen (Eds.). *Information Flow: The Logic of Distributed Systems*. Cambridge University Press, 1997.

[BC06] B.R.C. Bedregal; A.P. Cruz. Propositional logic as a propositional fuzzy logic. *Electronic Notes in Theoretical Computer Science*, 143:5–12, 2006.

[Ben03] Mordechai Ben-Ari. *Mathematical Logic for Computer Science* (2nd ed.). Springer, 2003.

[Bet59] E.W. Beth. *The Foundations of Mathematics*. North Holland, 1959.

[BGG98] K. Bhargavan; C.A. Gunter; E.L. Gunter; M. Jackson; D. Obradovic; P. Zave. The Village Telephone System: A Case Study in Formal Software Engineering. *Lecture Notes In Computer Science*; Vol.1479:49 – 66, 1998. Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics.

[BBC97] L. Blair; H. Bowman; A. Chetwynd; G. Blair (Eds.). *Formal Specifications of Distributed Multimedia Systems*. CRC Press, 1997.

[BB95] G. Bojadziev; M. Bojadziev. *Fuzzy Sets, Fuzzy Logic, Applications*. World Scientific Publishing, Singapore, 1995.

[Bos97] D. Bostock. *Intermediate Logic*. Clarendon Press - Oxford, 1997.

[Bow82] K.A. Bowen. *Programming with full first order logic*. In: J.E. Hayes et al. (eds.), *Machine Intelligence 10*. Elis Horwood, Chichester, 1982.

[BLC00] A.P. Braga; T.B. Ludermir; A.C.P.L.F. Carvalho. *Redes Neurais Artificiais: Teoria e Aplicações*. LTC, 2000.

[Bun83] A. Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, London, 1983.

[CE06] W. Carnielli; R.L. Epstein. *Computabilidade, Funções Computáveis, Lógica e os Fundamentos da Matemática*. Editora UNESP, 2006.

[CGF87] M.A. Casanova; F.A.C. Giorno; A.L. Furtado. *Programação em Lógica e a Linguagem Prolog*. Editora Edgard Blücher Ltda., 1987.

CL73 CL73 C.H. Chang; R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[Cos92] M.M.C. Costa. *Introdução à Lógica Modal Aplicada à Computação*. VIII Escola de Computação, 3 a 12 de agosto de 1992, Gramado/RS.

[CHRP73] A. Colmerauer; H. Hanoui; P. Roussel; R. Pasero. *Un Systeme de Communication Homme-Machine en Francais*. Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1973.

[Cox05] E. Cox. *Fuzzy Modelling and Genetic Algorithms for Data Mining and Exploration*. Elsevier, 2005.

[CS94] M. Craven; J. Shavlik. Using sampling and queries to extract rules from trained neural networks. In: *Proceedings of the 11th International Conference on Machine Learning*, pages 37-45, New Brunswick, 1994.

[CM04] A.M.P. Cruz; J.E.A. Moura. *A Lógica na Construção dos Argumentos*. Notas em Matemática Aplicada 14, SBMAC, São Carlos, 2004.

[DOS03] M.C. Daconta; L.J. Obrst; K.T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Wiley, 2003.

[Dal80] D. van Dalen. *Logic and Structures* (2^{nd} edition). Berlin: Springer-Varlag, 1980.

[Dav89] R.E. Davis. *Truth, Deduction, and Computation: Logic and semantic for computer science*. Computer Science Press, 1989.

[DS90] E.W. Dijkstra; C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer verlag, 1990.

[Doe94] K. Doets.*From Logic to Logic Programming*. The MIT Press, Cambridge, Massachusetts, 1994.

[Dow98] M. Downward. *Logic And Declarative Language*. Taylor & Francis, 1998.

[Dum77] M. Dummett. *Elements of Intuitionism*. Oxford U.P., 1977.

[EN99] R. Elmasri; S. Navathe. *Fundamentals of database systems* (3rd edition). Addison-Wesley Publishing Company, 1999.

[End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press Inc., New York, 1972.

[Fit96] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Second Edition, Springer-Verlag, 1996.

[Fir88] M.W. Firebaugh. *Artificial Intelligence: A Knowledge Based Approach*. Boyd & Fraser Publishing Company, 1988.

[Fre02] A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer, 2002.

[Gal86] J.H. Gallier. *Logic for Computer Science*. Harper and Row, New York, 1986.

[Gir87] J.Y. Girard.Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

[G+89] P. Gochet et al. *From Standard Logic to Logic Programming: Introducing a Logic Based Approach to Artificial Intelligence*. André Thayse ed.. John Wiley & sons, 1989.

[GH75] H.B. Griffiths; P.J. Hilton. *Matemática Clássica: Uma interpretaçã contemporânea*. Traduzida por Elza F. Gomide. São Paulo, Edgars Blücher, Ed. da Universidade de São Paulo, 1975.

[Gog97] J.A. Goguen. *Theorem Proving and Algebra* (Draft). University of California at San Diego, Department of Computer Science and Engineering, 1997.

[GLT89] J.Y. Girard; Y. Lafont; P. Taylor. *Proof and Types*. Cambridge Tracts in Theoretical Computer Science, Vol. 7. Cambridge University Press, 1989.

[GTW77] J.A. Goguen; J.W. Thatcher; E.G. Wagner. *An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Type*. RC 6487, IBM Thomas J. Watson Research Center, New York, 1976.

[GW87] R. Gonzales; P. Wintz. *Digital Image Processing* (2nd ed.). Addison-Wesley Publishing, 1987.

[Har79] D. Harel. *First Order Dynamic Logic*. Lecture Notes in Computer Science, LNCS, Vol. 68, 1979.

[HPW96] J. Harland; D. Pym; M. Winikoff. *Programming in Lygon: An overview*. Proceeding of the Fifth International Conference on Algebraic Methodology and Software Technology, 391–405, Munich, july, 1996.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. The MIT Press, 1988.

[Hey56] A. Heyting. *Intuitionism: An Introduction*. North Holland, 1956.

[HA50] D. Hilbert; W. Ackermann. *Principles of Mathematical Logic*. Chelsea, 1950.

[Hin55] J.Hintikka. *Form and Content in Quantification Theory*. Acta Philosophica Fennica, 8, 1955.

[HL94] P. Hill; J. Lloyd. *Godel Programming Languages (Logic Programming)*. MIT Press, may 1994.

[Hod92] J.S. Hodas. *Lolli: An Extension of λProlog with Linear Logic Context Management*. Proceeding of the 1992 Workshop on the λProlog Programming Languages. Philadelphia, summer 1992.

[Hod95] J.S. Hodas. *Lolli: A Linear Logic Programming Language*. In http://www.cs.hmc.edu/ hodas/research/lolli/, última atualização em 26/07/1995. Acessado em 17/01/2002.

[HMM83] J. Halpern; Z. Manna; B. Moszwkowski. A Hardware Semantics Based Upon Temporal Intervals. In: *Procedding of* 19^{th} *International Colloquium on Automata, Language and Programming*. Lecture Notes in Computer Science, LNCS, Vol. 154:278-292, 1983.

[Hod83] A. Hodges. *Alan Turing: The Enigma*. Simon and Schuster, New York, 1983.

[HR99] M. Huth; M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 1999.

[Kas96] N. K. Kasabov. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. Ed. The MIT Press, Massachusetts Institute of Technology, 1996.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

[Kow72] R. Kowalski. *The Predicate Calculus as a Programming Languages*. Anais do International Symposium and Summer School on Mathematical Foundations of Computer Science, Jablona near Warsaw, Poland, 1972.

[Kow74] R. Kowalski. *Predicate Logic as a programming language*. Proceedings IFIP'74, North-Holland 1974, páginas, 569-574.

[Lac05] L.W. Lacy. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, 2005.

[Lan64] P.J. Landim. *The Mechanical Evaluation of Expressions*. Computer Journal, 6:308-320,1964.

[Mak92] J.A. Makowsky. *Model Theory and Computer Science: An Appetizer*. In Handbook of Logic in Computer Science Vol. 1. S. Abramsky, Dov. M. Gabbay and T.S.E. Maimbaum (eds). Clarendon Press - Oxford, 1992.

[Man99] V. Manca. String Rewriting and Metabolism: A logical perpesctive. In *Computing with Bio-Molecules: Theory and Experiments*, Gheorghe Paun (ed). Springer, 1999.

[Mar90] A. Margaris. *First Order Mathematical Logic*. New York: Dover Publications, Inc., 1990.

[MM88] R.C.B. Martins; A.V. Moura. *Desenvolvimento Sistemâtico de Programas Corretos: A abordagem denotacional*. VI Escola de Computação, Campinas-SP, 1988.

[Men87] E. Mendelson. *Introduction to Mathematical Logic*. Third Edition. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California. A division of Wadsworth, Inc. 1987.

[Mer53] C.A. Meredith. *Single axioms for the systems (C,N), (C,O) and (A,N) of the two valued propositional calculus*. IN J. Comp. Syst. (3):155-164, 1953.

[Mer93] J.P. Mermet (ed.). *Fundamentals and Standards Hardware Description Languages*. Kluwer Academic Publisher, 1993.

[MP79] Z. Manna; A. Pneuli. The Modal Logic of Programs. In: *Proceeding of* 6^{th} *International Colloquium on Automata, Language and Programming*. Lecture Notes in Computer Science, LNCS, Vol. 71:385-411, Springer-Verlag, 1979.

[Min00] J. Minker. *Logic-Based Artificial Intelligence*. Springer, 2000.

[Moo95] R. Moore. *Logic and Representation*. CLSI Lecture Notes N^o 39, CLSI Publications, Stanford, 1995.

[Muh04] H. Mühlenbein. Toward a theory of organism and evolving automata. In: *Frontiers of Evolutionary Computation*, Anil Menon (ed.). Springer, 2004.

[Ngu99] H.T. Nguyen; E.A Walker. *A First Course in Fuzzy Logic.* Chapman and Hall, 1999.

[Nie02] Y. Nievergett. *Foundations of Logic and Mathematics: Applications to Computer Science and Cryptography*. Birkhäuser, 2002.

[NP01] T. Nipkow; L.C. Paulson. *Isabelle HOL: Then Tutorial*. Springer-Verlag, 2001.

[OUS01] OUSIA Site. *Estudos em Aristóteles*. In http://www.ifcs.ufrj.br/f̃santoro/ousia/frames.htm. Accesado em 27/11/2001.

[Pal97] L.A.M. Palazzo. *Introdução à Programação PROLOG*. Pelotas: Editora da Universidade Católica de Pelotas, EDUCAT, 1997.

[Pau94] L.C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828, Springer-Verlag, 1994.

[PPW79] L.M. Pereira; F.C.N. Pereira; D.H.D. Warren. *User's Guide to DECsystem-10 Prolog (Provisional Version)*, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1979.

[PM95] L. Pólos; M. Masuch (eds). *Applied Logic: How, What and Why: Logical Approaches to Natural Language*. Springer, 1995.

[Ree85] S.V. Reeves. *Theorem-Proving by Semantic Tableaux*. PhD. Thesis, University of Birmingham, 1985.

[Ric88] E. Rich. *Inteligência Artificial*. McGraw-Hill, 1988.

[RK94] E. Rich; K. Knight. *Inteligência Artificial*, 2^a ed. Makron Books, 1994.

[Rob65] J.A. Robinson. A Machine-Oriented Based on the Resolution Principie. *Journal of ACM*, New York, V.12, n.1., p.23-41, Jan. 1965.

[Rob79] J.A. Robinson. *Logic: Form and Function*. North Holland, New York, 1979.

[Ros04] T.J. Ross. *Fuzzy Logics: With Engineering applications* (2nd Ed.), John Wiley & Sons, Inc., 2004.

[Ros53] J.B. Rosser. *Logic for Mathematicians*. McGraw Hill, 1953 (second edition, 1978, Chelsea).

[RN02] S.J. Russell; P. Norvig. *Artificial Intelligence: A Modern Approach* (2nd Ed.). Prentice Hall, 2002.

[RS92] M. Ryan; M. Sadler. Valuation Systems and Consequence Relations. In: *Handbook of Logic in Computer Science*, Vol. 1. S. Abramsky, Dov. M. Gabbay and T.S.E. Maimbaum (eds). Clarendon Press - Oxford, 1992.

[SB04] R.H.N. Santiago; B.R.C. Bedregal. *Computabilidade: Os limites da computação*. Notas em Matemática Aplicada, Vol. 11, SBMAC, São Carlos, 2004.

[Sch95] S. Schulze-Kremer. *Molecular Bioinformatics: Algorithms and Applications*. Walter de Gruyter, 1995.

[SB05] W. Siler; J.J. Buckley. *Fuzzy Expert Systems and Fuzzy Reasoning*. John Wiley & Sons, Inc., 2005.

[Smu68] R.M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlim, 1968.

[SHH02] H. Stuckenschmidt; J. Hartmann; F. van Harmelen. *Learning Structural Classification Rules for Web-page Categorization*. Proceedings of the Fifteenth International FLAIRS conference, may 2002.

[TB01] N. Tamura; M. Bonbara. *LLP: A Linear Logic Programming Languages and its Compiler System*. http://bach.cs.kobe-u.ac.jp/llp/, última atualização em 16/10/2001. Acessado em 17/01/2002.

[TM96] D.E. Thomas; P.R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publisher, 1996.

[Tho87] S. Thompson. *An Introduction to Type Theory and Constructive Mathematics* (Draft). University of Kent at Canterbury, Computing Laboratory, 1987.

[Thu98] B. Thuraisingham. *Data Mining*. CRC Press, 1998.

[Woj99] R. Wojtecki. *Air Logic Control for Automated Systems*. CRC, 1999.

[xre01] xrefer site. *Logic, history of*. In http://www.xrefer.com/entry/552643. Accesado em 06/11/2001.

[Zad65] L. Zadeh. *Fuzzy Sets*. Information Control 8, (1965).

[Zem73] J.J. Zeman. *Modal Logic*. Oxford, 1973.

Índice