

Sobre os Fundamentos da Programação Lógica Paraconsistente

Tarcísio Genaro Rodrigues

Dissertação apresentada ao
Departamento de Filosofia do Instituto
de Filosofia e Ciências Humanas da
Universidade Estadual de Campinas
para obtenção do grau de Mestre em
Filosofia (Linha de Pesquisa).

Orientador: **Prof. Dr. Marcelo Esteban Coniglio**

*Durante a elaboração deste trabalho
o autor recebeu apoio financeiro da
Fapesp, processo 2008/07760-2.*

Setembro de 2010

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IFCH - UNICAMP
Bibliotecária: Sandra Aparecida Pereira CRB nº 7432**

Rodrigues, Tarcísio Genaro
R618s Sobre os Fundamentos da Programação Lógica Paraconsistente /
Tarcísio Genaro Rodrigues. - - Campinas, SP : [s. n.], 2010.

Orientador: Marcelo Esteban Coniglio.
Dissertação (mestrado) - Universidade Estadual de Campinas,
Instituto de Filosofia e Ciências Humanas.

1. Lógica matemática não clássica. 2. Lógica simbólica e matemática. 3.
Inconsistência (Lógica). 4. Programação Lógica. 5. Linguagens formais
- Semântica. I. Coniglio, Marcelo Esteban. II. Universidade Estadual de
Campinas, Instituto de Filosofia e Ciências Humanas. III. Título.

Título em inglês: On the foundations of paraconsistent logic programming

Palavras chaves em inglês (keywords): Non-classical mathematical logic
Symbolic logic and mathematics
Inconsistency (Logic)
Logic Programming
Formal languages – Semantics

Área de concentração: Filosofia

Titulação: Mestre em Filosofia

Banca examinadora: Marcelo Esteban Coniglio, Juliana Bueno Soler,
Hércules de Araújo Feitosa, Walter Alexandre Car-
nielli, Hugo Luiz Mariano

Data da defesa: 29-09-2010

Programa de Pós-Graduação: Filosofia

*À minha família e principalmente
à minha Mãe, Edna*

Agradecimentos

Gostaria de agradecer aqui aos Professores do Centro de Lógica e em especial ao meu orientador, Professor Marcelo Coniglio. Todos eles foram responsáveis pelo que sei hoje em Lógica, mas especialmente o Marcelo, por tantas disciplinas cursadas sob sua orientação e horas de trabalho conjunto que possibilitaram que essa dissertação chegasse a um fim. Quero agradecer também aos outros membros da banca, Professores Hércules Feitosa, Juliana Soler e Walter Carnielli pela paciência, comentários e preciosas correções. Ao professor Paolo Gentilini, por nos enviar um trabalho seu que foi de fundamental importância para que pudéssemos obter nossos principais resultados. Aos colegas de estudo, em especial ao Anderson de Araújo por algumas conversas muito interessantes sobre nosso tema e por chamar nossa atenção ao trabalho do Professor Gentilini. Agradeço também aos meus familiares mais próximos: Ada, Gisela, Tarcísio e Edna.

Finalmente, agradeço à Fundação de Amparo à Pesquisa do Estado de São Paulo (Fapesp), pelo suporte financeiro.

Resumo

A Programação Lógica nasce da interação entre a Lógica e os fundamentos da Ciência da Computação: teorias de primeira ordem podem ser interpretadas como programas de computador. A Programação Lógica tem sido extensamente utilizada em ramos da Inteligência Artificial tais como Representação do Conhecimento e Raciocínio de Senso Comum. Esta aproximação deu origem a uma extensa pesquisa com a intenção de definir sistemas de Programação Lógica paraconsistentes, isto é, sistemas nos quais seja possível manipular informação contraditória. Porém, todas as abordagens existentes carecem de uma fundamentação lógica claramente definida, como a encontrada na programação lógica clássica. A questão básica é saber quais são as lógicas paraconsistentes subjacentes a estas abordagens.

A presente dissertação tem como objetivo estabelecer uma fundamentação lógica e conceitual clara e sólida para o desenvolvimento de sistemas bem fundados de Programação Lógica Paraconsistente. Nesse sentido, este trabalho pode ser considerado como a primeira (e bem sucedida) etapa de um ambicioso programa de pesquisa. Uma das teses principais da presente dissertação é que as Lógicas da Inconsistência Formal (**LFI**'s), que abrangem uma enorme família de lógicas paraconsistentes, proporcionam tal base lógica.

Como primeiro passo rumo à definição de uma programação lógica genuinamente paraconsistente, demonstramos nesta dissertação uma versão simplificada do Teorema de Herbrand para uma **LFI** de primeira ordem. Tal teorema garante a existência, em princípio, de métodos de dedução automática para as lógicas (quantificadas) em que o teorema vale. Um pré-requisito fundamental para a definição da programação lógica é justamente a existência de métodos de dedução automática. Adicionalmente, para a demonstração do Teorema de Herbrand, são formuladas aqui duas **LFI**'s quantificadas através de sequentes, e para uma delas demonstramos o teorema da eliminação do corte. Apresentamos também, como requisito indispensável para os resultados acima mencionados, uma nova prova de correção e completude para **LFI**'s quantificadas na qual mostramos a necessidade de exigir o Lema da Substituição para a sua semântica.

Abstract

Logic Programming arises from the interaction between Logic and the Foundations of Computer Science: first-order theories can be seen as computer programs. Logic Programming have been broadly used in some branches of Artificial Intelligence such as Knowledge Representation and Commonsense Reasoning. From this, a wide research activity has been developed in order to define paraconsistent Logic Programming systems, that is, systems in which it is possible to deal with contradictory information. However, no such existing approaches has a clear logical basis. The basic question is to know what are the paraconsistent logics underlying such approaches.

The present dissertation aims to establish a clear and solid conceptual and logical basis for developing well-founded systems of Paraconsistent Logic Programming. In that sense, this text can be considered as the first (and successful) stage of an ambitious research programme. One of the main thesis of the present dissertation is that the Logics of Formal Inconsistency (**LFI**'s), which encompasses a broad family of paraconsistent logics, provide such a logical basis.

As a first step towards the definition of genuine paraconsistent logic programming we shown, in this dissertation, a simplified version of the Herbrand Theorem for a first-order **LFI**. Such theorem guarantees the existence, in principle, of automated deduction methods for the (quantified) logics in which the theorem holds, a fundamental prerequisite for the definition of logic programming over such logics. Additionally, in order to prove the Herbrand Theorem we introduce sequent calculi for two quantified **LFI**'s, and cut-elimination is proved for one of the systems. We also present, as an indispensable requisite for the above mentioned results, a new proof of soundness and completeness for first-order **LFI**'s in which we show the necessity of requiring the Substitution Lemma for the respective semantics.

Sumário

1	Introdução: Da Lógica à Programação Lógica	1
1.1	A Lógica no Século XX	2
1.2	Desenvolvimento da Programação Lógica	9
1.2.1	A Origem da Dedução Automática	9
1.2.2	Resolução e Cláusulas de Horn	11
1.2.3	O Paradigma Declarativo: Algoritmo = Lógica + Controle	15
1.2.4	Cálculo com Negações	16
2	Aspectos Formais da Programação Lógica Clássica	19
2.1	Sintaxe	21
2.1.1	Linguagens de Primeira Ordem	21
2.1.2	Programas Lógicos	22
2.1.3	Substituições e Unificadores mais Gerais	26
2.1.4	Resolução SLD	30
2.2	Semântica	33
2.2.1	Semântica para Lógica de Primeira Ordem	33
2.2.2	Corretude da Resolução SLD	35
2.2.3	Modelos de Herbrand	35
2.2.4	Operador de Consequência Imediata e Pontos Fixos	38
2.3	Informação Negativa e Raciocínio não-Monotônico	44
3	Rumo à Programação Lógica Paraconsistente I: Bases Lógicas	47
3.1	Programação Lógica Paraconsistente	48
3.2	A Necessidade de um Teorema de Herbrand	54
3.3	Lógicas da Inconsistência Formal	54
3.3.1	As LFI 's básicas: mbC e mCi	56
3.3.2	LFI 's quantificadas: os sistemas QmbC e QmCi	62
3.3.3	Semântica de Estruturas Paraconsistentes	74
3.3.4	Correção e completude das lógicas QmbC e QmCi	83

4	Rumo à Programação Lógica Paraconsistente II: Um Teorema de Herbrand	103
4.1	Sequentes para QmbC e QmCi	104
4.1.1	Definição	105
4.1.2	Completude dos Sequentes em relação à Formulação Hilbertiana	109
4.1.3	Correção em Relação às Valorações Paraconsistentes	115
4.1.4	Eliminação do Corte para QmbC	116
4.2	Teorema de Herbrand para QmbC	125
5	Próximos Passos	129
5.1	Problemas com a Dualidade entre Satisfatibilidade e Validade: Considerações Semânticas	130
5.2	Um Método de Resolução Restrita	131
5.3	Conclusões	131

Capítulo 1

Introdução: Da Lógica à Programação Lógica

Neste primeiro capítulo serão desenvolvidos aspectos históricos e metodológicos da programação lógica. Dessa maneira, pretende-se contextualizar o presente projeto no panorama mais geral das pesquisas em Lógica e Inteligência Artificial, que tiveram um grande avanço durante todo o Século XX. Para tanto, busca-se evidenciar como o caráter universal com que se apresenta a lógica matemática e, em particular, o cálculo de primeira ordem clássico, determinaram o desenvolvimento dos primeiros métodos automáticos de demonstração na década de 1950 e de alguns ramos da Inteligência Artificial nas décadas de 1960 e 1970. Essas duas áreas são o solo do qual se ergue a pesquisa em programação lógica. A programação lógica é um belo exemplo de interação entre lógica e tecnologia e não pode ser entendida em sua totalidade sem que se entendam esses seus dois lados.

A linha de raciocínio seguirá, nesta introdução, em torno da seguinte progressão de relações:

- Completude e teorema de Herbrand
- Teorema de Herbrand e resolução
- Resolução e programação lógica
- Programação lógica e PROLOG.

1.1 A Lógica no Século XX

A lógica matemática tem um papel universal em pelo menos dois sentidos: por um lado, possibilita de maneira homogênea a descrição sintática de um tipo de linguagem comum a diversas áreas do conhecimento, bem como o entendimento de fundo necessário a certos conceitos gerais que garantem expressividade a diversas disciplinas: as noções de predicado e função de primeira ordem e a possibilidade de compor expressões com conectivos e quantificadores têm uma parte essencial em várias áreas do conhecimento. Isso pode ser entendido como o **caráter descritivo da lógica**. Por outro, através da relação de consequência lógica, podem-se expressar as consequências das diversas teorias quando elas se encontram devidamente expressas em uma linguagem apropriada: tal é o **caráter dedutivo da lógica**. Para tanto, não importa que as teorias em questão sejam do tipo formal, como por exemplo as diversas teorias algébricas, *com várias interpretações possíveis*, ou interpretadas, quando o que se busca é *caracterizar um único modelo*, como a mecânica newtoniana. A distinção entre função descritiva e dedutiva é trabalhada por Hintikka em [Hin96], onde afirma o caráter descritivo como fundamental, anterior mesmo ao dedutivo.¹ Tais características são importantes para se entender o desenvolvimento da própria programação lógica. Elas ajudam a entender porque a lógica matemática está nos fundamentos de duas áreas que poderiam ser tomadas como suas mães: o campo da *dedução automática* e o da *Inteligência Artificial*. Mas, para chegar ao ponto em que se encontra, com tal grandeza expressiva e uma aceitação que extrapola a área da matemática e da filosofia, a lógica clássica passou por um grande desenvolvimento.

Kant no século XVIII, confundido a Lógica com a dedução silogística, a considerava uma disciplina acabada e perfeita, dado que não havia sofrido modificações significativas desde Aristóteles. Mas as coisas mudaram de figura em meados do Século XIX. Os trabalhos de Boole [Boo47] e De Morgan [DM47] marcam o começo da abordagem matemática em lógica clássica. Boole desenvolveu o caráter algébrico do silogismo aristotélico, à maneira do que Descartes fez com a geometria grega. Inicia-se, então, uma tradição – cultivada por Pearce, Schröder e outros – conhecida como *álgebra da lógica*, que lançou as bases da abordagem posteriormente desenvolvida pela *teoria dos modelos*.

Ainda no Século XIX, através das discussões fundacionais que perpassaram a matemática, evoluiu como pano de fundo à visão *logicista* dos fundamentos. Estava presente em meio a teorias que buscavam bases sólidas para determinadas áreas, como na *Begriffsschrift* de Frege, que elaborou pela primeira vez um *cálculo conceitual*, com linguagem e regras de dedução formalmente especificadas, a ser

¹[Hin96, p.9]

usada para os *Grundgesetze der Arithmetik*². No trabalho de Peano, apesar de não contar com um sistema de *regras*, fazia-se presente enquanto sistema notacional. Nessa perspectiva fundacional, a lógica teve a missão de eliminar paradoxos que surgiam na teoria dos conjuntos (a primeira grande teoria que se propunha a abarcar a matemática como um todo), com a teoria Russelliana de tipos. Desse modo, constituiu-se a outra linha tradicional de investigação em lógica matemática, de carácter preponderantemente sintático.³

Mas foi com o programa de Hilbert para a fundamentação da matemática que certos conceitos essenciais foram pela primeira vez propostos. Em Frege ou Whitehead e Russell, por exemplo, o foco estava na lógica de ordem superior, não se destacando o fragmento de primeira ordem à parte. Tudo indica que foi Hilbert o primeiro a reconhecer que sistemas de prova de primeira ordem eram dignos de estudo⁴. Pelo começo do ano de 1920, em seus seminários sobre os fundamentos da matemática, Hilbert foi levado a propor o que considerava *o principal problema da lógica matemática*: o problema da decisão⁵ para a lógica de primeira ordem⁶. Em seus *Princípios da Lógica Teórica* [HA28], Hilbert e Ackermann o definem precisamente: encontrar um processo “que permita decidir a validade de expressões lógicas em um número finito de operações”⁷.

Martin Davis, grande contribuidor para a historiografia da dedução automática, além de co-autor de um método pioneiro de demonstração automática⁸, afirma que em [HA28] “algumas das idéias centrais no trabalho em dedução automática apareceram pela primeira vez”⁹. Entre elas, além do problema da decisão, há a questão da completude do cálculo funcional clássico. Essas questões estão relacionadas com o cerne do programa de Hilbert: encontrar demonstrações de consistência de teorias matemáticas formalizadas, fazendo uso exclusivo de métodos finitários de argumentação.

Alguns anos mais tarde, com os teoremas de incompletude de Gödel, ficou claro que, para sistemas com uma certa expressividade, tais demonstrações de consistência não seriam realizáveis internamente aos próprios sistemas. Isso acabava com as perspectivas de algum sistema fraco da aritmética (ou seja, o que se entende formalmente por *finitário*) provar sua própria consistência. Mas o programa de Hilbert não deixou apenas resultados negativos, foi em seu contexto que surgiu

²Os Fundamentos da Aritmética, obra fundamental de Frege, com a qual pretendia reduzir a aritmética às suas bases lógicas.

³[DvH86, p.44]

⁴[Bur98, p.382]

⁵*Entscheidungsproblem*, no original em alemão, forma pela qual é normalmente referenciado.

⁶[Soa96, p.6]

⁷[HA28, pp.72-73], conforme citado em [Soa96, *ibidem*].

⁸O procedimento de Davis-Putnam [DP60].

⁹[Dav83, p.11]

a importante noção de *metamatemática*: a investigação matemática de sistemas lógicos formalizados. Esse contexto deu sentido à questão da completude e, mais tarde, possibilitou o estudo sistemático de diversos sistemas de prova e suas inter-relações. Desse modo, o desenvolvimento dos primeiros provadores automáticos, na década de 1950, pode também entrar na conta dos “herdeiros” dessas primeiras investigações metamatemáticas.

Do ponto de vista dos fundamentos da matemática, a completude é um resultado notório por si. O próprio fato de que tal problema tenha passado a fazer sentido foi significativo no desenvolvimento da lógica como um todo, conforme a análise de van Heijenoort e Dreben em [DvH86, §1]. Na linha do *logicismo*, que se estendeu de Frege até Russell e Whitehead, tal questão não poderia ter surgido tendo em vista o caráter universal que imprimiam à lógica: a ela cabia a formalização de toda a matemática. Sistemas quantificacionais *puros* não estavam no centro das investigações. O caráter absoluto da lógica os impedia de considerar seus sistemas como totalidades passíveis de estudo, a não ser pela derivação (interna) de teoremas. Já do ponto de vista da linha da *álgebra da lógica*, faz falta a própria noção de *sistema formal*, tudo se dá por um viés modelo-teórico semântico.

A completude, demonstrada por Gödel em [Gö30], é um resultado metamatemático que relaciona o logicismo com a abordagem da álgebra da lógica. O teorema da completude mostrou por meios *não construtivos* que as axiomatizações (o sistema em que trabalhava era essencialmente o fragmento de primeira ordem dos *Principia*, seguindo a notação de Hilbert e Ackermann [HA28]) “detectavam” sintaticamente todas as teorias que não tinham modelos. A prova (na verdade, sua generalização para teorias enumeráveis pelo teorema da compacidade) consistia em mostrar que todo sistema axiomático de primeira-ordem ou é inconsistente (entendido como *derivando uma contradição*) ou tem um modelo. Ou seja, as teorias sintaticamente inconsistentes (*que derivam contradições*) são exatamente aquelas semanticamente inconsistentes (*que não têm modelos*). Com isso, na própria opinião de Gödel, obteve-se a fundamentação de um método usual na época – o de se provar que certas teorias não deduzem contradições, através de demonstrações da existência de modelos.¹⁰

Tendo em vista que pode haver cálculos definidos sobre linguagens não recursivamente enumeráveis, uma demonstração de caráter não construtivo é essencial. O argumento original de Gödel, apesar de estar baseado em argumentos não construtivos, não se estende a teorias definidas sobre linguagens não-enumeráveis e se baseia na existência de *domínios numéricos* para teorias não contraditórias. A formulação mais usada atualmente, baseada na demonstração de Henkin [Hen49], admite teorias sobre linguagens arbitrárias e faz uso de *modelos sintáticos*, obti-

¹⁰[DvH86, p.48]

dos da própria linguagem da teoria, enriquecida com constantes testemunhas para fórmulas existenciais. Para se chegar a tais modelos também é necessário algum *princípio de escolha* (como o teorema de Teichmüller-Tukey em [Sho67, p.47]) que demonstre a existência de extensões completas para teorias consistentes.

Relacionado a esse resultado positivo para o problema da completude, há outra negativa aos problemas levantados em [HA28]. A completude de Gödel pode ser entendida como uma solução não construtiva para o problema da decisão. Por um lado, reduz a validade lógica à derivação sintática por meio de axiomas e regras de inferência (o que, a primeira vista, parece indicar a possibilidade de uma solução finitária). Mas, por outro, demanda algum equivalente do axioma da escolha para a demonstração da existência de contra-modelos para teorias contraditórias. A descoberta de alguma solução completamente finitária para o problema da decisão era uma das motivações da lógica formal na década de 1920 e 1930. Isso veio a se mostrar impossível com os trabalhos de Church [Chu36] e Turing [Tur37], em que demonstram por absurdo a inexistência de um tal procedimento.

Apesar de não construtiva, a completude de Gödel demonstra que o conjunto das sentenças logicamente válidas é recursivamente enumerável. Restringindo nossa atenção às bases da dedução automática, a demonstração de Gödel estava construída sobre o trabalho de Skolem, o qual, segundo Davis, foi fundamental para essa área:

The Key work for automated deduction was that of Skolem. He carried out a systematic study of the problem of the existence for an interpretation which will satisfy a given formula of the predicate calculus.

[Dav83, p.9]

Para sua demonstração da completude, Gödel adaptou e generalizou um trabalho de Skolem publicado em 1920 [Sko20]. Skolem escrevera esse artigo para clarificar o teorema de Löwenheim, publicado em [Lö15]. Tal teorema foi o primeiro resultado significativo para a lógica de primeira ordem¹¹: afirma que, se uma fórmula é satisfatível, também o é em um domínio enumerável. Em [Sko20], Skolem generaliza este resultado para um conjunto enumerável de fórmulas.

Para chegar ao resultado, Skolem prova antes que para toda fórmula de primeira ordem existe uma outra, em uma forma normal, que é satisfatível em algum domínio exatamente quando a original também o é. Essas formas normais estão no que designa-se, hoje em dia, por fragmento Π_2 , isto é, são fórmulas constituídas por uma sequência de quantificadores universais seguidos por uma sequência

¹¹Conforme afirma Burris em [Bur98, p.365]. Há quem diga mesmo que “Com esse artigo, a lógica de primeira ordem se tornou uma área de interesse especial, devido às surpreendentes propriedades metamatemáticas (...)”, [WSBA09, p.17].

de existenciais e, ao final, uma fórmula livre de quantificadores. Para chegar às interpretações numéricas para teorias não contraditórias, Gödel fez uso da *forma normal de Skolem* e pôde restringir sua atenção apenas às fórmulas nesse formato (depois de mostrar que seu resultado mais geral era obtido a partir do resultado para as formas normais). O que Gödel demonstrou, de fato, foi que toda fórmula ou tem um modelo enumerável, ou é *refutável*, no sentido de sua negação ser demonstrável.

Em 1928 [Sko28], Skolem introduziu a noção de *Skolemização* de uma fórmula, reduzindo o problema da satisfatibilidade ao fragmento das fórmulas com apenas quantificadores universais. Toda fórmula é satisfatível em um domínio se, e somente se, sua versão Skolemizada também o é. Sobre esse artigo de 1928, Davis afirma:

This remarkable paper, not only has a treatment of what is usually called Herbrand's theorem in writings on automated deduction, but has a clear and complete definition of what is usually called in this field the Herbrand universe for a formula.

[Dav83, p.10]

A Skolemização é feita em dois passos. Primeiramente, os quantificadores são postos no início, com renomeações de variáveis se necessário. Por exemplo, considere a seguinte fórmula:

$$\exists x (\forall y P(y) \wedge Q(x)) \rightarrow \forall x (Q(x) \wedge \exists x R(x)) .$$

Para encontrar sua forma skolemizada, renomeam-se as variáveis quantificadas de tal modo que cada quantificador tenha uma variável diferente:

$$\exists x (\forall y P(y) \wedge Q(x)) \rightarrow \forall z (Q(z) \wedge \exists w R(w)) .$$

Se a subfórmula $\forall y P(y) \wedge Q(x)$ é denotada por $\psi(x)$, e $Q(z) \wedge \exists w R(w)$ por $\gamma(z)$, é possível ver que a fórmula acima tem o seguinte formato:

$$\exists x \psi(x) \rightarrow \forall z \gamma(z) .$$

Deve-se, agora, escolher um dos dois quantificadores $\exists x$ e $\forall z$ para colocá-lo no início, de forma a obter $\exists x (\psi \rightarrow \forall z \gamma)$ ou $\forall z (\exists x \psi \rightarrow \gamma)$. Primeiramente $\exists x$:

$$\exists x \left((\forall y P(y) \wedge Q(x)) \rightarrow \forall z (Q(z) \wedge \exists w R(w)) \right)$$

e então $\forall z$:

$$\exists x \forall z \left((\forall y P(y) \wedge Q(x)) \rightarrow (Q(z) \wedge \exists w R(w)) \right) .$$

Procede-se igualmente sobre a subfórmula $\forall y P(y) \wedge Q(x)$ com a substituição por sua versão com o quantificador no início, $\forall y (P(y) \wedge Q(x))$:

$$\exists x \forall z (\forall y (P(y) \wedge Q(x)) \rightarrow (Q(z) \wedge \exists w R(w))) .$$

Agora é a vez de $Q(z) \wedge \exists w R(w)$, que será trocada por $\exists w (Q(z) \wedge R(w))$:

$$\exists x \forall z (\forall y (P(y) \wedge Q(x)) \rightarrow \exists w (Q(z) \wedge R(w))) .$$

Como antes, há duas possibilidades. Escolhe-se primeiramente o quantificador $\exists w$ e na sequência $\forall y$:

$$\begin{aligned} &\exists x \forall z \exists w (\forall y (P(y) \wedge Q(x)) \rightarrow (Q(z) \wedge R(w))) \\ &\exists x \forall z \exists w \forall y ((P(y) \wedge Q(x)) \rightarrow (Q(z) \wedge R(w))) . \end{aligned}$$

Agora, deve-se eliminar os quantificadores existenciais pela adição de símbolos funcionais. Com a retirada de um existencial, adiciona-se um símbolo funcional de aridade igual ao número de quantificadores universais dos quais o existencial está no escopo. A variável x , ligada por um existencial, será substituída ao longo da fórmula pelo novo símbolo de constante f_x , tendo em vista que seu quantificador não está no escopo de nenhum universal:

$$\begin{array}{c} \exists x \forall z \exists w \forall y (P(y) \wedge Q(x) \rightarrow Q(z) \wedge R(w)) \\ \hline \forall z \exists w \forall y (P(y) \wedge Q(f_x) \rightarrow Q(z) \wedge R(w)) \end{array}$$

E a variável w será trocada por $g_w(z)$, tendo em vista que seu quantificador está no escopo de $\forall z$:

$$\begin{array}{c} \forall z \exists w \forall y (P(y) \wedge Q(f_x) \rightarrow Q(z) \wedge R(w)) \\ \hline \forall z \forall y (P(y) \wedge Q(f_x) \rightarrow Q(z) \wedge R(g_w(z))) \end{array}$$

Pode-se, então, afirmar que essa última fórmula tem algum modelo se, e somente se, a fórmula inicial também:

$$\exists x (\forall x P(x) \wedge Q(x)) \rightarrow \forall x (Q(x) \wedge \exists x R(x)) \text{ é satisfável}$$

\iff

$\forall z \forall y (P(y) \wedge Q(f_x) \rightarrow Q(z) \wedge R(g_w(z)))$ é satisfável .

A partir dessa técnica, Skolem conseguiu um procedimento de prova (na verdade, de refutação) que não dependia de um sistema dedutivo em particular. O método consistia em substituir, sistematicamente, as variáveis universalmente quantificadas por todos os termos formados pelas constantes e símbolos funcionais da linguagem da fórmula em questão. Tais termos compõem o chamado *Universo de Herbrand* da fórmula. O que se chama na literatura, muitas vezes de *Teorema de Herbrand*, diz que a fórmula é insatisfável se, e somente se, alguma conjunção dessas substituições é falsa, no sentido do cálculo proposicional.¹² Herbrand, posteriormente, refinou esse resultado de maneira a funcionar para todas as fórmulas, não apenas para formas Skolemizadas. A demonstração de Herbrand também é mais informativa que os métodos semânticos de Skolem, tendo em vista que é construtiva: a partir de uma *derivação* em um sistema de primeira ordem, constrói-se uma outra, agora, de uma disjunção de fórmulas livres de quantificadores e fechadas. Da demonstração dessa disjunção, pode-se obter novamente a demonstração da fórmula original. Mais acertadamente, a versão que lida com insatisfabilidade é chamada de Teorema de Skolem-Herbrand-Gödel, enquanto reserva-se o nome de Teorema de Herbrand para o resultado sobre provabilidade. Na presente dissertação, no entanto, as duas versões não serão diferenciadas por tais nomes: a de Skolem será chamada de versão por *insatisfabilidade*, enquanto a devida a Herbrand de versão por *provabilidade*. No Capítulo 4, haverá um breve retorno às diferenças entre as duas versões. Para uma discussão detalhada do assunto, veja, por exemplo, [Gal85, p.303-304 e p.365].

Pode-se ver o grande foco sobre as noções de refutação e insatisfabilidade, o que está relacionado com o fato de que um dos métodos mais impactantes de dedução automática, o procedimento de *resolução*¹³, é um procedimento de refutação. Na verdade, no cálculo funcional clássico, os problemas da refutação, da insatisfabilidade e da validade universal são equivalentes¹⁴. Também é comum considerar

¹²Nesse sentido, o Teorema de Herbrand pode até ser visto como uma espécie de “finite model property” para sentenças quantificadas. (Walter Carnielli, em comunicação pessoal). Uma determinada lógica tem esta *propriedade de modelo finito* se todas suas fórmulas que não são teoremas têm contra-modelos finitos.

¹³Introduzido por Alan Robinson em [Rob65].

¹⁴Há também o fato de que o problema da validade universal (lógica) é um caso especial do problema da consequência lógica, isto é, uma sentença é universalmente válida exatamente quando é consequência lógica do conjunto vazio:

$$\varphi \text{ é válida} \iff \vDash \varphi$$

Algumas considerações sobre essas equivalências, na perspectiva das **LFI**'s, podem ser encontradas na Seção 5.1.

equivalentes os problemas da validade e da satisfatibilidade, como afirmam Hilbert e Bernays:

It is customary to refer to the two equivalent problems of universal validity and satisfiability by the common name of the decision problem of the restricted predicate calculus.

[HA50, p.113]

Mas, apesar de intimamente relacionados, os problemas da validade lógica e da satisfatibilidade, no que diz respeito ao seu grau de irresolubilidade, não são equivalentes. A questão da validade universal é semi-decidível¹⁵, enquanto a da existência de modelos não é sequer semidecidível. No panorama clássico, o problema da satisfatibilidade é equivalente ao da existência de contramodelos:

$$\Gamma, \varphi \text{ é satisfável} \iff \Gamma \not\models \neg\varphi,$$

enquanto o da insatisfatibilidade, ao da consequência lógica:

$$\Gamma, \varphi \text{ é insatisfável} \iff \Gamma \models \neg\varphi.$$

1.2 Desenvolvimento da Programação Lógica

1.2.1 A Origem da Dedução Automática

A lógica clássica de primeira ordem¹⁶ encontra-se fundamentada em diversos formalismos, criados nas décadas de 1920 e 1930, à Hilbert ou à Gentzen. Tais formalismos foram desenvolvidos com o intuito de serem ferramentas de investigação metamatemática, seguindo as linhas gerais do programa de Hilbert de busca por demonstrações finitárias de consistência de teorias matemáticas formalizadas. Com o desenvolvimento dos computadores digitais, havia uma certa expectativa sobre as possibilidades de usarem-se os formalismos lógicos como ferramentas para “fazer” matemática, através de deduções automáticas de teoremas matemáticos realizadas pelos computadores digitais.

¹⁵Isto é, existe um procedimento capaz de identificar todas as fórmulas universalmente válidas, mas não existe procedimento capaz de sempre determinar quando as fórmulas não o são. Logo, o problema de decidir se uma fórmula é consequência lógica de um conjunto recursivamente enumerável, ou mesmo recursivo, de fórmulas ($\Gamma \models \varphi$) é, em geral, apenas semi-decidível.

¹⁶Que veio a ser considerada *a lógica por excelência*, posição hoje em dia desafiada por vários vieses, como atesta a enorme área de lógicas não clássicas, estando as paraconsistentes inclusas. Veja [Hin96] e [BF85] também.

A “onipresença” da lógica de primeira ordem, no tocante ao seu carácter dedutivo e interpretativo, influenciou fortemente a área de automatização de demonstrações desde sua origem e, posteriormente, da programação lógica. Possibilitou o estabelecimento de um formalismo sintático universalmente reconhecido, com suas linguagens com predicados e funções, unidos por conectivos proposicionais e quantificadores. Há também grande consenso quanto a uma semântica padrão (a concepção tarskiana de verdade para linguagens formalizadas, cf. [Tar44]). Além disso, as variadas espécies de cálculos (como os sistemas axiomáticos hilbertianos, os sequentes e a dedução natural) permitem que se realizem deduções arbitrárias a partir das informações codificadas em fórmulas de primeira ordem. Com efeito, considere-se o depoimento de Robert Kowalski¹⁷ em um artigo sobre o início da pesquisa em programação lógica:

Logic programming shares with mechanical theorem proving the use of logic to represent knowledge and the use of deduction to solve problems by deriving logical consequences.

[Kow88, p.38]

Os métodos de Skolem e Herbrand surgidos em meio às investigações metamatemáticas do programa de Hilbert, conforme visto na seção anterior, também foram determinantes sobre a forma como evoluíram os primeiros provadores de teoremas. A utilização de tais técnicas em dedução automática foi inicialmente sugerida por Abraham Robinson (sugestão essa publicada em [Rob57]), segundo Davis:

The first suggestion that methods based on “Herbrand’s theorem” were appropriate for general purpose theorem-provers seems to have been made by Abraham Robinson in a brief talk delivered at the Summer Institute for Symbolic Logic at Cornell University in 1954.

[Dav83, p.16]

Além disso, há outro fator fundamental que contribuiu para que a lógica clássica se tornasse o principal alvo da pesquisa em dedução automática: na lógica de primeira ordem clássica, há a possibilidade de reduzir todas as sentenças à forma normal conjuntiva. Tal característica foi um suporte do método divisor de águas na área de dedução automática: o procedimento de resolução introduzido em [Rob65]. Este método será o assunto da próxima seção.

¹⁷Kowalski é considerado um dos pais da programação lógica, por ser autor da resolução **SLD**, o método de dedução aplicado no contexto da programação lógica e do **PROLOG**. Tal método será abordado no Capítulo 2.

1.2.2 Resolução e Cláusulas de Horn

A resolução é uma regra de inferência: a única regra de inferência de um sistema de dedução projetado especialmente para ser usado por computadores, ao longo de procedimentos de dedução automática. Ela foi introduzida por Robinson em [Rob65]. Sistemas axiomáticos usuais são abstrações dos processos dedutivos empregados nas ciências formais (como as diversas áreas da Matemática) e têm outros propósitos. Uma característica comum a eles é a simplicidade de suas regras de inferência, que devem ser facilmente verificáveis por *humanos*. Isto não é necessário quando se quer tirar proveito das capacidades de cálculo dos computadores. Este método de refutação, baseado na versão por insatisfatibilidade do teorema de Herbrand, marcou época na pesquisa em demonstração automática. Segundo Davis:

The explosion of interest which has produced the field as we know it today can be traced to [Rob65] in which the elegance and simplicity of the resolution principles as a basis for mechanized deduction first appeared.

[Dav83, p.1]

Como visto na seção 1.1, o teorema de Herbrand (em uma de suas versões) fornece um procedimento de refutação para primeira ordem. Tal procedimento consiste em substituir as variáveis livres de uma fórmula sem quantificadores pelos termos fechados de sua própria linguagem e testar se o resultado, entendido como uma fórmula do cálculo proposicional, é válido (veja-se Seção 1.1, p.8). A resolução combina essas duas etapas em um único procedimento. Com base na resolução, foi possível estabelecer um método de refutação que permite decidir a insatisfatibilidade de fórmulas de primeira ordem. O método, vez ou outra, pode também chegar a determinar a satisfatibilidade de certas fórmulas de primeira ordem, mas não sempre, tendo em vista que tal problema é indecidível.

O método de Robinson assume que as fórmulas são de um tipo particular: as *cláusulas*. Aceita, como entrada, um conjunto finito de *cláusulas*, em que cada uma é um conjunto finito de *literais*. Os literais, por sua vez, são fórmulas atômicas (*literais positivos*), ou negações delas (*literais negativos*). Cada cláusula é interpretada como o fechamento universal da disjunção de seus literais e um conjunto de cláusulas como a conjunção de seus elementos. No que segue, C será usada para denotar algum conjunto de cláusulas, κ alguma cláusula e as letras minúsculas do fim do alfabeto denotarão as variáveis. Logo, as cláusulas:

$$\kappa_1 = \{P(f(x), y), \neg Q(z)\} \text{ e } \kappa_2 = \{\neg P(x, f(y)), Q(z)\}$$

representam, respectivamente:

$$\forall x, y, z (P(f(x), y) \vee \neg Q(z)) \text{ e } \forall x, y, z (\neg P(x, f(y)) \vee Q(z)).$$

E o conjunto C formado por essas duas cláusulas ($C = \{\kappa_1, \kappa_2\}$) representa:

$$\forall x, y, z (P(f(x), y) \vee \neg Q(z)) \wedge \forall x, y, z (\neg P(x, f(y)) \vee Q(z)).$$

Literais poderão ser denotados por L e fórmulas atômicas por A e B . Um conjunto de variáveis será representado por \vec{x} .

Formalmente, a resolução é um método para decidir a insatisfatibilidade de um conjunto finito de cláusulas. Na lógica clássica, isso é equivalente a provar que a negação de uma cláusula é consequência lógica das outras. Suponha-se que C é um conjunto finito de cláusulas e κ é uma cláusula. Então:

$$C \vDash \neg \kappa \iff C, \kappa \text{ é insatisfável.}$$

Também no contexto clássico, a negação de uma cláusula (que é universalmente quantificada) é equivalente à existência de elementos que falsifiquem simultaneamente seus literais. Dado um literal L , seu complemento \bar{L} é formado, se o literal é uma negação, excluindo-se o sinal de negação e, caso contrário, adicionando-se. Isto é, $\overline{\neg A} = A$ e $\bar{A} = \neg A$. Suponha, então, que $\kappa = \{L_0, \dots, L_n\}$ e em \vec{y} estão todas as variáveis de κ , então:

$$\vDash \neg \kappa \iff \vDash \exists \vec{y} (\bar{L}_0 \wedge \dots \wedge \bar{L}_n).$$

A resolução é um método de refutação que computa contra-exemplos para um conjunto finito de cláusulas. Isto é, trata-se de um método para computar substituições para as variáveis que falsifiquem alguma das cláusulas de entrada. Os contra-exemplos computados estão restritos unicamente aos termos formados pelos símbolos funcionais já presentes na entrada. Pelo teorema de Herbrand, o método tem sucesso exatamente quando a entrada é *insatisfável*. As substituições computadas são de um tipo especial, são os chamados *unificadores*. Unificadores são sempre relativos a um conjunto de expressões, isto é, um unificador é uma substituição que *unifica* um conjunto de expressões. Unificar expressões significa torná-las sintaticamente idênticas. Expressões são sequências de símbolos da assinatura de uma linguagem¹⁸ e incluem fórmulas, termos ou mesmo sequências desses. Por exemplo, para unificar as seguintes sequências de termos:

$$(f(x), y, z) \text{ e } (x', f(y'), z'),$$

¹⁸Para a definição de *assinatura*, veja-se a Definição 3.14, na Seção 3.3.2.

basta substituir em ambas x' por $f(x)$, y por $f(y')$ e z por z' , o que resulta em:

$$(f(x), f(y'), z') \text{ e } (f(x), f(y'), z').$$

Sobre substituições e unificadores, deve-se consultar a Seção 2.1.3, que traz mais detalhes.

A unificação é uma das etapas da resolução. Como já foi dito, a resolução recebe de entrada um conjunto finito de cláusulas. O passo básico do método consiste em encontrar dois literais de diferentes cláusulas de entrada que possam, depois de uma substituição, tornarem-se um a negação do outro. Ou seja, o método procede pela unificação de duas fórmulas atômicas, uma delas tomada de alguma cláusula e a outra formada pela eliminação do símbolo de negação de algum literal de uma cláusula diferente. Por exemplo, considere-se o conjunto C formado pelas seguintes cláusulas:

$$\begin{aligned} \kappa_1 &= \{\neg Q(x)\}, \\ \kappa_2 &= \{\neg P(x', f(y')), Q(x')\} \text{ e} \\ \kappa_3 &= \{P(f(x''), y'')\}. \end{aligned}$$

Se forem escolhidos os literais $\neg Q(x) \in \kappa_1$ e $Q(x') \in \kappa_2$, pode-se ver que pela simples substituição de x por x' , obtem-se literais os quais um é a negação do outro (ou seja, estão sendo unificadas as fórmulas atômicas $Q(x)$ e $Q(x')$). Seja C' o conjunto formado a partir de C pela realização de tal substituição sobre as cláusulas:

$$C' = \{ \{\neg Q(x')\}, \{\neg P(x', f(y')), Q(x')\}, \{P(f(x''), y'')\} \}.$$

Se a substituição for de um tipo especial, feita pelos chamados *unificadores mais gerais*, tem-se a garantia de que C é insatisfável exatamente quando C' o é. Sobre tais unificadores, veja-se a Seção 2.1.3. Por enquanto, basta saber que está sendo realizada apenas esse tipo especial de substituição. Na lógica clássica, há ainda outra propriedade fundamental para a programação lógica (clássica): $(A \vee B) \wedge (D \vee \neg B)$ é equivalente a $A \wedge D$. Portanto C' , por sua vez, é insatisfável exatamente quando o seguinte conjunto o é:

$$C'' = \{ \{\neg P(x', f(y'))\}, \{P(f(x''), y'')\} \}.$$

A cláusula $\{\neg P(x', f(y'))\} \in C''$ é chamada de *resolvente* de κ_1 e κ_2 . Pode-se dar sequência ao procedimento pela unificação de $P(f(x''), y'')$ com $P(x', f(y'))$ através da substituição de x' por $f(x'')$ e y'' por $f(y')$:

$$C''' = \{ \{\neg P(f(x''), f(y'))\}, \{P(f(x''), f(y'))\} \}.$$

Tal conjunto é equivalente em termos de insatisfatibilidade aos outros (C , C' e C'') e é insatisfável na lógica clássica. Para que fique mais clara a insatisfatibilidade, na sintaxe usual C''' fica:

$$\forall x'', y' (\neg P(f(x''), f(y'))) \wedge \forall x'', y' (P(f(x''), f(y'))).$$

Como havia sido dito, o processo de refutação é construtivo e resulta em elementos que, quando substituídos pelas variáveis, falsificam o conjunto inicial de cláusulas:

$$C = \{\{-Q(x)\}, \{-P(x', f(y'))\}, \{Q(x')\}, \{P(f(x''), y'')\}\}.$$

O conjunto das substituições feitas ao longo do processo descrito acima fornece exatamente esses elementos. Inicialmente, x foi substituído por x' e este por $f(x'')$. Tem-se também a substituição de y'' por $f(y')$. Logo, se essas substituições forem feitas diretamente em C , obtém-se o conjunto contraditório:

$$\{\{-Q(f(x''))\}, \{-P(f(x''), f(y'))\}, \{Q(f(x''))\}, \{P(f(x''), f(y'))\}\}.$$

À primeira vista pode parecer estranho que um método que sirva para encontrar contra-exemplos seja suficiente para determinar a insatisfatibilidade. A insatisfatibilidade é a impossibilidade de que haja um modelo. Significa que todas as substituições, em todos os possíveis domínios, são contra-exemplos. Logo, um método que determine a insatisfatibilidade encontrando *um único contra-exemplo* parece estranho. Está justamente nisso a força do teorema de Herbrand. Tal teorema possibilita *reduzir o espaço de busca* pela insatisfatibilidade aos termos formados pelo material sintático já presente nas fórmulas.

A resolução original tem sérios problemas quanto à eficiência de suas implementações. Quaisquer literais, de quaisquer cláusulas, podem ser selecionados para o cálculo dos resolventes durante a busca por uma refutação. Surgiu, então, uma restrição da resolução original que torna o processo de seleção mais eficiente. Tal restrição, a resolução **SLD** (abreviação de *Selection-rule driven Linear resolution for Definite clauses*), se restringe às chamadas *Cláusulas de Horn* (também chamadas de cláusulas definidas). Cláusulas de Horn são aquelas que têm, no máximo, um literal positivo. Os *programas lógicos* são conjuntos de cláusulas de Horn com exatamente um literal positivo, as *regras*. Cláusulas que não apresentam nenhum literal positivo são denominadas *consultas*. A resolução **SLD** recebe como entrada um programa lógico P e uma consulta N e, então, efetua a resolução entre a consulta e alguma regra de P . O ganho de eficiência vem do fato de que, para cada regra, é possível a seleção de um único literal (o literal positivo) para a resolução com N (pois essa possui apenas literais negativos). O procedimento é feito novamente para o resolvente N' da consulta original (que possui, novamente, apenas

literais negativos) e assim sucessivamente. O próximo capítulo traz mais detalhes sobre este método.

A programação lógica surgiu como a aplicação da resolução **SLD** a uma área relacionada à Inteligência Artificial: o processamento de linguagem natural. Restrita sua abrangência às Cláusulas de Horn (que estavam sendo usadas para descrever a gramática do francês), o método **SLD** introduzido em [Kow74] é completo e constitui a única regra de inferência do sistema chamado na época de **PROLOG** (de *PROgramation en LOGique*, cf. [Col93, p.331]). Portanto, a programação lógica tem sua origem imediata no desenvolvimento de duas principais linhas de pesquisa: uma ligada ao campo da Inteligência Artificial e a outra ao da dedução automática de teoremas¹⁹, como afirma [Llo87, p.1]. Em meio à pesquisa em Inteligência Artificial, o desenvolvimento de aplicações para processamento de linguagem natural serviu como primeira aplicação e motivo imediato para o desenvolvimento do **PROLOG**.

1.2.3 O Paradigma Declarativo: Algoritmo = Lógica + Controle

Dois maneiras de entender o significado de um programa, em geral, já estavam dadas na época. Pelo lado da *semântica operacional*, entende-se o significado de um programa através de sua ação direta sobre cada entrada. Pelo da *semântica denotacional*, um programa é entendido como o conjunto de todas as possíveis relações entre seus valores de entrada e os de saída. O viés operacional tenta dar conta dos processos pelos quais o programa chega às soluções. A vertente denotacional tenta entender a computação abstraindo seu lado dinâmico. Vê a computação como um processo já completado.

A programação lógica, seguindo uma idéia de Kowalski [Kow79], é uma maneira de deixar clara a diferença entre a parte declarativa (relacionada com a semântica denotacional) e a operacional dos programas. O programa lógico, em si, é uma especificação do problema de certa maneira *independente da computação da resposta*. Isto é possível pela interpretação do conjunto de suas cláusulas de Horn como sentenças da lógica clássica de primeira ordem. A parte operacional cabe ao interpretador, o programa responsável por executar a resolução **SLD**. Este programa usa alguma estratégia de seleção para escolher quais átomos das consultas serão resolvidos antes de quais. Paralelamente, a programação lógica pode também ser entendida como uma forma de imprimir um caráter operacional à própria lógica.

Os artigos seminais de Kowalski e van Emden ([Kow74] e [vEK76]) dialogam com essas duas perspectivas. Em [vEK76], os autores vão elaborar a semântica

¹⁹Tais áreas não devem ser vistas de maneira dicotômica, tendo em vista que, por exemplo, um dos focos iniciais da pesquisa em Inteligência Artificial foi na área de dedução automática.

dos programas lógicos em termos de pontos fixos de operadores de consequência direta. Tal teoria pode ser encarada como uma especialização da semântica denotacional, que começava a se delinear a partir de trabalhos como os de Scott [Sco70]. Como declarado pelos próprios autores, a semântica proposta podia ser encarada como um caso especial do teorema de completude para a lógica de primeira ordem clássica. Troca-se a interpretação tarskiana pela de pontos fixos e sua contra-partida, os sistemas axiomáticos, pela resolução **SLD**, que fornece uma descrição operacional do programa lógico:

We show that operational semantics is included in the part of syntax concerned with proof theory and that fixpoint semantics is a special case of model-theoretic semantics. With this interpretation of operational semantics as syntax and fixpoint semantics as semantics, the equivalence of operational and fixpoint semantics becomes a special case of Gödel's completeness theorem.

[vEK76, p.734]

É de se observar, no entanto, que esta clara diferenciação entre o lado *operacional* e o *declarativo* possibilitado pela programação lógica pode se tornar um pouco turva nas implementações de fato. Por exemplo, é comum que as implementações de **PROLOG** disponibilizem ao programador mecanismos de controle de caráter claramente operacional, como o **CUT** (que é usado para modificar a estratégia de resolução seguida pelo interpretador). Para detalhes, o leitor deve consultar [Llo87, §11].

1.2.4 Cálculo com Negações

O fragmento das cláusulas de Horn da lógica de primeira ordem pode ser interpretado como uma linguagem de programação. É expressivo o suficiente para definir qualquer função computável (para as definições e demonstrações desse fato, recomenda-se [Apt90, §4]). No entanto, sua sintaxe é limitada do ponto de vista da representação de informações. Conforme será visto com mais detalhes na Seção 2.3, nenhum literal negado é consequência lógica de um programa lógico. A possibilidade de se lidar com deduções de informações negativas a partir de programas lógicos está, portanto, necessariamente fora do alcance da lógica clássica.²⁰

A necessidade de deduzir informações negativas estava inicialmente relacionada às regras de inferência em bancos de dados dedutivos.²¹ Este tipo de banco

²⁰Para um resultado corroborando esta afirmação, veja-se o Lema 2.53 da Seção 2.3.

²¹Veja-se [GMN78, pp.23–24].

de dados surgiu da necessidade de integrar os bancos de dados relacionais com a programação lógica. *Bancos de dados* são sistemas usados em computação para armazenar informações. Os *bancos de dados relacionais*, introduzidos em [Cod69], atacam este problema pela codificação de conjuntos de informações como relações definidas sobre domínios finitos. Por exemplo, um banco relacional para armazenar nomes de pessoas conhecidas poderia fazer uso de uma relação unária R , definida sobre o conjunto dos possíveis nomes de pessoas P , e armazenar os nomes “Maria”, “Joao” e “Antonio”:

$$R \subseteq P \text{ e}$$

$$R = \{(Maria), (Joao), (Antonio)\} .$$

Os *bancos de dados dedutivos* usam a programação lógica para especificar esse tipo de relação por meio dos *fatos*, cláusulas de Horn formadas por um único literal positivo. O exemplo anterior pode ser obtido pelo seguinte conjunto de cláusulas C^R :

$$C^R = \{ \{R(Maria)\}, \{R(Joao)\}, \{R(Antonio)\} \} .$$

Os bancos de dados dedutivos possibilitam consultas às informações por meio da resolução **SLD**. Tal abordagem não admite símbolos funcionais, pois isto implicaria necessariamente em domínios infinitos de interpretação. Esta restrição da programação lógica denomina-se **Datalog** e, como na programação lógica, deduz apenas consequências lógicas clássicas.

Em sistemas lógicos, a informação negativa é tratada da mesma maneira que a positiva: é deduzida a partir dos axiomas por regras de inferência. Em bases de dados dedutivas, é interessante que a informação negativa esteja representada implicitamente. Isto é, um fato negativo $\neg F$ é deduzido sempre que se falha em deduzir F . Por exemplo, para um conjunto de cláusulas C^{filho} com os seguintes fatos:

$$C^{filho} = \{ \{filho(Joao, Maria)\}, \{filho(Antonio, Joao)\} \} ,$$

indicando que João é filho de Maria e que Antônio é filho de João, seriam esperadas as seguintes deduções:

$$C^{filho} \vdash \neg filho(Maria, Joao)$$

$$C^{filho} \vdash \neg filho(Antonio, Maria) .$$

E isso, como dito, é impossível do ponto de vista da lógica clássica.

Portanto, está clara a necessidade de duas maneiras de se deduzir negações: uma explícita, comum aos sistemas lógicos, e a outra implícita, para o tratamento de informação negativa em bases de dados e sistemas de representação de conhecimento. A forma implícita de dedução recebe diversos nomes na literatura:

It is called “conversion for negative information representation” by Nicolas and Gallaire ²², the “closed world assumption (CWA)” by Reiter ²³, and “interpreting negation as failure” as discussed in the chapter by Clark ²⁴.

[GMN78, p.23]

Do ponto de vista implícito, deduz-se a negação de um fato se não se tem indícios de sua veracidade. Um exemplo comum são as informações sobre horários de ônibus; se não consta algum ônibus saindo em determinado horário, pode-se entender – e acertadamente – que não sairá ônibus algum naquele horário. Não é necessário que a tabela registre todos os horários em que não há ônibus. Um exemplo interessante de negação implícita pode ser encontrado no campo do Direito. Considere como primitivo o predicado monádico *é inocente*. Em regimes de exceção, não é incomum que as pessoas sejam tomadas por culpadas, isto é, *não inocentes*, até que provem o contrário. Ou seja, na falta de informações mais detalhadas, caso não seja possível a demonstração de inocência, afirma-se a *não inocência*. Logo, em ditaduras, os julgamentos de *culpa* (entendida formalmente como *não inocência*) são formulados através de uma interpretação implícita da negação.

O outro tipo de negação demanda uma prova explícita para sua dedução. Seguindo o exemplo anterior, o *princípio da presunção de inocência*, afirmado em nossa Constituição e na Declaração dos Direitos do Homem e do Cidadão, diz o oposto: todo acusado deve ser considerado inocente até que se prove o contrário. Logo, para que isto se cumpra, negações de inocência devem ser julgadas explicitamente. Evidentemente, poderia-se inverter a análise se o predicado tomado como primitivo fosse o que afirmasse formalmente a culpa. Nesse caso, regimes de exceção fariam uso da interpretação explícita da negação para afirmar a inocência (entendida formalmente como *não culpa*).

A extensão da programação lógica por um tipo implícito de negação será abordada brevemente na Seção 2.3 do Capítulo 2. Por sua vez, a Seção 3.1 do Capítulo 3 tratará de um tipo explícito. Finalizamos assim este breve capítulo introdutório, no qual os aspectos históricos e as motivações que levaram ao desenvolvimento da programação lógica foram apresentados. Para que esta dissertação seja autocontida, o próximo capítulo tratará dos aspectos técnicos e formais da programação lógica clássica.

²²[NG78]

²³[Rei78]

²⁴[Cla78]

Capítulo 2

Aspectos Formais da Programação Lógica Clássica

Neste capítulo, será feito um breve apanhado dos fundamentos da programação lógica clássica. A programação lógica é uma maneira de programar computadores usando a lógica clássica de primeira ordem. Para descrever os programas faz-se uso da sintaxe dos predicados, símbolos de função e quantificadores. Para computar os resultados, empregam-se métodos de dedução automática. Em seu formato original, a programação lógica dispõe de uma sintaxe e regras de inferência limitadas aos fatos positivos (isto é, às fórmulas atômicas, *sem o conectivo de negação*) de uma linguagem de primeira ordem. Hoje em dia, já existe uma extensão sua bem estabelecida que permite a programação com fórmulas atômicas negadas: a programação lógica geral, que será brevemente abordada na última seção deste capítulo.

A programação lógica é o fundamento teórico do **PROLOG**, uma linguagem de programação de uso bastante difundido em áreas centrais à Inteligência Artificial e à linguística computacional. Como exemplo, é amplamente utilizado em processamento de linguagem natural e na representação de conhecimento.

Na terminologia da teoria de linguagens de programação (uma área específica da ciência da computação), o **PROLOG** é uma linguagem de programação de *alto nível*. Isso significa que ele nos permite programar de um patamar *mais abstrato*. Em linguagens de alto nível, é possível fazer um programa que não tenha que referenciar diretamente características físicas da máquina. Quanto ao nível de abstração, existe outro tipo de linguagem de programação: as linguagens de *baixo nível*, que demandam uma interação mais direta entre o programador e características particulares de cada computador. Dessa maneira, aos usuários de linguagens de baixo nível, por exemplo, cabe a disposição completa dos dados pela memória.

Através do conceito de *variável*, tal tarefa é automatizada em linguagens de alto nível. A interpretação usual das variáveis, em linguagens de programação, é de que elas denotam *regiões* da memória do computador. A vantagem sobre as de baixo nível, que também têm que disponibilizar o acesso à memória, fica em livrar o programador de saber exatamente quais regiões serão usadas por quais variáveis, entre outros detalhes. O processo de conversão de um programa escrito em uma linguagem de alto nível para um código na linguagem própria da máquina (seu *assembly*) é denominado *compilação* e executado por programas especiais chamados *compiladores*. Pode ser que não haja explicitamente esta conversão, mas que o programa seja interpretado por um outro, chamado de *interpretador*, que faz as vezes de uma máquina de Turing universal. Em tais casos, diz-se que a linguagem é *interpretada*.

O **PROLOG** é a concretização da programação lógica como uma ferramenta, normalmente um interpretador, programado em computadores digitais. Se trata de uma maneira de especificar os fatos (da gramática de alguma língua, ou de algum outro campo de conhecimento, por exemplo) na linguagem de predicados da lógica de primeira ordem para, então, deduzir outros fatos que seguem logicamente desses primeiros. A única regra de dedução de que faz uso a programação lógica em geral é a *resolução SLD*, que será explicada na Seção 2.1.4. O **PROLOG** também usa tal regra para efetuar suas computações.¹

Os aspectos formais da programação lógica foram, em grande parte, determinados por sua origem, que, como visto no capítulo anterior, está nas investigações em dedução automática. A resolução **SLD**, por exemplo, é uma restrição da *resolução*, um procedimento geral para a demonstração automática de teoremas. Ambas essas formas de resolução são costumeiramente entendidas como procedimentos de refutação, isto é, como maneiras automáticas para demonstrar *negações*. Outra característica que possuem em comum é a sintaxe de suas linguagens. A resolução geral admite, como entrada, conjuntos de *cláusulas*, enquanto a **SLD** exige *cláusulas de Horn*. Será visto, na Seção 2.1, o que são esses dois tipos de cláusulas e como conjuntos formados pelo segundo tipo podem ser entendidos como especificações de programas.

A programação lógica clássica, apesar de fazer uso de um procedimento de refutação, não admite negações em seus programas, nem é capaz de demonstrar qualquer fato negativo. As demonstrações com a resolução **SLD** são da *negação da negação* de algum predicado. Classicamente, esse tipo de dedução equivale a deduzir um fato positivo. Há, no entanto, métodos já bem estabelecidos para

¹Para todos os efeitos, não serão abordadas as diferenças entre programação lógica e **PROLOG**. Apenas chama-se a atenção para o fato de que o **PROLOG**, por razões de eficiência, faz uso de certas alterações na resolução **SLD**. Para uma breve descrição das diferenças, veja [Apt90, p.658].

concluir fatos negativos. Na Seção 2.3, será analisado um em particular, a chamada *negação por falha*. A melhor referência para os temas abordados neste capítulo é o livro de Lloyd [Llo87]. Um bom apanhado, sucinto porém completo, encontra-se em [Apt90]. Também fez-se uso do texto mais recente [Doe94].

2.1 Sintaxe

2.1.1 Linguagens de Primeira Ordem

Programas lógicos são conjuntos de certos tipos de fórmulas de alguma linguagem de primeira ordem. Em geral, uma fórmula de primeira ordem é uma sequência de elementos da *assinatura* de sua linguagem. Assinaturas de primeira ordem são formadas por símbolos de variáveis, funções, predicados e as constantes lógicas: os quantificadores (\forall e \exists) e conectivos (\neg , \rightarrow , \wedge e \vee). Por ser de primeira ordem, as variáveis estão restritas aos elementos dos domínios de interpretação, não podem assumir, como valores, predicados ou funções. Para uma descrição mais detalhada desse tipo de linguagem, o leitor é remetido à Seção 3.3.2. Além desses elementos, no que segue, será feito uso de duas constantes lógicas especiais: a disjunção vazia, simbolizada por \square e a conjunção vazia, denotada por \square^\sim . Uma disjunção é verdadeira quando pelo menos um de seus elementos é, logo, o significado de \square será sempre falso (dado que não tem nenhum elemento). Simetricamente, uma conjunção é verdadeira quando todos os seus elementos o são, logo, o significado de \square^\sim é sempre verdadeiro.

As fórmulas mais básicas de uma linguagem de primeira ordem são suas fórmulas atômicas, constituídas por um símbolo de predicado composto com termos. Termos são expressões que denotam elementos do domínio de interpretação. Por exemplo, suponha que se esteja formalizando as relações de parentesco de um conjunto de pessoas. Dadas as especificidades do grupo que se está tratando, poderia-se decidir fazer uso de símbolos de constantes para denotar as pessoas e de um único predicado $F(x, y)$ para simbolizar a relação *x é filho de y*.

Como termos, haveria, por exemplo:

Maria, João, Zequinha, Fernanda

e, como fórmulas atômicas:

$F(\text{Zequinha}, \text{Maria}), F(\text{Fernanda}, \text{João})$,

para simbolizar que *Zequinha é filho de Maria* e *Fernanda é filha de João*.

Uma tentativa de, nesta linguagem restrita, expressar que Fulano esteve casado com Sicrano, poderia ser dada por:

$\exists x (F(x, \text{Fulano}) \wedge F(x, \text{Sicrano}))$.

Evidentemente, foram feitos pressupostos quanto a casamentos e a existência de filhos que podem não corresponder à realidade.

Uma sequência de quantificadores universais $\forall x_1 \dots \forall x_n$ poderá ser simbolizada por $\forall \vec{x}$, representando por \vec{x} o conjunto de variáveis $\{x_1, \dots, x_n\}$. O mesmo vale para uma sequência de existenciais $\exists y_1 \dots \exists y_n \equiv \exists \vec{y}$, com $\vec{y} = \{y_1, \dots, y_n\}$. Uma ocorrência de uma variável x em uma fórmula de primeira ordem φ é dita livre, se ocorre fora do escopo de um quantificador $\forall x$. O fechamento universal de uma fórmula φ , com todas suas variáveis livres em \vec{x} , é denotado por $\forall \varphi \equiv \forall \vec{x}(\varphi)$ e, caso não tenha nenhuma variável livre: $\forall(\varphi) \equiv \varphi$. As mesmas notações se estendem para o quantificador existencial: $\exists(\varphi) \equiv \exists \vec{y} \varphi$.

2.1.2 Programas Lógicos

Uma grande novidade introduzida pela programação lógica é a possibilidade de definir programas de computador por meio da lógica clássica de primeira ordem. Para se chegar a definição de um *programa lógico*, são necessários alguns conceitos auxiliares, como os de *literal*, *cláusula* e *cláusula de Horn*.

Definição 2.1 (Literal). Um *literal* é uma fórmula atômica, ou a negação de uma fórmula atômica.

■

No que segue, fórmulas atômicas serão denotadas pelas metavariáveis A , B ou C e literais por L , possivelmente acompanhados de subscritos. A possibilidade de um símbolo será indicada colocando-o entre colchetes (“[]”). Por exemplo, é possível representar um literal “ L ” por “[\neg]A”.

O método de resolução foi introduzido por Robinson em [Rob65] e é um método de refutação. Sua entrada é constituída por um conjunto de *cláusulas*.

Definição 2.2 (Cláusula). Uma *cláusula* κ é um conjunto finito de literais:

$$\kappa = \{L_1, \dots, L_n\}.$$

■

Uma cláusula é sempre interpretada como a quantificação universal da disjunção de seus elementos (\vec{x} são todas as variáveis livres de κ):

$$\kappa \equiv \forall \vec{x} (L_1 \vee \dots \vee L_n).$$

Para o caso em que o número de literais de uma cláusula é zero, ela significa uma contradição. Tal interpretação se justifica porque uma cláusula é verdade sempre que algum de seus elementos é. Logo, quando não há qualquer elemento, a cláusula é trivialmente falsa. Nesse caso será denotada por \square :

$$\models \square \iff \models \perp \iff \models \emptyset .$$

Um conjunto Ξ de cláusulas é, então, interpretado como a conjunção de seus elementos:

$$\begin{aligned} \Xi &= \{\kappa_0, \dots, \kappa_n\} \\ \Xi &\equiv \forall \vec{x} \kappa_0 \wedge \dots \wedge \forall \vec{x}' \kappa_n . \end{aligned}$$

Note-se que, na lógica clássica, para toda a fórmula, existe um conjunto de cláusulas que lhe é equivalente (em termos de insatisfatibilidade). Isso é uma consequência do teorema de Herbrand, na versão devida a Skolem (veja-se a Seção 1.1, p.8). A seguir, um exemplo simplificado será dado (sua skolemização já está na forma normal prenexa):

- Primeiramente, a fórmula original:

$$F = \exists x (\forall y A(y) \wedge \neg B(x)) \wedge \forall z (B(z) \vee \exists w \neg A(w)) ,$$

- e sua versão skolemizada:

$$\forall z \forall y \left((A(y) \wedge \neg B(f_x)) \wedge (B(z) \vee \neg A(f_w(z))) \right) .$$

- Pela versão de insatisfatibilidade do Teorema de Herbrand², tal fórmula é insatisfatível se, e somente se, alguma conjunção de instâncias fechadas da seguinte fórmula também é:

$$A(y) \wedge \neg B(f_x) \wedge (B(z) \vee \neg A(f_w(z))) .$$

Note que a fórmula acima já está na forma normal conjuntiva. Caso contrário, poderia-se passá-la a tal forma e garantir que o resultado final seja uma conjunção de cláusulas.

- Por fim, note que a seguinte instanciação de variáveis torna a fórmula proposicionalmente falsa e, portanto, a fórmula original é insatisfatível:

$$A(f_w(f_x)) \wedge \neg B(f_x) \wedge (B(f_x) \vee \neg A(f_w(f_x))) .$$

²Como visto no capítulo anterior (Seção 1.1, Página 8), tal teorema seria melhor chamado de Teorema de Skolem-Herbrand-Gödel.

Logo, se forem denotadas por κ_1 , κ_2 e κ_3 as cláusulas:

$$\begin{aligned}\kappa_1 &= \{A(f_w(f_x))\} \\ \kappa_2 &= \{\neg B(f_x)\} \\ \kappa_3 &= \{B(f_x), \neg A(g_w(f_x))\},\end{aligned}$$

tem-se que:

$$F \text{ é insatisfável} \iff \{\kappa_1, \kappa_2, \kappa_3\} \text{ é insatisfável}.$$

Denominam-se literais *negativos* aqueles formados pela negação de uma fórmula atômica. Positivos são os literais formados por uma fórmula atômica simplesmente. Dessa maneira, dada uma cláusula κ , pode-se dividi-la em dois conjuntos: sua parte positiva κ^P (com seus literais positivos) e negativa κ^N (com os negativos):

$$\begin{aligned}\kappa &= \kappa^P \cup \kappa^N \\ \kappa &= \{A_1, \dots, A_n\} \cup \{\neg B_1, \dots, \neg B_m\}.\end{aligned}$$

Podem, também, ocorrer os casos em que n ou m são iguais a zero.

Dadas tais considerações, uma cláusula pode ser expressa na chamada *notação clausal*:

$$\kappa \equiv (A_1, \dots, A_n) \leftarrow (B_1, \dots, B_m)$$

que simboliza:

$$(A_1 \vee \dots \vee A_n) \leftarrow (B_1 \wedge \dots \wedge B_m),$$

em que $A_i \in \kappa^P$ e $\neg B_j \in \kappa^N$. Como anteriormente, está subentendida a quantificação universal (em \vec{x} estão todas as variáveis de κ):

$$\kappa \equiv \forall \vec{x} \left((A_1 \vee \dots \vee A_n) \leftarrow (B_1 \wedge \dots \wedge B_m) \right).$$

A programação lógica clássica faz uso de um tipo especial de cláusula. Os programas e, também, as consultas a serem feitas a partir deles são codificados nas chamadas *cláusulas de Horn*.

Definição 2.3 (Cláusula de Horn). Uma *cláusula de Horn* é uma cláusula cuja parte positiva tem, no máximo, um literal.

■

Pode-se ver, então, que tais cláusulas são de dois tipos: com nenhum, ou com exatamente um literal positivo. Em notação clausal:

$$\begin{aligned}\kappa_1 &\equiv A_1 \leftarrow (B_1 \wedge \dots \wedge B_m) \\ \kappa_2 &\equiv \leftarrow (B_1 \wedge \dots \wedge B_m),\end{aligned}$$

com o segundo tipo equivalente a:

$$\kappa_2 \equiv \neg B_1 \vee \dots \vee \neg B_m .$$

Definição 2.4 (Programa Lógico). Um *programa lógico* é um conjunto de cláusulas de Horn com exatamente um literal positivo. ■

Ou seja, uma cláusula genérica de um programa lógico, denominada *regra*, tem a seguinte forma:

$$A_0 \leftarrow A_1 \wedge \dots \wedge A_n .$$

Denomina-se A_0 *cabeça da regra* e $A_1 \wedge \dots \wedge A_n$, seu *corpo*.

Definição 2.5 (Fatos). Regras com corpo vazio são chamadas de *fatos*:

$$A_0 \leftarrow .$$

Definição 2.6 (Consultas). Uma *consulta* é uma cláusula de Horn com nenhum literal positivo. ■

Consultas podem também ser chamadas de *cláusulas negativas*.

A programação lógica é uma maneira de especificar, por meio das cláusulas de Horn, programas e consultas a serem realizadas a tais programas. Os procedimentos de computação da programação lógica, e do **PROLOG**, partem de um conjunto de tais cláusulas constituído pela união de um programa P com uma consulta κ . A resolução **SLD** computa, então, se tal conjunto de fórmulas é insatisfável. Ou seja, para uma consulta $\kappa = \{\neg B_0, \dots, \neg B_m\}$, ou em notação clausal $\leftarrow (B_0, \dots, B_m)$, isso é equivalente ao fato de que, em cada modelo de P , é válida a negação de κ ($\neg \forall \kappa \equiv \exists (B_0 \wedge \dots \wedge B_m)$):

$P \cup \{\kappa\}$ tem uma refutação **SLD**

\iff

$P \cup \{\kappa\}$ é insatisfável

\iff

$\forall P \models \exists \kappa .$

Do ponto de vista da programação lógica, uma cláusula como $A \leftarrow (B_1, \dots, B_m)$ tem duas interpretações. Pode-se entendê-la pelo viés *declarativo*: A é verdadeiro, se as fórmulas B_1, \dots, B_m também são. A outra maneira é a interpretação *procedural*: para resolver A , é necessário resolver B_1, \dots, B_m antes. Tais interpretações são exemplos, na programação lógica, das semânticas *denotacionais* e *operacionais* de linguagens de programação. Tais interpretações são oriundas da teoria das semânticas de linguagens de programação e se tratam de duas maneiras de entender um programa. Do ponto de vista denotacional, um programa é visto como uma descrição estática de *qual deveria ser o estado de coisas* para que o resultado esteja *correto* com respeito aos valores de entrada do programa. Do ponto de vista operacional, um programa é uma *receita de como*, partindo de valores de entrada, se pode ir *construindo a saída pretendida*. Na prática, as linguagens de programação são influenciadas por ambas e tendem a privilegiar uma ou outra interpretação, conforme sua especificidade. O **PROLOG**, idealmente, é uma linguagem de programação de viés prioritariamente denotacional. Mais informações podem ser encontradas na Seção 1.2.3.

2.1.3 Substituições e Unificadores mais Gerais

Nas linguagens de programação de alto nível, é possível fazer uso de *variáveis*: abstrações para indicar as entradas e saídas dos programas e também para denotar estágios intermediários da computação. Por exemplo, em linguagens imperativas (que favorecem a interpretação operacional de seus programas), é possível escrever o seguinte programa que, dados dois valores de entrada, retorna sua média aritmética:

```
media(X,Y)
  Z = X + Y
  Z = Z/2
  retorna Z
```

Em que X e Y são usadas para indicar as duas entradas e Z para efetuar os cálculos intermediários e indicar o valor de retorno. Cada linha do programa deve ser entendida como uma *instrução* para a máquina. O programa pode ser interpretado como algo do tipo:

- *Receba dois valores de entrada,*
- *efetue sua soma e armazene-a em algum lugar da memória,*

- *divida o valor do passo anterior por dois e armazene-o no mesmo lugar*
- *retorne como resultado o valor obtido pela última instrução.*

Já o uso das variáveis feito pela programação lógica é diferente e se assemelha mais à maneira como as variáveis são tratadas na lógica matemática. Na lógica de primeira ordem, as variáveis são componentes sintáticos associados aos quantificadores e usados para denotar elementos dos domínios das interpretações. Na programação lógica, as variáveis são atribuídas por substituições, da mesma maneira que na lógica matemática. Mas as substituições são feitas automaticamente, através de um método originário de procedimentos de dedução automática: o algoritmo de *unificação*. Tal algoritmo é usado para encontrar substituições de um tipo especial, os chamados *unificadores mais gerais* (umg's).

Definição 2.7 (Substituição). Fixada uma linguagem de primeira ordem L , uma substituição é um função cujo domínio é algum conjunto finito de variáveis e o contradomínio, o conjunto dos termos de L .

■

A substituição das variáveis x_i pelos termos t_i , com $0 \leq i \leq n$, será representada da seguinte maneira:

$$\Theta = \{x_1/t_1, \dots, x_n/t_n\}.$$

Esta notação também implica que as variáveis x_1, \dots, x_n são diferentes umas das outras e que $x_i \neq t_i$. Uma substituição é dita *fechada* se não há variáveis livres em seus termos t_i .

Definição 2.8 (Renomeação). Uma *renomeação* é uma substituição cuja imagem é seu próprio domínio e, além disso, é injetiva.

■

Ou seja, *renomeações são permutações de variáveis*. Substituições são aplicadas sobre *expressões* quaisquer da linguagem. Isto é, para qualquer sequência E de símbolos da assinatura de L , $E\Theta$ representa a expressão obtida a partir de E pela substituição *simultânea* de todas as ocorrências das variáveis do domínio de Θ , pelos respectivos termos $t_i = \Theta(x_i)$.

Definição 2.9 (Instância, Instância fechada e Variante). A expressão $E\Theta$ é chamada de uma *instância* de E . Uma instância é dita fechada se não tem variáveis livres e uma *variante* se a substituição é uma renomeação.

■

Por exemplo, $x' < z' \leftarrow x' < y, y < z'$ é uma variante de $x < z \leftarrow x < y, y < z$, uma vez que:

$$x' < z' \leftarrow x' < y, y < z' = (x < z \leftarrow x < y, y < z)\{x/x', z/z', x'/x, z'/z\}$$

e, $0 < 2 \leftarrow 0 < 1, 1 < 2$ é uma instância fechada, dado que:

$$0 < 2 \leftarrow 0 < 1, 1 < 2 = (x < z \leftarrow x < y, y < z)\{x/0, z/2, y/1\}.$$

Substituições podem ser compostas. Dadas duas substituições:

$$\Theta = \{x_1/t_1, \dots, x_n/t_n\} \quad \text{e} \quad \eta = \{y_1/s_1, \dots, y_m/s_m\},$$

sua composição $\Theta\eta$ é definida a partir do conjunto:

$$\{x_1/t_1\eta, \dots, x_n/t_n\eta, y_1/s_1, \dots, y_m/s_m\}$$

pela remoção dos pares $x_i/t_i\eta$, para os quais $x_i = t_i\eta$ e também dos pares y_i/s_i , para os quais $y_i \in \{x_1, \dots, x_n\}$. Por exemplo, se $\Theta = \{x/3, y/f(x, 1), z/w\}$ e $\eta = \{x/4, w/z\}$, $\Theta\eta = \{x/3, y/f(4, 1)\}$.

O próximo teorema garante que os diferentes agrupamentos possíveis de uma sequência de substituições não alteram o resultado final (dessa maneira, não se precisa preocupar com a parentização). Sua demonstração será omitida por ser uma consequência quase imediata das definições de substituição e composição de substituições.

Teorema 2.10. *Para todas as substituições Θ, η, γ e expressão E :*

1. $(E\Theta)\eta = E(\Theta\eta)$,
2. $(\Theta\eta)\gamma = \Theta(\eta\gamma)$.

Diz-se que uma substituição Θ é mais geral que outra substituição η se existir uma terceira γ tal que:

$$\eta = \Theta\gamma.$$

Unificadores são um tipo de substituição, usados para computar as saídas dos programas lógicos.

Definição 2.11 (Unificador). Uma substituição Θ é um *unificador* das expressões H e J , se:

$$H\Theta = J\Theta.$$

■

Um unificador Θ de A e B é dito um *unificador mais geral* se é mais geral que qualquer outro unificador de A e B . Ou seja, Θ é um unificador mais geral de A e B se:

$$A\Theta = B\Theta$$

e, toda vez que $A\eta = B\eta$, deve haver uma substituição γ tal que:

$$\eta = \Theta\gamma.$$

Exemplo 2.12. Sejam $H = f(x)$, $g(f(z))$ e $J = f(g(x'))$, $g(z')$ duas expressões a serem unificadas. Isso pode ser obtido pelo unificador $\eta = \{x/g(a), z/b, x'/a, z'/f(b)\}$, com a e b constantes:

$$H\eta = f(g(a)), g(f(b)) = J\Theta.$$

No entanto, η não é um umg, pois $\Theta = \{x/g(x'), z'/f(z)\}$ é também um unificador de H e J para o qual não existe γ tal que:

$$\Theta = \eta\gamma.$$

Por outro lado, Θ é um umg de H e J . Por exemplo:

$$\eta = \Theta\{x'/a, z/b\}.$$

Teorema 2.13 (Teorema da Unificação). *Existe um algoritmo que recebe como entrada quaisquer duas expressões e retorna, se elas são unificáveis, um unificador mais geral para ambas. Caso contrário, retorna a informação de que não é possível unificá-las.*

Dado o escopo da presente dissertação, o algoritmo e a demonstração de sua correteza serão omitidos. Este resultado (a demonstração da correteza em relação aos unificadores mais gerais, juntamente com um algoritmo) é devido a Robinson [Rob65].

Definição 2.14 (Unificadores Relevantes). Um umg Θ de H e J é *relevante* se todas as variáveis que aparecem em seu domínio ou nos termos de sua imagem ocorrem em H ou J .

■

O próximo teorema garante que pode-se restringir apenas a unificadores relevantes. Sua demonstração é uma consequência direta das características do algoritmo de unificação e, portanto, não será apresentada (veja-se [Apt90, pp.500-502]).

Teorema 2.15. *Se duas fórmulas atômicas são unificáveis, então têm um umg que é relevante.*

2.1.4 Resolução SLD

A programação lógica realiza suas computações pela combinação de dois mecanismos: o cálculo de unificadores mais gerais e a substituição de subfórmulas das consultas pelo corpo de alguma regra do programa. O cálculo dos unificadores é realizado pelo algoritmo da unificação mencionado na seção anterior. A substituição de subfórmulas é realizada sobre consultas a partir de regras dos programas, através dos *resolventes*.

Definição 2.16 (Resolvente). Seja $\kappa = A \leftarrow (B_1, \dots, B_k)$ uma regra e $N = \leftarrow (A_1, \dots, A_n)$ uma consulta. Se, para algum i ($1 \leq i \leq n$) A_i e A são unificáveis pelo unificador mais geral Θ , então:

$$N' = \leftarrow (A_1, \dots, A_{i-1}, B_1, \dots, B_k, A_{i+1}, \dots, A_n)\Theta$$

é denominado um *resolvente* para N e κ , com unificador mais geral Θ . ■

Definição 2.17 (Derivação SLD). Uma *derivação SLD* de um programa lógico P e uma consulta N é uma sequência N, N_1, N_2, \dots de consultas juntamente com uma sequência $\kappa_0, \kappa_1, \dots$ de variantes de cláusulas de P e uma sequência $\Theta_0, \Theta_1, \dots$ de substituições tais que, para todo $i = 0, 1, \dots$:

1. N_{i+1} é um resolvente de N_i e κ_i com unificador mais geral Θ_i ,
2. κ_i não tem variáveis em comum com $N_0, \kappa_0, \dots, \kappa_{i-1}$,

de tal maneira que não são admitidas derivações finitas (digamos, com cláusula negativa final N_F), quando ainda é possível encontrar uma variante de alguma cláusula de P e um resolvente para tal cláusula e N_F . ■

As cláusulas $\kappa_0, \kappa_1, \dots$ são chamadas de *cláusulas de entrada* da derivação. Diz-se, quando uma das N_i é vazia (e, portanto, a última cláusula negativa da derivação), que tal derivação é uma *refutação SLD*. Uma derivação **SLD** *falhou* (ou *falhou finitamente*) se é finita e não é uma refutação.

Exemplo 2.18. Considere-se o seguinte programa lógico P que formaliza algumas propriedades da função *sucessor*:

$$\begin{aligned} 0 = 0 &\leftarrow & (\kappa_1) \\ s(x) = s(y) &\leftarrow x = y & (\kappa_2) \\ x = y &\leftarrow s(x) = s(y) & (\kappa_3) \end{aligned}$$

Logo, a linguagem de P é constituída por um símbolo de constante e um símbolo de função (0 e s), além de um símbolo de predicado binário ($=$). Uma refutação para P e a consulta $N = \leftarrow (s(s(0)) = x)$ é formada pela seguinte sequência de resolventes, cláusulas de entrada e unificadores mais gerais (UmG's):

Resolventes	Cláusulas de Entrada	UmG's
$\leftarrow (s(s(0)) = x)$	$(s(x_1) = s(y)) \leftarrow (x_1 = y) \equiv \kappa_2\{x/x_1\}$	$\{x_1/s(0), x/s(y)\}$
$\leftarrow (s(0) = y)$	$(s(x_2) = s(y_1)) \leftarrow (x_2 = y_1) \equiv \kappa_2\{x/x_2, y/y_1\}$	$\{x_2/0, y/s(y_1)\}$
$\leftarrow (0 = y_1)$	$(0 = 0) \leftarrow () \equiv \kappa_1\{\}$	$\{y_1/0\}$
□		

O resultado final da computação é a composição dos UmG's:

$$\{x_1/s(0), x/s(y)\}\{x_2/0, y/s(y_1)\}\{y_1/0\} \equiv \{x_1/s(0), x/s(s(0)), x_2/0, y/s(0), y_1/0\}.$$

A substituição relevante para a consulta original: $\leftarrow (s(s(0)) = x)$ é a restrição da computação à sua única variável livre (x no caso): $\{x/s(s(0))\}$, desprezando-se o histórico das outras substituições.

A existência de uma refutação **SLD** para $P \cup \{N\}$, em que P é um programa e N uma consulta, pode ser vista como a derivação de uma contradição, a partir do conjunto inicial. Isso significa que, se $N = \forall \vec{x}(\neg A_1 \vee \dots \vee \neg A_n)$, o fato de que $P \cup \{N\}$ deriva uma contradição (ou seja, é insatisfável) é equivalente a:

$$P \vDash \exists \vec{x}(A_1 \wedge \dots \wedge A_n).$$

Um ponto importante é que a sequência de unificadores $\Theta_1, \dots, \Theta_k$ computados durante a refutação proporcionam justamente as *testemunhas* para o existencial sobre as variáveis de \vec{x} :

$$P \vDash (A_1 \wedge \dots \wedge A_n)\Theta_1, \dots, \Theta_k.$$

Esse resultado será provado no Teorema 3.56.

Definição 2.19 (Resultado da Computação). A restrição de $\Theta_1, \dots, \Theta_k$ às variáveis de N é chamada de *resultado da computação* de $P \cup \{N\}$.

■

Para um determinado programa P e alguma consulta $N = \leftarrow (A_1, \dots, A_n)$, as possíveis refutações **SLD** são necessariamente obtidas pela repetição dos passos:

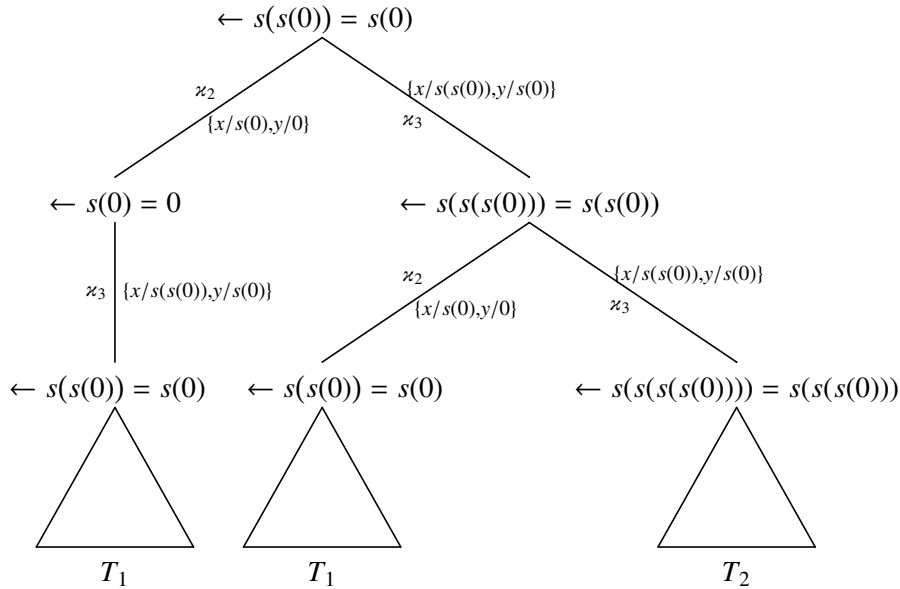
1. escolha de uma fórmula atômica A_i da consulta,

2. escolha de uma regra $\kappa = A \leftarrow (B_1, \dots, B_m)$ de P para a qual A seja unificável com A_i ,

até que não seja mais possível unificar nenhum átomo da consulta com nenhuma variante de cabeça de regra do programa. O primeiro item acima depende de todo o histórico da derivação. Tal histórico é constituído pela sequência de consultas, cláusulas de entrada e umg's que resultaram na cláusula N . Uma *regra de seleção* é uma função que, para cada histórico e consulta, retorna um átomo selecionado. **SLD** é uma sigla para *Selection rule-driven Linear resolution with Definite clauses* (resolução linear orientada a regras de seleção para cláusulas definidas). Cláusulas de Horn são também conhecidas como cláusulas definidas e *linear* é porque as derivações são *sequências* de consultas em que as posteriores são resolventes que possuem um antecedente entre as cláusulas anteriores da sequência. Portanto, a resolução **SLD** é um procedimento geral capaz de suportar uma multiplicidade de regras de seleção.

São possíveis, também, derivações infinitas.

Exemplo 2.20. Considere-se um exemplo para o qual não há possibilidade de derivações finitas (nem de refutação, nem de falha). Para a consulta: $\leftarrow s(s(0)) = s(0)$ e o programa P do exemplo anterior, serão representadas todas as possíveis derivações na seguinte árvore (os nós são consultas e as arestas trazem a informação de qual cláusula de P a cláusula de entrada é variante, bem como as substituições usadas para a unificação). Se um nó N' é ligado a um outro N'' por uma aresta marcada com κ_i , significa que N'' é um resolvente de N' e κ_i :



Em que T_1 e T_2 representam sub-árvores infinitas, nenhuma das quais contém \square e nem ramos finitos. Logo, não é possível nenhuma refutação (e, também, nenhuma falha finita).

Na sequência, serão enunciados dois resultados técnicos necessários à completude da resolução **SLD** (Teorema 2.52). Não será dada a demonstração do primeiro resultado, que pode ser encontrada em [Apt90, pp. 507-509].

Lema 2.21 (Lema das Variantes). [LS91] *Seja N_0, N_1, \dots e N'_0, N'_1, \dots duas derivações **SLD** de $P \cup \{N\}$ em que $N = N_0 = N'_0$, com as cláusulas de entrada C_0, C_1, \dots e C'_0, C'_1, \dots respectivamente. Suponha-se que cada C'_i é uma variante de C_i , que as fórmulas atômicas selecionadas em cada consulta N'_i e N_i estão na mesma posição e que os umg's são relevantes (Definição 2.14). Então, para $i \geq 0$, N_i e N'_i são variantes um dos outro.*

Corolário 2.22 (Corolário das Variantes). *Sejam Φ e Ψ duas derivações **SLD** para $P \cup \{N\}$ satisfazendo as condições do Lema 2.21. Suponha que Φ é uma refutação **SLD** com resultado computado Θ . Então, Ψ é uma refutação com resultado computado η tal que $N\Theta$ é uma variante de $N\eta$.*

Demonstração. Basta aplicar o Lema 2.21 para cada resolvente das duas refutações, passo a passo. \square

2.2 Semântica

2.2.1 Semântica para Lógica de Primeira Ordem

O significado de uma fórmula de primeira ordem é obtido classicamente pela definição de semântica devida a Tarski ([Tar44]). Será dada a seguir, resumidamente, tal definição. O leitor encontrará um tratamento mais elaborado (e geral) na Seção 3.3.3.

Uma interpretação para uma linguagem L de primeira ordem é constituída por:

- um conjunto não vazio D , chamado *domínio* de L ,
- para cada símbolo n -ário de função f da assinatura de L , uma função $f_I : D^n \rightarrow D$,
- para cada relação n -ária r da assinatura de L , um predicado $r_I \subseteq D^n$.

No que segue, será presumido que as linguagens sempre possuem uma constante para cada elemento dos domínios de suas interpretações. Evidentemente, tal constante é interpretada como o elemento correspondente.

A partir de uma interpretação, é possível definir o valor de verdade para cada fórmula da linguagem de L . Para começar, um termo composto $f(t_1, \dots, t_n)$ é interpretado recursivamente como $f_i(t_1^I, \dots, t_n^I)$, em que t_j^I é a interpretação do termo t_j . Então, para uma fórmula $\varphi \in L$, interpretam-se suas variáveis livres como se estivessem universalmente quantificadas e, conforme o tipo de φ , tem-se (recursivamente) as seguintes interpretações. Se φ é

- uma fórmula atômica $P(t_1, \dots, t_n)$, sua interpretação ($I \models P(t_1, \dots, t_n)$) é:

$$I \models P(t_1, \dots, t_n) \iff (t_1^I, \dots, t_n^I) \in P_I$$

- uma conjunção $\alpha \wedge \beta$:

$$I \models \alpha \wedge \beta \iff I \models \alpha \text{ e } I \models \beta$$

- uma disjunção $\alpha \vee \beta$:

$$I \models \alpha \vee \beta \iff I \models \alpha \text{ ou } I \models \beta$$

- uma implicação $\alpha \rightarrow \beta$:

$$I \models \alpha \rightarrow \beta \iff \text{não é o caso de } I \models \alpha \text{ ou } I \models \beta$$

- uma negação $\neg\alpha$:

$$I \models \neg\alpha \iff \text{não é o caso de } I \models \alpha$$

- um universal $\forall x \alpha$:

$$I \models \forall x \alpha \iff \text{para todo } d \in D, I \models \alpha_x[d']$$

- um existencial $\exists x \alpha$:

$$I \models \exists x \alpha \iff \text{para algum } d \in D, I \models \alpha_x[d']$$

Sendo que d' é a constante da linguagem que interpreta d .

2.2.2 Corretude da Resolução SLD

Agora será enunciada a corretude da resolução **SLD** em relação a semântica usual da lógica clássica de primeira ordem. Lembre-se que a resolução **SLD** é um procedimento de refutação, isto é, se ela é bem sucedida (termina em \square) tendo como entrada um programa P e uma consulta $N = \leftarrow (A_1, \dots, A_k)$, isto quer dizer que a negação de N é consequência lógica do programa: $\forall P \models \neg \forall (\neg A_1 \wedge \dots \wedge \neg A_k)$ ou, equivalentemente: $\forall P \models \exists (A_1 \wedge \dots \wedge A_k)$. Lembre-se também que $\exists \varphi$ e $\forall \varphi$ simbolizam os fechos existenciais e universais de φ , respectivamente. Para os próximos resultados, $\exists (A_1 \wedge \dots \wedge A_k)$ será simbolizada por N^\sim . Note que como um caso especial para a *cláusula vazia*, \square^\sim denotará a conjunção vazia, que é sempre verdadeira. Tendo isso em mente, a demonstração do próximo lema é trivial:

Lema 2.23. *Se M é um resolvente de uma consulta N e de uma cláusula \varkappa , com unificador mais geral Θ , então:*

$$\varkappa \models M^\sim \rightarrow (N^\sim \Theta).$$

Teorema 2.24 (Corretude da Resolução **SLD**). *Seja P um programa lógico e $N = \leftarrow (A_1, \dots, A_k)$ uma consulta. Suponha que exista uma refutação **SLD** de $P \cup \{N\}$ com a sequência de substituições $\Theta_0, \dots, \Theta_n$. Então:*

$$P \models (A_1 \wedge \dots \wedge A_k) \Theta_0 \dots \Theta_n.$$

Demonstração. Seja N_0, \dots, N_{n+1} , com $N_0 = N$ e $N_{n+1} = \square$, a sequência de cláusulas negativas de tal resolução **SLD**, com $\varkappa_0, \dots, \varkappa_n$ as cláusulas de entrada da derivação. Pela aplicação do Lema 2.23 $n + 1$ vezes, obtém-se:

$$P \models \square^\sim \rightarrow (N^\sim \Theta_0, \dots, \Theta_n),$$

o que implica o resultado. \square

Corolário 2.25. *Se existe uma refutação **SLD** de $P \cup \{N\}$, então tal conjunto é inconsistente.*

Demonstração. Imediata a partir do teorema anterior. \square

2.2.3 Modelos de Herbrand

Serão introduzidos, nesta seção, os conceitos semânticos necessários para a demonstração de uma forma de completude para a resolução **SLD**. Para tanto, será feito uso de uma classe especial de modelos para programas lógicos, os chamados *modelos de Herbrand*.

Seja L uma linguagem de primeira ordem com algum símbolo de constante (sempre que L não apresentar originalmente nenhuma constante, será tomada sua extensão por meio de uma constante arbitrária \star).

Definição 2.26 (Universo de Herbrand). O *Universo de Herbrand* de uma linguagem L é o conjunto dos termos fechados que podem ser formados das constantes e funções de L e será denotado por U_L .

■

Se, por exemplo, L for a linguagem do programa lógico P definido na seção anterior, então:

$$U_L = \{0, s(0), s(s(0)), \dots\}.$$

Para a linguagem L' do seguinte programa P' (que não possui nenhuma constante):

$$s(X) \leq s(Y) \leftarrow X \leq Y,$$

seu universo de Herbrand será:

$$U_{L'} = \{\star, s(\star), s(s(\star)), \dots\},$$

em que \star é uma constante nova.

Definição 2.27 (Base de Herbrand). A *Base de Herbrand* de uma linguagem de L é o conjunto das sentenças atômicas (isto é, das fórmulas atômicas fechadas) e será denotada por B_L .

■

Para os exemplos anteriores:

$$B_L = \{(0 = 0), (0 = s(0)), (s(0) = 0), (s(0) = s(0)), (s(s(0)) = 0), \dots\}$$

e:

$$B_{L'} = \{\star \leq \star, \star \leq s(\star), s(\star) \leq \star, s(\star) \leq s(\star), s(s(\star)) \leq \star, \dots\}.$$

Definição 2.28 (Instâncias de um Programa). Dado um programa P , define-se $inst(P)$ como todas as possíveis instâncias fechadas das cláusulas de P por substituições com contra-domínio em U_P .

■

Ou seja, para o programa definido logo acima:

$$inst(P') = \{(s(\star) \leq s(\star)) \leftarrow (\star \leq \star), (s(\star) \leq s(s(\star))) \leftarrow (\star \leq s(\star)), \dots\}.$$

Definição 2.29 (Interpretação de Herbrand). Uma *interpretação de Herbrand* para uma linguagem L , é uma interpretação para L , tal que:

1. seu domínio é o universo de Herbrand U_L de L ,
2. cada constante de L é interpretada por si mesma,
3. se f é um símbolo de função n -ário, então sua interpretação é a função $f^I : (U_L)^n \rightarrow U_L$ que associa a cada sequência t_1, \dots, t_n de termos fechados de U_L , o termo fechado $f(t_1, \dots, t_n)$,
4. se r é um símbolo de predicado n -ário, sua interpretação é algum subconjunto de $(U_L)^n$.

■

Desta forma, cada interpretação de Herbrand para L é unicamente determinada por um subconjunto I da base de Herbrand B_L . Isto é feito pela interpretação, de cada símbolo de predicado r de L , como o conjunto $\{(t_1, \dots, t_n) \mid r(t_1, \dots, t_n) \in I\}$. Ou seja, pode-se identificar interpretações de Herbrand de L com subconjuntos de sua base de Herbrand.

Definição 2.30 (Modelo de Herbrand). Um modelo de Herbrand para um conjunto de sentenças S , sobre uma linguagem L , é uma interpretação de Herbrand para L que é modelo de S .

■

Lema 2.31. *Seja S um conjunto de sentenças universais sobre uma linguagem L . Se S tem modelo, então também tem um modelo de Herbrand.*

Demonstração. Seja I um modelo de S . Tome $M_H = \{A \mid A \in B_L \text{ e } I \models A\}$. M_H determina uma interpretação I_H de L . Por indução no tamanho das fórmulas, não é difícil ver que I e I_H satisfazem as mesmas formulas fechadas livres de quantificadores. O lema é consequência direta disso. □

Dado que os universos de Herbrand definidos nesta seção são formados exclusivamente por símbolos já presentes na linguagem do programa (com excessão, possivelmente, da constante \star que não altera a validade de nenhum resultado), o próximo corolário pode ser entendido como uma versão do Teorema de Herbrand (em sua versão de satisfatibilidade, veja na Seção 1.1, p.8). Sua demonstração é imediata a partir do lema anterior, basta levar em consideração que a união de um programa com uma consulta é um conjunto de sentenças universais.

Corolário 2.32 (Teorema de Herbrand). *Seja P um programa e N uma consulta. $P \cup \{N\}$ é consistente se, e somente, tem um modelo de Herbrand.*

Por fim, serão introduzidos dois conceitos que muito recorrentes na área, os de *Modelo de Herbrand Minimal* e *Modelo de Herbrand Mínimo*. Lembre-se que interpretações (e modelos) de Herbrand podem ser identificados com os subconjuntos da base de Herbrand.

Definição 2.33 (Modelo de Herbrand Minimal). Para um conjunto S de fórmulas, diz-se que um modelo de Herbrand seu é *mínimo*, se nenhum subconjunto próprio dele é modelo de S .

■

Por exemplo, para $S = \{A \vee B\}$, com A e B diferentes fórmulas atômicas fechadas, são possíveis dois modelos de Herbrand minimais: $\{A\}$ e $\{B\}$, pois cada um desses conjuntos tem um único elemento e ambos são modelos de Herbrand de S .

Definição 2.34 (Modelo de Herbrand Mínimo). Dado um conjunto de fórmulas S , seu *modelo de Herbrand mínimo*, μ_S , é, se existir, o modelo de Herbrand contido em qualquer outro modelo de Herbrand de S .

■

Para o exemplo acima ($S = \{A \vee B\}$), não há modelo de Herbrand mínimo, apenas dois modelos minimais disjuntos.

2.2.4 Operador de Consequência Imediata e Pontos Fixos

O estudo de modelos de Herbrand de programas lógicos, seguindo a tradição iniciada em [vEK76], faz uso de uma ferramenta denominada *operador de consequência imediata*, que mapeia interpretações de Herbrand em interpretações de Herbrand.

Definição 2.35 (Operador de Consequência Imediata). Seja P um programa lógico, define-se o *operador de consequência imediata* T_P de P , para uma interpretação de Herbrand I do seguinte modo:

$$A \in T_P(I) \iff \text{para alguma substituição } \Theta \text{ e cláusula } B \leftarrow (B_1, \dots, B_n) \text{ de } P, \\ \text{tem-se } A \equiv B\Theta \text{ fechada e } I \models (B_1 \wedge \dots \wedge B_n)\Theta.$$

■

Para entender melhor o operador de consequencia imediata, considere o seguinte programa P :

$$\begin{array}{lcl} x = y & \leftarrow & s(x) = s(y) \\ 0 = 0 & \leftarrow & \end{array}$$

Se o termo $s^n(0)$ for representado pelo numeral n , tem-se:

$$\begin{aligned} T_P(\emptyset) &= \{ 0 = 0 \}, \\ T_P(\{ 5 = 4 \}) &= \{ 0 = 0, 4 = 3 \}, \\ T_P(T_P(\{ 5 = 4 \})) &= \{ 0 = 0, 3 = 2 \}. \end{aligned}$$

Lema 2.36. *Para um programa P e uma interpretação de Herbrand I , I é modelo de P , se, e somente se, $T_P(I) \subseteq I$.*

Demonstração. Primeiramente, é possível ver, pela definição da semântica que:

$$I \models P \iff I \models inst(P).$$

Agora, $I \models inst(P)$ se, e somente se, para toda cláusula $A \leftarrow (B_1, \dots, B_n) \in inst(P)$, $I \models B_1 \wedge \dots \wedge B_n$ implica $I \models A$, ou seja, $A \in I$. E isso vale se, e somente se, $T_P(I) \subseteq I$. \square

Na sequência, será obtida uma caracterização dos modelos de Herbrand mínimos de um programa P através de certos conceitos relacionados a teoria dos *reticulados completos*.

Definição 2.37 (Reticulado Completo). Um *reticulado completo* é um conjunto R munido de uma relação binária \leq que obedece às seguintes propriedades:

$$\begin{array}{lll} x \leq x & & \text{(Reflexividade)} \\ x \leq y, y \leq z & \implies & x \leq z \quad \text{(Transitividade)} \\ x \leq y, y \leq x & \implies & x = y \quad \text{(Anti-Simetria)} \end{array}$$

e, além disso, para todo conjunto $A \subseteq R$, existem elementos $\cup A, \cap A \in R$ (o *supremo* e o *ínfimo* de A , respectivamente) tais que:

$$\text{para todo } x \in A: \quad x \leq \cup A,$$

$$\text{para todo } x \in A: \quad \cap A \leq x$$

e

$$\text{se para todo } x \in A: \quad x \leq z, \quad \text{então} \quad \cup A \leq z,$$

$$\text{se para todo } x \in A: \quad z \leq x, \quad \text{então} \quad z \leq \cap A.$$

■

A família de subconjuntos de um conjunto base, com a relação de *contido em* (\subseteq), é um exemplo clássico de reticulado completo. Para isso, basta tomar o ínfimo e o supremo de uma família de conjuntos como sua intersecção e a união, respectivamente. O operador de consequência imediata T_P que foi definido acima é um exemplo de *operador monotônico* sobre reticulados completos. Tais operadores são aqueles que preservam a relação de ordem.

Definição 2.38 (Operador Monotônico). $T : R \rightarrow R$ é um *operador monotônico* sobre o reticulado (R, \leq) se, para $x, y \in R$:

$$x \leq y \quad \Longrightarrow \quad T(x) \leq T(y) .$$

■

Definição 2.39 (Operador Finitário). $T : R \rightarrow R$ é um *operador finitário* sobre o reticulado (R, \leq) se, para toda família $I = \{I_f\}_{f \in \omega}$ de elementos $I_f \in R$, tais que $I_f \leq I_{f+1}$, tem-se:

$$T\left(\cup I\right) \leq \cup\left\{T\left(I_f\right) \mid f \in \omega\right\} .$$

■

Definição 2.40 (Operador Contínuo). Um operador sobre um reticulado completo é *contínuo* se é monotônico e finitário.

■

Definição 2.41 (Pré Ponto Fixo). Um elemento $p \in R$, sendo (R, \leq) um reticulado completo e T um operador monotônico sobre R é dito *pré ponto fixo* de T , se:

$$T(p) \leq p .$$

■

Logo, pelo Lema 2.36, uma interpretação $I \subseteq B_P$ é modelo de P se, e somente se, é um pré ponto fixo de T_P . Portanto, para estudar os modelos de Herbrand de um programa P é suficiente estudar os pré pontos fixos de seu operador de consequência imediata T_P .

Definição 2.42 (Pós Ponto Fixo e Ponto Fixo). Dado um operador monotônico T sobre um reticulado completo com uma ordem \leq , p é um *pós ponto fixo* de T se:

$$p \leq T(p).$$

Se p é tanto um pré ponto fixo quanto um pós ponto fixo de T , ele é um *ponto fixo* de T :

$$p = T(p).$$

■

A seguir, está enunciado um teorema sobre pontos fixos de operadores monotônicos definidos sobre reticulados completos. A demonstração será omitida por se tratar de um resultado clássico de teoria da ordem e de reticulados, portanto, fora do escopo desta dissertação.

Teorema 2.43 (Teorema do Ponto Fixo). [Tar55] *Um operador monotônico T sempre tem um menor ponto fixo $\mu_{pf}(T)$ que é também seu menor pré ponto fixo.*

Definição 2.44 (Potências de um Operador). Seja T um operador monotônico sobre um reticulado completo. Define-se, por recursão:

$$T \uparrow 0 [I] = I,$$

$$T \uparrow (n + 1) [I] = T(T \uparrow n [I]),$$

$$T \uparrow \omega [I] = \bigcup_{n < \omega} T \uparrow n [I].$$

E abrevia-se $T \uparrow \alpha [\emptyset]$ por $T \uparrow \alpha$.

■

Lema 2.45 (Continuidade de T_P). *Seja P um programa. Então:*

1. T_P é *finitário*,
2. T_P é *monotônico*.

Demonstração. Para o ítem 1, considere-se a sequência $I_0 \subseteq I_1 \subseteq \dots$ de interpretações de Herbrand e suponha $A \in T_P(\cup \{I_n \mid n \in \omega\})$. Então, para algumas fórmulas atômicas B_1, \dots, B_k , a cláusula $A \leftarrow (B_1, \dots, B_n)$ está em $inst(P)$ e, além disso, $\cup \{I_n \mid n \in \omega\} \models B_1 \wedge \dots \wedge B_k$. Isto significa que, para algum I_n , aquele que contem todos os B_1, \dots, B_k , $I_n \models B_1 \wedge \dots \wedge B_k$. Logo, $A \in T_P(I_n)$.

O ítem 2 é uma consequência imediata da definição. □

O próximo teorema (o qual, como foi feito para o Teorema 2.43, apenas será enunciado sem demonstração), caracteriza os menores pré pontos fixos e pontos fixos de operadores contínuos.

Lema 2.46. *Se T é um operador contínuo, então $T \uparrow \omega$ é seu menor pré ponto fixo e também seu menor ponto fixo.*

Agora, será enunciado um teorema que resume as relações entre os conceitos de *modelo de Herbrand mínimo*, *menor ponto fixo*, *menor pré ponto fixo* e $T \uparrow \omega$.

Teorema 2.47 (Teorema da Caracterização). [vEK76] *Se P é um programa lógico, então ele tem um modelo de Herbrand M_P que satisfaz as seguintes propriedades:*

1. M_P é o menor modelo de Herbrand de P ,
2. M_P é o menor pré ponto fixo de T_P ,
3. M_P é o menor ponto fixo de T_P ,
4. $M_P = T_P \uparrow \omega$.

Demonstração. Pela aplicação do Lema 2.36, do Teorema 2.43 e do Lema 2.46. □

Definição 2.48 (Conjunto Sucesso). O *conjunto sucesso* de um programa P é o conjunto formado por todas as fórmulas A da base de Herbrand de P , para as quais existe um refutação **SLD** de $P \cup \{\leftarrow A\}$. ■

Corolário 2.49. *O conjunto sucesso de um programa P está contido em seu modelo de Herbrand mínimo.*

Demonstração. Pelo Corolário 2.25 e Teorema 2.47. □

Lema 2.50 (Lema da Substituição). *Seja P um programa, N uma consulta e Θ uma substituição. Suponha que exista uma refutação **SLD** para $P \cup \{N\Theta\}$. Existe, então, uma refutação **SLD** para $P \cup \{N\}$.*

Demonstração. Será feita por indução no tamanho n da derivação **SLD** de $P \cup \{N\Theta\}$. Pelo Corolário 2.22, pode-se assumir que Θ não atua sobre nenhuma das variáveis das cláusulas de entrada da refutação. Suponha-se que $N = \leftarrow (A_1, \dots, A_k)$.

Se $n = 1$, então $k = 1$ e $A_1\Theta$ é unificável com a cabeça de um fato (Definição 2.5) que é cláusula de entrada. Então, A_1 é unificável com a cabeça da mesma cláusula, o que demonstra o caso base.

Se $n > 1$, considere-se a primeira cláusula de entrada $B_0 \leftarrow (B_1, \dots, B_m)$ da refutação. Para um umg η , $A_i \Theta \eta = B_0 \eta$ em que $A_i \Theta$ é a fórmula atômica selecionada de $N \Theta$. Assim, pelo que assumido acima sobre Θ (que ela não atua sobre nenhuma das variáveis das cláusulas de entrada), tem-se que $A_i \Theta \eta = B_0 \Theta \eta$ e, então, A_i e B_0 são unificáveis. Para algum umg ξ e uma substituição γ , tem-se $\Theta \eta = \xi \gamma$.

Isso quer dizer que existe uma refutação **SLD** para:

$$P \cup \{ \leftarrow (A_1 \Theta, \dots, A_{i-1} \Theta, B_1 \Theta, \dots, B_m \Theta, A_{i+1} \Theta, \dots, A_k \Theta) \eta \},$$

de tamanho $n - 1$. Pela hipótese de indução, existe uma refutação **SLD** de:

$$P \cup \{ \leftarrow (A_1, \dots, A_{i-1}, B_1, \dots, B_m, A_{i+1}, \dots, A_k) \xi \}.$$

Considere, agora, uma derivação **SLD** de $P \cup \{N\}$ em que a primeira fórmula atômica selecionada é A_i e a primeira cláusula de entrada é $B_0 \leftarrow (B_1, \dots, B_m)$, com o umg ξ . Seu primeiro resolvente é $\leftarrow (A_1, \dots, A_{i-1}, B_1, \dots, B_m, A_{i+1}, \dots, A_k) \xi$, o que, pelo dito acima, conclui o lema. \square

Lema 2.51. *O modelo de Herbrand mínimo de um programa P está contido no conjunto sucesso de P .*

Demonstração. Suponha $A \in \mu_{pf}(T_P)$. Pelo item 4 do Teorema 2.47, para algum $k > 0$, $A \in T_P \uparrow k$. Por indução em k , será demonstrado que existe uma refutação **SLD** de $P \cup \{ \leftarrow A \}$. Para $k = 1$, é óbvio.

Se $k > 1$, para algumas fórmulas atômicas fechadas B_1, \dots, B_n , a cláusula $A \leftarrow (B_1, \dots, B_n)$ está em $inst(P)$ e $\{B_1, \dots, B_n\} \subseteq T_P \uparrow k - 1$. Pela hipótese de indução, para $i = 1, \dots, n$, existe uma refutação **SLD** de $P \cup \{ \leftarrow B_i \}$. Como todos os B_i são fechados, existe uma refutação de $P \cup \{ \leftarrow (B_1, \dots, B_n) \}$.

Considere, agora, uma refutação **SLD** de $P \cup \{ \leftarrow A \}$ com primeira cláusula de entrada uma da qual $A \leftarrow (B_1, \dots, B_n)$ seja uma instância fechada. Seu primeiro resolvente é uma cláusula negativa da qual $\leftarrow (B_1, \dots, B_n)$ é uma instância fechada. Logo, pelo Lema 2.50, tem-se o resultado. \square

Teorema 2.52 (Completeness da Resolução **SLD**). *Seja P um programa e N uma consulta. Suponha que $P \cup \{N\}$ é inconsistente. Existe, então, uma refutação **SLD** de $P \cup \{N\}$.*

Demonstração. Seja $N = \leftarrow (A_1, \dots, A_m)$. Pela hipótese, μ_P não é modelo de $P \cup \{N\}$. Logo, N não é válida em μ_P e, portanto, existe uma substituição Θ tal que $\{A_1 \Theta, \dots, A_m \Theta\} \subseteq \mu_P$. Pelo Lema 2.51, para $i = 1, \dots, m$ existe uma refutação **SLD** de $P \cup \{ \leftarrow A_i \Theta \}$ em que $A_i \Theta$ é uma fórmula fechada. Logo, existe uma refutação **SLD** para $P \cup \{N \Theta\}$ e, pelo Lema 2.50, existe uma refutação **SLD** para $P \cup \{N\}$. \square

2.3 Informação Negativa e Raciocínio não-Monotônico

Na programação lógica clássica, faz falta a possibilidade de especificar e computar informações negadas. Tendo em vista que a base de Herbrand B_P de um programa P é um modelo de P que *não verifica nenhum literal negativo*, logicamente, para todo literal negativo $\neg A$, tem-se:

$$P \not\models \neg A .$$

A resolução **SLD** é um método correto para primeira ordem clássica. Isto é, se o fato de que existe uma refutação **SLD** para $P \cup \{ \leftarrow (A_1, \dots, A_n) \}$ for denotado por $P \vdash_{\text{SLD}} \exists \vec{x} (A_1 \wedge \dots \wedge A_n)$, pelo Teorema 3.56:

$$P \vdash_{\text{SLD}} \exists \vec{x} (A_1 \wedge \dots \wedge A_n) \quad \Longrightarrow \quad P \models \exists \vec{x} (A_1 \wedge \dots \wedge A_n) .$$

Logo, a resolução **SLD** não pode deduzir nenhuma fórmula negada.

Uma relação de consequência \vdash é chamada *fracamente correta* se:

$$P \vdash \varphi \quad \Longrightarrow \quad P \cup \{ \varphi \} \text{ é satisfatível ,}$$

e diz-se que \vdash é *monotônica*, se não deixa de deduzir fatos que já deduzia ao acrescentarem-se premissas:

$$P \vdash \varphi \quad \Longrightarrow \quad P \cup P' \vdash \varphi .$$

Caso contrário, é chamada de *não-monotônica*. Claramente, a resolução **SLD** também é fracamente consistente e monotônica. O próximo lema mostrará que sua falta de expressividade, em relação à dedução de fórmulas atômicas negadas, é comum a qualquer método fracamente correto e monotônico para a dedução de literais a partir de programas.

Lema 2.53. *Seja \vdash um método de dedução para o qual $P \vdash \neg A$, com A uma fórmula atômica e P um programa lógico. Então, \vdash não é correto. Além do mais, se \vdash for fracamente correto, não é monotônico.*

Demonstração. Como já foi dito acima, a base de Herbrand B_P é sempre um modelo para P , em que $B_P \not\models \neg A$, para toda fórmula atômica A , e, portanto, \vdash não pode ser correto. Agora, suponha que tal método é monotônico. Logo, $P \cup \{A\} \vdash \neg A$, mas $P \cup \{A\} \cup \{\neg A\}$ é necessariamente inconsistente. Então, \vdash não pode ser fracamente correto. \square

Mesmo assim, faz sentido deduzir fatos negados de programa positivos em certos casos. Tome-se, como exemplo, a lógica das bases de dados. Bancos de dados

e seus relacionamentos podem ser representados como conjuntos de cláusulas sem símbolos funcionais. Por exemplo, se houvesse uma base de informações com o um predicado $P(x)$ para indicar se x é uma pessoa, o conjunto C das seguintes cláusulas pode indicar as pessoas de que o sistema tem conhecimento:

$$B = \{P(Joao) \leftarrow, P(Maria) \leftarrow, P(Natalia) \leftarrow, P(Zequinha) \leftarrow\}.$$

E poderia-se expressar o relacionamento de *ser filho de* por meio de um predicado binário $F(x, y)$ (x é filho de y). Logo, o conjunto de cláusulas R indica os parentescos que o sistema tem conhecimento:

$$R = \{F(Natalia, Joao) \leftarrow, F(Zequinha, Maria) \leftarrow\}.$$

É de se esperar, então, que se pudesse efetuar as seguintes deduções a partir de $B \cup R$:

$$B \cup R \vdash \neg P(pedra),$$

$$B \cup R \vdash \neg F(Joao, Natalia).$$

Mas, pelo Lema 2.53, qualquer extensão fracamente correta da resolução **SLD** que efetuasse essas derivações não poderia ser monotônica. As primeiras extensões da resolução **SLD** para deduzir literais negativas foram propostas justamente no contexto de bases de dados dedutivas. Nesta seção será abordada apenas a *negação por falha*.

A negação por falha foi introduzida originalmente em [Cla78] e será denotada por $\overset{\frown}{\neg}$. Tal regra permite deduzir a negação de uma fórmula atômica fechada A (representada por $\overset{\frown}{\neg} A$) a partir de um programa lógico P , se todas as possíveis derivações **SLD** feitas a partir de $P \cup \{\leftarrow A\}$ falham finitamente. A regra da negação por falha será abreviada por **NAF** e a extensão da resolução **SLD** pela admissão dessa regra será denotada por **SLDNF** (de resolução **SLD** com negação por falha):

$$P \vdash_{\text{SLDNF}} \overset{\frown}{\neg} A \quad \iff \quad P \cup \{\leftarrow A\} \text{ sempre falha finitamente.}$$

Pela completude da resolução **SLD** (Teorema 2.52), se $P \cup \{\leftarrow A\}$ não tem uma refutação **SLD**, então $P \cup \{\overset{\frown}{\neg} A\}$ é consistente ($\overset{\frown}{\neg}$ entendida semanticamente como \neg), o que demonstra que **SLDNF** é fracamente correto. Logo, pelo Lema 2.53, **SLDNF** é um método de dedução não-monotônico. Note-se que **NAF** deduz apenas as negações dos fatos para os quais todas as possíveis derivações **SLD** são finitas. Portanto, é uma regra efetiva de dedução (se o fato não tem derivações infinitas, nem refutações, é possível encontrar em tempo finito todas as suas possíveis derivações).

É possível a extensão da programação lógica pela admissão de literais negadas com a negação não-monotônica por falha ($\overset{\frown}{\neg}$) no corpo das regras.

Definição 2.54 (Programa Lógico Geral). Um *programa lógico geral* permite a negação por falha nas literais do corpo de suas regras:

$$A_1 \leftarrow [\neg]A_1, \dots, [\neg]A_n .$$

■

A partir desse tipo de programa é possível a dedução das consultas gerais.

Definição 2.55 (Consulta Geral). Uma *consulta geral* é simplesmente uma conjunção de literais de negação não-monotônica:

$$[\neg]A_1 \wedge \dots \wedge [\neg]A_n .$$

■

A resolução **SLDNF**, que não será analisada com detalhes na presente dissertação, se presta a deduzir o fechamento existencial de alguma consulta geral (*CG*) a partir de um programa lógico geral (*PG*):

$$PG \vdash_{\text{SLDNF}} \exists \vec{y} (CG) .$$

Com isto finalizamos uma rápida e sucinta descrição dos principais aspectos teóricos da Programação Lógica clássica. No próximo capítulo começaremos a dar os primeiros passos do nosso programa de pesquisa em programação Lógica paraconsistente, apresentando portanto os primeiros resultados originais desta dissertação.

Capítulo 3

Rumo à Programação Lógica Paraconsistente I: Bases Lógicas

Neste capítulo, será apresentada a primeira parte de nosso programa de pesquisa: estabelecer as bases lógicas sobre as quais poderá estar assentado um amplo arcabouço formal para programação lógica paraconsistente. Para tanto, define-se como ponto de partida o formalismo das **LFI's**, as Lógicas da Inconsistência Formal. Tais cálculos abarcam uma variada gama de raciocínios paraconsistentes. Uma característica destas lógicas é assumir, como primitivos, os próprios conceitos de *consistência e inconsistência*. Isto é feito pela admissão de um novo conectivo unário (\circ), que indica que alguma fórmula é consistente. Deste modo, as propriedades da consistência (e inconsistência) podem ser codificadas da maneira que melhor convém em cada situação, por intermédio de esquemas axiomáticos que envolvem tal conectivo.

No início, será feito um breve apanhado do desenvolvimento da programação lógica paraconsistente e abordada sua relação com a introdução de um tipo explícito de negação (veja-se Seção 1.2.4) nos programas lógicos. Tal negação tem caráter *monotônico*, diferentemente da *negação por falha* tratada na Seção 2.3. Será descrito, também, o fragmento monotônico comum a todas abordagens à programação lógica paraconsistente: o fragmento dos *Programas Lógicos Estendidos Definidos*. Tal fragmento tem uma série de caracterizações na literatura, mas carece de uma fundamentação lógica clara. Conjecturamos que certas **LFI's** explicitam justamente suas bases lógicas.

Espera-se, com a presente dissertação, dar um passo em direção a uma programação lógica paraconsistente, à maneira do que foi feito em [dAP07] para um **Datalog** paraconsistente. O **Datalog** é a restrição da programação lógica clássica a cláusulas sem símbolos funcionais e serve como linguagem de consulta a bancos

de dados (vejam-se as Seções 1.2.4 e 2.3). No artigo acima citado, as autoras introduzem uma linguagem de consulta a bancos de dados que admitem informações negadas e inconsistências sem trivializações, usando por base a lógica paraconsistente **LFI**. Tal sistema é denominado **P-Datalog**⁷ e, da mesma forma que o **Datalog**, não comporta linguagens com símbolos de funções e constantes. Tendo em vista que os domínios em foco são finitos (os bancos de dados), tudo se resolve com o caso proposicional. A programação lógica é uma extensão desse tipo de linguagem, pois admite símbolos de função e, portanto, demanda domínios enumeráveis.

Os programas lógicos estendidos definidos apresentam propriedades que os qualificam como um bom ponto de partida para o presente projeto. Entre elas, está a monotonicidade de sua relação de consequência (para a derivação de literais com a negação explícita) e o fato de que ela é necessariamente paraconsistente, como será visto na primeira seção deste capítulo. Estas características permitem que as **LFI**'s possam modelar sua relação de consequência sem maiores complicações. Além disso, as semânticas para o caso geral (os programas lógicos estendidos) sempre coincidem na interpretação que fazem deste fragmento (veja-se o Teorema 3.5). Posteriormente, pode-se estudar o caso mais geral (com a negação não-monotônica) a partir de uma base monotônica bem estabelecida. Mas, para tratar tais questões dentro do âmbito da paraconsistência sem se apelar para resultados da lógica clássica, se faz necessário um teorema de Herbrand para as **LFI**'s. Com isso, seria possível lidar com os símbolos de função e constantes e, por exemplo, estudar uma aritmética paraconsistente dentro do paradigma da programação lógica. Com tal teorema, tem-se a garantia de haver algum método finitário de dedução para um fragmento suficientemente grande para abarcar o necessário a uma legítima programação lógica paraconsistente estendida definida.

Por último, serão abordadas as **LFI**'s e suas extensões de primeira ordem, juntamente com uma prova de completude com relação à semântica de bivalorações paraconsistentes, definidas sobre estruturas clássicas de primeira ordem. Com isso, estão estabelecidos os resultados técnicos fundamentais para a continuação do programa de pesquisa, continuado no Capítulo 4.

3.1 Programação Lógica Paraconsistente

Como visto na Seção 1.2, o **PROLOG** surgiu tanto do desenvolvimento de métodos de dedução automática, como do desenvolvimento de procedimentos de computação relacionados à área de processamento de linguagem natural. Tais campos estão intimamente relacionados com o início da pesquisa em Inteligência Artificial (IA). Hoje em dia, uma das principais áreas da IA, a representação do conhe-

cimento, aplica amplamente a programação lógica. A IA é normalmente entendida como o estudo e desenvolvimento de sistemas sensíveis ao contexto, capazes de simular certas características que se assemelham à inteligência humana, como, por exemplo, a possibilidade de adquirir novas informações e corrigir sua atuação frente às novidades. Logo, a representação do conhecimento está nos fundamentos da IA: se deseja-se projetar algum programa para atuar, ou guiar inteligentemente nossas ações em algum ambiente ou situação, deve-se supri-lo com as informações necessárias. Tais informações devem ter uma representação acessível ao programa, para que ele seja capaz de extrair as consequências necessárias.

O carácter universal da lógica (que foi ressaltado na argumentação da Seção 1.1) possibilitou que esta servisse de suporte a tais investigações, de maneira a fornecer tanto uma linguagem para representar fatos, quanto métodos para extrair conclusões por meio de demonstrações formais. Tais métodos podem ser programados em computadores digitais devido a resultados que, como o Teorema de Herbrand, garantem que a expressividade da lógica de primeira ordem é, de certa maneira, redutível à da máquina (devido à possibilidade de sentenças indecidíveis, não é totalmente apropriado dizer que a lógica é inteiramente redutível a procedimentos mecânicos). A programação lógica é uma maneira pela qual as diferentes áreas podem encontrar um acesso aos métodos da lógica formal.

Outra importante área de pesquisa, relacionada à representação do conhecimento e à IA, é a do raciocínio de senso comum, marcada por métodos de raciocínio não-monotônicos. Como pode ser visto na Seção 2.3, a programação lógica pode suportar *naturalmente* modos não-monotônicos de dedução, através de literais negados e sua interpretação pela *negação por falha*. Nessa perspectiva, deduz-se a negação de um fato sobre o qual não há informações suficientes para deduzi-lo. Além disso, conforme afirmado em [DP98, p.241], a maior parte dos formalismos não-monotônicos têm uma contrapartida no lado da programação lógica.¹ Raciocínios não-monotônicos possibilitam lidar com informações incompletas: uma vez que haja mais informações sobre um determinado fato, deixa-se de deduzir certas consequências e passa-se a deduzir outras. Mas as informações, normalmente, não são apenas incompletas, mas também contraditórias. A introdução de um novo tipo de negação, a chamada negação *explícita* (\neg), está relacionada com esta propriedade: ao admitirem-se fatos explicitamente negados, não se pode simplesmente deixar de deduzi-los quando aparecem contradições. Informações introduzidas explicitamente não são facilmente descartadas, devido justamente à presença de evidência concretas, *explícitas*, quanto à sua pertinência. Logo, tal negação tem carácter monotônico (uma vez deduzida, não se pode deixar de derivá-

¹Como exemplos, os autores citam a Lógica *Default* [Rei80] e as lógicas auto-epistêmicas [Moo87], [Moo85].

la). Dessa maneira surgiu a programação lógica estendida, que admite os dois tipos de negação nos corpos das regras dos programas e apenas a negação explícita na cabeça das regras. Se for designada por “ \neg ” a negação *explícita* (*monotônica*) e por “ $\acute{\neg}$ ”, a *default* (*não-monotônica*), em geral, uma regra de um programa lógico estendido tem o seguinte formato (em que os A_i representam fórmulas atômicas e os L_i , literais de negação explícita):

$$L_0 \leftarrow [\acute{\neg}]L_1, \dots, [\acute{\neg}]L_n \text{ ou, equivalentemente:}$$

$$[\neg]A_0 \leftarrow [\acute{\neg}][\neg]A_1, \dots, [\acute{\neg}][\neg]A_n .$$

Devido à monotonicidade das deduções envolvendo a negação explícita, qualquer abordagem à programação lógica estendida deve estar apta a lidar com a presença de contradições. Em [DP98, p.242], os autores afirmam que existem três abordagens para se lidar com informações contraditórias: a *explosiva*, a de *revisão de crenças* e a *paraconsistente*. Aqui, não serão abordadas as duas primeiras. Mas, apenas para que fique claro, a primeira é a abordagem clássica: todas as fórmulas são dedutíveis de uma contradição e a segunda possibilita que o programa (ou o banco de dados) seja revisado de forma a obter novamente a consistência. Tais abordagens parecem se esquivar a uma característica inerente a este tipo de programação: ela é necessariamente paraconsistente. Portanto, a abordagem paraconsistente é a que nos interessa, por demandar que se admitam contradições sem trivialização e que leve-se isto em conta durante as deduções. Para os programas lógicos estendidos, existem inúmeras semânticas dentro da perspectiva paraconsistente e esta variedade está relacionada à presença da negação não-monotônica, que mantém um comportamento explosivo. Em [DP98], os autores fazem um excelente apanhado delas, dividindo-as em dois grandes grupos: as baseadas nas *Semânticas bem Fundadas* (*Well Founded Semantics*) e nas *Semânticas de Conjunto Resposta* (*Answer Set Semantics*). Todas, no entanto, coincidem na caracterização que fazem do fragmento dos *Programas Lógicos Estendidos Definidos*. Este tipo de programa lógico faz uso apenas da negação explícita (de caráter monotônico) e será o assunto do restante da seção.

Definição 3.1. Os *Programas Lógicos Estendidos Definidos* são aqueles que permitem negações explícitas em qualquer parte de suas regras.

■

Logo, em geral, uma regra de um programa lógico estendido definido tem o seguinte formato:

$$L_0 \leftarrow L_1, \dots, L_n ,$$

ou:

$$[\neg]A_0 \leftarrow [\neg]A_1, \dots, [\neg]A_n .$$

As deduções aqui tratadas, a partir de tais programas, serão de conjunções de literais formados pela negação explícita. Logo, não serão abordados métodos de derivação não-monotônicos.

Para ilustrar a definição acima, considere-se o seguinte exemplo simples, retirado de [DP98], de *taxonomia do senso comum*:

Exemplo 3.2. Considere-se as seguintes regras para identificar pássaros e mamíferos:

- Animais ovíparos, de sangue quente, que têm bico são pássaros;
- Animais com pelos, de sangue quente são mamíferos;
- Pássaros não são mamíferos e vice-versa;
- Pássaros voam;
- Mamíferos aleitam a sua prole.

Esta base de informações pode ser codificada no seguinte programa lógico estendido definido:

$$\begin{aligned} \text{passaro}(X) &\leftarrow \text{bico}(X), \text{ sangue_quente}(X), \text{ oviparo}(X) . \\ \text{mamifero}(X) &\leftarrow \text{pelos}(X), \text{ sangue_quente}(X) . \\ \neg \text{mamifero}(X) &\leftarrow \text{passaro}(X) . \\ \neg \text{passaro}(X) &\leftarrow \text{mamifero}(X) . \\ \text{voa}(X) &\leftarrow \text{passaro}(X) . \\ \text{aleita}(X) &\leftarrow \text{mamifero}(X) . \end{aligned}$$

Se forem adicionadas as informações relevantes para os gatos (g) e para os patos (p):

$$\text{pelos}(g). \text{ sangue_quente}(g). \text{ bico}(p). \text{ sangue_quente}(p). \text{ oviparo}(p).$$

seriam obtidos, corretamente, que gatos são mamíferos: $\text{mamifero}(g)$ e não pássaros: $\neg \text{passaro}(g)$ e que patos são pássaros: $\text{passaros}(p)$ e não mamíferos: $\neg \text{mamiferos}(p)$. Além de que patos voam e gatos aleitam suas crias.

Agora, para o caso de ornitorrincos (o), se forem adicionadas as informações relevantes:

$$\text{pelos}(o). \text{ sangue_quente}(o). \text{ bico}(o). \text{ oviparo}(o).$$

seriam obtidas as contradições: $mamifero(o)$, $\neg mamifero(o)$, $passaro(o)$ e $\neg passaro(o)$, tendo em vista o caráter monotônico dos programas lógicos em questão. Seria deduzido, também, que ornitorrincos voam e aleitam suas crias. Desta forma, é fundamental uma interpretação paraconsistente para que não sejam deduzidas, a partir destas contradições, fatos arbitrários sobre gatos e patos.

A importância do fragmento monotônico descrito na Definição 3.1 fica evidente pela introdução, em [DP98, p.248], de uma semântica para tal fragmento que é o “denominador comum de quase todas as outras semânticas examinadas” em seu *survey*. Adiante no artigo, os autores afirmam que ela está também relacionada aos *programas generalizados de Horn* ([BS89]) e aos *programas com negação forte* ([Wag93], [Wag94]) que foram, ainda segundo os autores do *survey*, os principais responsáveis pela introdução de formas de raciocínio paraconsistente na área de programação lógica ([DP98, 242-243]). Tais abordagens seminais não serão aqui descritas, dado que são essencialmente o mesmo que a semântica em questão. Isso ficará claro com o Teorema 3.5.

A semântica em questão [DP98, Definição 3, p.249] está definida para os programas lógicos estendidos definidos (Definição 3.1) e se baseia na semântica usual para programas lógicos (Definição 2.30). Ela é construída fazendo uso de pontos fixos para operadores de consequência imediata associados aos programas (Definição 2.35). Para encontrar um modelo para um programa estendido definido E , os autores o supõem *totalmente instanciado*. As operações realizadas para se chegar à semântica que proprõem assumem, inicialmente, que o programa não possui variáveis, pois todas devem estar instanciadas por todos os possíveis termos fechados da linguagem. Esta é uma das partes problemáticas. Fazendo isto, estão assumindo *implicitamente* alguma forma de teorema de Herbrand para a lógica original, a lógica inerente ao fragmento da programação lógica estendida definida, necessariamente paraconsistente. Note-se que, mesmo que os programas não fossem totalmente instanciados para se chegar à tal semântica, é problemático depender que os resultados válidos para a lógica clássica sejam transferidos tacitamente à lógica paraconsistente.

Para a definição da semântica, o que se faz, essencialmente, é atribuir um predicado novo a cada predicado negado em um programa lógico estendido definido e, dessa maneira, transformar o programa original em um programa lógico *clássico*. A partir dele, seu conjunto de consequências é definido *a partir do programa transformado* como o menor ponto fixo do operador de consequência imediato associado. Passa-se, por último, à tradução reversa do conjunto de consequências computado a partir do operador clássico.

Definição 3.3. Seja E um programa lógico estendido definido. O modelo M_E do programa é obtido da seguinte forma:

1. Transforme o programa E em um programa positivo E^\neg renomeando as suas literais A e $\neg A$, respectivamente, por A^p e A^n .
2. Seja μ_{E^\neg} o modelo mínimo usual de E^\neg , associado ao operador T_{E^\neg} .
3. Então, para obter M_E , reverta a transformação do primeiro passo, transformando $A^p \in \mu_{E^\neg}$ ($A^n \in \mu_{E^\neg}$) em $A \in M_E$ ($\neg A \in M_E$).

■

Note-se que, por estar definida por meio dos operadores clássicos, tal semântica é monotônica. Na sequência, será dado um exemplo adaptado do mesmo artigo para tal semântica.

Exemplo 3.4. Seja E o seguinte programa lógico estendido definido:

$$\begin{aligned} \text{voa}(X) &\leftarrow \text{passaro}(X). \\ \neg \text{voa}(X) &\leftarrow \text{pinguim}(X). \\ \text{passaro}(X) &\leftarrow \text{pinguim}(X). \end{aligned}$$

Considere-se o caso em que os fatos $\text{passaro}(\text{tweety})$ e $\text{pinguim}(\text{fred})$ são adicionados às regras acima. M_E seria, então:

$$\begin{aligned} &\{\text{voa}(\text{tweety}), \text{passaro}(\text{tweety})\} \cup \\ &\{\text{voa}(\text{fred}), \neg \text{voa}(\text{fred}), \text{passaro}(\text{fred}), \text{pinguim}(\text{fred})\} \end{aligned}$$

Note-se que, na correspondente teoria clássica, haveria a trivialização, devido à presença dos fatos contraditórios $\text{voa}(\text{fred})$ e $\neg \text{voa}(\text{fred})$. Se um conjunto de fatos for adicionado ao programa, o resultado será um modelo $M_{E'}$ contendo o descrito acima, devido a monotonicidade da semântica descrita na Definição 3.3.

Um dos principais resultados enunciados em [DP98] é o seguinte teorema que caracteriza completamente as semânticas mais importantes descritas por eles como isomorfias a M_E .

Teorema 3.5. [DP98, Teorema 8, p.251] *Seja E um programa lógico estendido definido. Diferenças sintáticas à parte, todas as semânticas para este tipo de programa descritas em [BS89], [PR91], [Sak92], [Pea93], [ADP95], [AP96], [SI95], [Wag94] e [Wag93] são isomorfias a M_E .*

A demonstração deste teorema é feita ao longo de todo o artigo, durante a apresentação de cada semântica. Por isso, não será possível reproduzi-la aqui. Mas toma-se como base este resultado para o que segue.

3.2 A Necessidade de um Teorema de Herbrand

O teorema de Herbrand possibilita reduzir a derivabilidade de uma lógica de primeira ordem ao seu caso proposicional. Isto é conseguido pela transformação de deduções de fórmula gerais, com quantificadores, em deduções de certas fórmulas equivalentes, sem quantificadores e fechadas com os termos da linguagem original. Deste modo, possibilita lidar com universos potencialmente infinitos de uma maneira finitária, como é o caso quando há símbolos de função e variáveis nos programas. Como visto acima, a semântica M_E para programas estendidos definidos pressupõe programas fechados para todos os possíveis termos da linguagem. Tal pressuposição parece problemática, uma vez que se baseia no teorema de Herbrand da lógica clássica, para a qual efetua a tradução do programa original. Uma melhor saída é encontrar um teorema de Herbrand para alguma lógica paraconsistente que sirva de base para os programas lógicos estendidos definidos, evitando assim tal tradução e explicitando melhor os fundamentos lógicos da programação estendida definida.

Mas a utilidade de tal teorema não está restrita apenas a uma justificação dos procedimentos já realizados normalmente na área da programação lógica paraconsistente. Devido à sua generalidade, abre a possibilidade de incluir na sintaxe dos programas lógicos paraconsistentes fórmulas mais complexas, com os conectivos de consistência, por exemplo. Além do que, se for estendido para outras **LFI's**, servirá para estudar o comportamento de programas lógicos mais gerais sobre a influência de alguns esquemas axiomáticos, como, por exemplo, o da propagação da consistência ($\circ A \wedge \circ B \rightarrow \circ(A \wedge B)$). Evidentemente, um teorema de Herbrand por si só não nos fornece uma regra de resolução eficaz: no caso clássico, levou-se anos até que métodos de dedução automática baseados neste teorema dessem frutos. Mas, mesmo assim, ele é um ponto de partida, antes de qualquer especialização que diga qual a resolução correta ou mesmo qual a forma lógica precisa das regras ou das consultas, justamente porque pode ser estabelecido para fórmulas quaisquer e não está preso a alguma forma normal.

Além de suas aplicações na programação lógica, ele é um indicativo de que *é possível* um procedimento de demonstração automática, embora sua implementação imediata não seja muito eficiente. Logo, abre as portas para do campo de pesquisa em dedução automática para as **LFI's** de primeira ordem.

3.3 Lógicas da Inconsistência Formal

Paraconsistência, isto é, o estudo de sistemas formais em que contradições não necessariamente trivializam, é um dos grandes desafios da lógica contemporânea. A

paraconsistência permite diferenciar entre inconsistência, contradição e trivialização. Além do evidente interesse do ponto de vista da filosofia, em particular sobre a questão da existência ou não de contradições ‘reais’, em anos recentes tem aumentado o interesse em paraconsistência por parte da comunidade de ciencias da computação, por causa das suas potenciais aplicações em Inteligência Artificial e no desenvolvimento de bases de dados inconsistentes.

Assim, o fenômeno da paraconsistência transcendeu as fronteiras da lógica filosófica para adentrar-se na área das aplicações tecnológicas. Como não podia ser de outra maneira, o desenvolvimento de sistemas de programação lógica paraconsistentes foi um passo natural na evolução da paraconsistência.

As lógicas paraconsistentes foram inicialmente propostas por Jaśkowski [Jas48] e Nelson [Nel59]. Existem diferentes abordagens à paraconsistência na literatura, destacando-se no cenário internacional a chamada ‘escola brasileira’, iniciada por Newton da Costa a partir da conhecida hierarquia de lógicas C_n (cf. [dC63]). Em [CM02] foi introduzida uma generalização dos sistemas de da Costa, as chamadas *Lógicas da Inconsistência Formal* (**LFI**'s, usando sua sigla em inglês). Este trabalho foi aprofundado em [CCM07] e [Mar05]. Nas **LFI**'s as noções de consistência e inconsistência são internalizadas na linguagem através do uso de conectivos \bullet e \circ , em que a fórmula $\bullet\alpha$ denota que a fórmula α é inconsistente, enquanto que $\circ\alpha$ denota que α é consistente. As propriedades básicas das **LFI**'s são, em termos formais, as seguintes:

- (1) $\alpha, \neg\alpha \not\vdash \beta$ mas $\alpha, \neg\alpha, \circ\alpha \vdash \beta$
- (2) $\alpha, \neg\alpha, \vdash \bullet\alpha$

As propriedades em (1) estabelecem que uma **LFI** é paraconsistente, no sentido que a partir de uma contradição não inferimos forçosamente qualquer outra fórmula; porém, a partir de uma contradição mais a informação adicional de que a fórmula contraditória é *consistente*, deriva-se qualquer outra fórmula. Por sua vez, a propriedade (2) estabelece que uma contradição implica que a fórmula é inconsistente; a recíproca não vale em todas as **LFI**'s, mas apenas naquelas em que esta propriedade é estipulada explicitamente. Assim, as **LFI**'s separam claramente as noções de contradição, inconsistência e trivialidade. As **LFI**'s generalizam os **C**-sistemas de da Costa, em que uma fórmula da linguagem (e não um conectivo primitivo) representa a consistência das fórmulas. Assim, por exemplo, no sistema C_1 a consistência da fórmula α é dada pela fórmula $\neg(\alpha \wedge \neg\alpha)$, denotada originalmente como α° .

Uma característica importante das **LFI**'s é que, em geral, não há interdefinibilidade dos conectivos, e muitas propriedades da implicação material não são mais válidas: em particular, $\alpha \rightarrow \beta$ e $\neg\alpha \vee \beta$ não são equivalentes, assim como as fórmu-

las $\alpha \rightarrow \beta$ e $\neg\beta \rightarrow \neg\alpha$ também não o são. Por outro lado, os silogismos disjuntivos

$$\frac{\alpha \vee \gamma \quad \beta \vee \neg\gamma}{\alpha \vee \beta}$$

não são mais válidos, dificultando então a definição de um análogo à regra de resolução de PROLOG no contexto das **LFI**'s: certas hipóteses de consistência (por exemplo, $\circ\gamma$) deveriam ser acrescentadas para validar a inferência acima. Voltaremos sobre este importante ponto no último capítulo desta dissertação.

O resto deste capítulo é dedicado a apresentar com um certo detalhe duas **LFI**'s importantes, **mbC** e **mCi**, introduzidas em [CM02], e a suas versões de primeira ordem, **QmbC** e **QmCi**, respectivamente, introduzidas em [Pod08]. Com relação a estas últimas, apresentaremos uma demonstração original da prova de completude, acrescentando alguns detalhes que ficaram obscuros, ao nosso entender, na prova original de completude apresentada em [Pod08].

Para o leitor interessado, maiores detalhes sobre as **LFI**'s podem ser encontrados em [CCM07].

3.3.1 As **LFI**'s básicas: **mbC** e **mCi**

Nesta seção apresentaremos brevemente duas **LFI**'s básicas, **mbC** e **mCi**, que serão a base proposicional da lógica paraconsistente quantificada proposta como fundamento da programação lógica paraconsistente.

Como foi mencionado anteriormente, a ideia principal das **LFI**'s é eliminar o chamado *princípio de explosão*, que estabelece que

$$\alpha, \neg\alpha \vdash \beta$$

ou, equivalentemente,

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

para toda fórmula α e β . Uma lógica em que as propriedades acima *não valem* em geral é dita *paraconsistente* (com relação à negação \neg). Assim, as **LFI**'s são lógicas paraconsistentes nas quais exigimos que uma contradição *mais* a informação de que a fórmula contraditória é consistente (ou 'bem comportada', ou tem 'comportamento clássico') garanta a explosão lógica:

$$\alpha, \neg\alpha \not\vdash \beta$$

para algumas α e β , porém

$$\circ\alpha, \alpha, \neg\alpha \vdash \beta$$

para toda α e β . Equivalentemente, substituímos o axioma da explosão

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

pela versão mais fraca

$$(*) \quad (\circ\alpha \rightarrow (\alpha \rightarrow (\neg\alpha \rightarrow \beta)))$$

Aqui, introduzimos um novo conectivo unário de consistência \circ tal que $\circ\alpha$ representa a afirmação da consistência de α . Uma lógica com uma negação paraconsistente \neg e com um operador de consistência \circ satisfazendo o axioma de explosão enfraquecido (*) é uma lógica da Inconsistência formal (**LFI**).

Dado que, salvo os conectivos de consistência \circ e de negação paraconsistente \neg , podemos assumir que os outros conectivos têm um comportamento clássico, a menor **LFI** proposicional é a lógica **mbC**, introduzida em [CM02] da seguinte maneira:

Definição 3.6 (Lógica **mbC**). Seja \mathcal{V} um conjunto enumerável de variáveis proposicionais. A lógica proposicional **mbC** (gerada por \mathcal{V}) é a lógica definida sobre o conjunto *For* de fórmulas gerado pelos conectivos $\{\neg, \circ, \wedge, \vee, \rightarrow\}$ a partir de \mathcal{V} como segue:

Axiomas positivos:

$$\begin{array}{ll} \alpha \rightarrow (\beta \rightarrow \alpha) & \text{(Mon)} \\ (\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow (\alpha \rightarrow \gamma)) & \text{(Trans)} \\ \alpha \rightarrow (\beta \rightarrow (\alpha \wedge \beta)) & \text{(E0)} \\ (\alpha \wedge \beta) \rightarrow \alpha & \text{(E1)} \\ (\alpha \wedge \beta) \rightarrow \beta & \text{(E2)} \\ \alpha \rightarrow (\alpha \vee \beta) & \text{(Ou0)} \\ \beta \rightarrow (\alpha \vee \beta) & \text{(Ou1)} \\ (\alpha \rightarrow \gamma) \rightarrow ((\beta \rightarrow \gamma) \rightarrow ((\alpha \vee \beta) \rightarrow \gamma)) & \text{(Ou2)} \\ \alpha \vee (\alpha \rightarrow \beta) & \text{(OuI)} \end{array}$$

Axioma Clássico

$$\alpha \vee \neg\alpha \quad \text{(TND)}$$

Axioma Paraconsistente:

$$\circ\alpha \rightarrow (\alpha \rightarrow (\neg\alpha \rightarrow \beta)) \quad \text{(Exp)}$$

Regra de inferência

– Modus Ponens:

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta} \quad (\text{MP})$$

■

A lógica **mbC** satisfaz interessantes propriedades, permitindo distinguir entre contradição e não-consistência (ou inconsistência). Por exemplo (ver [CCM07]):

Teorema 3.7. *Em **mbC** vale o seguinte:*

- (i) $\alpha, \neg\alpha \vdash_{\text{mbC}} \neg\circ\alpha$
- (ii) $\alpha \wedge \neg\alpha \vdash_{\text{mbC}} \neg\circ\alpha$
- (iii) $\circ\alpha \vdash_{\text{mbC}} \neg(\alpha \wedge \neg\alpha)$

*As recíprocas dessas regras não valem em **mbC**.*

Pelo fato de ser um subsistema da lógica clássica (se consideramos o fragmento que não utiliza \circ), a lógica **mbC** perde propriedades lógicas com relação à clássica. Algumas delas tem a ver com a contraposição. De fato, pode ser provado que algumas das formas da contraposição não podem ser recuperadas em nenhuma extensão paraconsistente de **mbC** sendo, portanto, intrínsecas à abordagem à paraconsistência baseada nos axiomas de **mbC**:

Teorema 3.8. *Seja **mbC'** o sistema obtido de **mbC** pela remoção do Axioma Paraconsistente (**Exp**), e seja **L** uma extensão de **mbC'**.*

- (i) *Se $\neg\alpha \rightarrow \neg\beta \vdash_{\mathbf{L}} \beta \rightarrow \alpha$ é válido em **L** então **L** não é mais paraconsistente com relação a \neg .*
- (ii) *Se $\neg\alpha \rightarrow \beta \vdash_{\mathbf{L}} \neg\beta \rightarrow \alpha$ é válido em **L** então **L** não é mais paraconsistente com relação a \neg .*

Devemos observar que este resultado melhora o Theorem 38, [CCM07].

Demonstração. (i) Na lógica **L** temos que $\neg\beta \vdash_{\mathbf{L}} \neg\alpha \rightarrow \neg\beta$, em virtude de (**Mon**) e **MP**. Dado que $\neg\alpha \rightarrow \neg\beta \vdash_{\mathbf{L}} \beta \rightarrow \alpha$ então $\neg\beta \vdash_{\mathbf{L}} \beta \rightarrow \alpha$. Assim, $\neg\beta, \beta \vdash_{\mathbf{L}} \alpha$ para todo α, β , por **MP**. O item (ii) é provado analogamente. □

A lógica **mbC**, assim como todas as extensões estudadas em [CCM07], apresenta uma particularidade que traz não poucas dificuldades técnicas, e que impossibilita o uso de técnicas lógicas usuais, além de contrariar, de certa maneira, nossa intuição: os conectivos não-clássicos, nomeadamente a negação paraconsistente \neg

e o operador de consistência \circ , não preservam equivalências lógicas, isto é: denotando por $\alpha \dashv\vdash_{\mathbf{mbC}} \beta$ a interderivabilidade entre α e β na lógica \mathbf{mbC} , temos que

$$\alpha \dashv\vdash_{\mathbf{mbC}} \beta \text{ não implica } \neg\alpha \dashv\vdash_{\mathbf{mbC}} \neg\beta$$

e, analogamente,

$$\alpha \dashv\vdash_{\mathbf{mbC}} \beta \text{ não implica } \circ\alpha \dashv\vdash_{\mathbf{mbC}} \circ\beta$$

Por exemplo, temos o seguinte resultado (ver [CCM07]):

Teorema 3.9. *Em \mathbf{mbC} :*

- (i) $(\alpha \wedge \beta) \dashv\vdash_{\mathbf{mbC}} (\beta \wedge \alpha)$ é válido,
mas $\neg(\alpha \wedge \beta) \dashv\vdash_{\mathbf{mbC}} \neg(\beta \wedge \alpha)$ não vale.
- (ii) $(\alpha \vee \beta) \dashv\vdash_{\mathbf{mbC}} (\beta \vee \alpha)$ é válido,
mas $\neg(\alpha \vee \beta) \dashv\vdash_{\mathbf{mbC}} \neg(\beta \vee \alpha)$ não vale.
- (iii) $(\alpha \wedge \neg\alpha) \dashv\vdash_{\mathbf{mbC}} (\neg\alpha \wedge \alpha)$ é válido,
mas $\neg(\alpha \wedge \neg\alpha) \dashv\vdash_{\mathbf{mbC}} \neg(\neg\alpha \wedge \alpha)$ não vale.
- (iv) $(\alpha \vee \neg\alpha) \dashv\vdash_{\mathbf{mbC}} (\beta \vee \neg\beta)$ é válido,
mas $\neg(\alpha \vee \neg\alpha) \dashv\vdash_{\mathbf{mbC}} \neg(\beta \vee \neg\beta)$ não vale.

Do ponto de vista semântico, isto se reflete da seguinte maneira: os conectivos binários (implicação \rightarrow , conjunção \wedge e disjunção \vee) são vero-funcionais. Isto é, o valor de verdade das fórmulas compostas por qualquer um destes conectivos é obtido funcionalmente a partir do valor de verdade das componentes. Por outro lado, o valor de verdade de uma fórmula α nem sempre determina o valor de verdade de $\neg\alpha$ e de $\circ\alpha$, sendo que estes valores podem ser independentes, como veremos depois. Isto significa que estes conectivos não-clássicos não são vero-funcionais. Existe, por outro lado, uma relação funcional entre os valores de verdade de α , $\neg\alpha$ e $\circ\alpha$, o que permite obter um dos valores a partir dos outros dois (mas isto só vai acontecer em extensões de \mathbf{mbC}).

É importante observar, porém, que em \mathbf{mbC} é possível definir uma negação forte, isto é, clássica. Considere, para toda β , a fórmula $\perp_\beta = (\beta \wedge \neg\beta \wedge \circ\beta)$. É fácil provar que \perp_β define o que em lógica se chama de fórmula *bottom* (ou trivializante), isto é: fixada β , então $\perp_\beta \vdash_{\mathbf{mbC}} \gamma$ para toda γ . A partir daí podemos definir uma negação \sim_β em \mathbf{mbC} como segue: $\sim_\beta\alpha = \alpha \rightarrow \perp_\beta$. Pode ser provado facilmente (ver [CCM07]) que \sim_β satisfaz as propriedades de uma negação clássica. Por exemplo (a seguir omitiremos, por simplicidade de notação, o índice β):

Teorema 3.10. *Em \mathbf{mbC} vale o seguinte:*

- (i) $\alpha, \sim\alpha \vdash_{\mathbf{mbC}} \gamma$ e então $\vdash_{\mathbf{mbC}} \sim\alpha \rightarrow (\alpha \rightarrow \gamma)$;
- (ii) $\vdash_{\mathbf{mbC}} \alpha \vee \sim\alpha$;
- (iii) $\vdash_{\mathbf{mbC}} \alpha \rightarrow \sim\sim\alpha$ e $\vdash_{\mathbf{mbC}} \sim\sim\alpha \rightarrow \alpha$;
- (iv) se $(\Gamma, \alpha \vdash_{\mathbf{mbC}} \gamma)$ e $(\Delta, \sim\alpha \vdash_{\mathbf{mbC}} \gamma)$ então $(\Gamma, \Delta \vdash_{\mathbf{mbC}} \gamma)$;
- (v) $\vdash_{\mathbf{mbC}} (\alpha \rightarrow \beta) \rightarrow ((\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \gamma))$;
- (vi) $\alpha \vdash_{\mathbf{mbC}} \beta$ implica $\gamma \rightarrow \alpha \vdash_{\mathbf{mbC}} \gamma \rightarrow \beta$, para toda γ ;
- (vii) $\vdash \sim(\alpha \rightarrow \beta) \rightarrow (\alpha \wedge \sim\beta)$.

Demonstração. Provaremos apenas os itens (v), (vi) e (vii), dado que a prova dos outros pode se consultada em [CCM07].

(v) Claramente $(\alpha \rightarrow \beta), (\beta \rightarrow \gamma), \alpha \vdash_{\mathbf{mbC}} \gamma$ vale em \mathbf{mbC} e então o resultado vale pelo Teorema da Dedução.

(vi) De $\gamma \rightarrow \alpha, \gamma$ segue α (por **MP**) e então segue β (por hipótese), donde segue $\gamma \rightarrow \beta$, por causa do axioma $(\beta \rightarrow (\gamma \rightarrow \beta))$ e **MP**. Isto é, $\gamma \rightarrow \alpha, \gamma \vdash_{\mathbf{mbC}} \gamma \rightarrow \beta$. Por outro lado, $\sim\gamma \vdash_{\mathbf{mbC}} \gamma \rightarrow \beta$. Dai obtemos que $\gamma \rightarrow \alpha \vdash_{\mathbf{mbC}} \gamma \rightarrow \beta$, pelo item (iv).

(vii) Dado que $\vdash_{\mathbf{mbC}} \sim\alpha \rightarrow (\alpha \rightarrow \beta)$ então de $\sim(\alpha \rightarrow \beta), \sim\alpha$ obtemos $\sim(\alpha \rightarrow \beta)$ e $(\alpha \rightarrow \beta)$, e daqui deduzimos α , pelo item (i). Mas $\sim(\alpha \rightarrow \beta), \alpha \vdash_{\mathbf{mbC}} \alpha$. Logo, pelo item (iv) segue que $\sim(\alpha \rightarrow \beta) \vdash_{\mathbf{mbC}} \alpha$. Por outro lado, dado que $\vdash_{\mathbf{mbC}} \beta \rightarrow (\alpha \rightarrow \beta)$ então de $\sim(\alpha \rightarrow \beta), \beta$ segue $\sim(\alpha \rightarrow \beta)$ e $(\alpha \rightarrow \beta)$, e daqui deduzimos $\sim\beta$, pelo item (i). Mas $\sim(\alpha \rightarrow \beta), \sim\beta \vdash_{\mathbf{mbC}} \sim\beta$. Assim, pelo item (iv) segue que $\sim(\alpha \rightarrow \beta) \vdash_{\mathbf{mbC}} \sim\beta$. Portanto $\sim(\alpha \rightarrow \beta) \vdash_{\mathbf{mbC}} (\alpha \wedge \sim\beta)$. O resultado segue pelo Teorema da Dedução.

□

A lógica \mathbf{mbC} tem como primitivo um operador de consistência \circ . Resultaria bastante natural definir a inconsistência de uma fórmula a partir da negação da sua consistência. Noutras palavras, se denotamos por $\bullet\alpha$ a *inconsistência* de α , então esperaríamos definir $\bullet\alpha$ através da fórmula $\neg\circ\alpha$. Porém, isto não produz, como esperado, uma dualidade entre \circ e \bullet em \mathbf{mbC} (ver [CCM07]), e é então necessário usar a negação forte \sim de \mathbf{mbC} para definir a inconsistência: $\bullet\alpha = \sim\circ\alpha$. A definição de um operador de inconsistência justifica o nome de ‘lógicas da inconsistência formal’.

O passo natural seguinte é incrementar o poder lógico de \mathbf{mbC} para permitir definir a inconsistência da maneira esperada, nomeadamente $\bullet\alpha = \neg\circ\alpha$, usando a

negação paraconsistente em vez da negação forte, e ainda obtendo a interdefinibilidade entre \circ e \bullet . Para levar adiante esta ideia é introduzido o sistema **mCi**, que estende **mbC** de maneira a permitir falar da inconsistência como operador primitivo dual do operador de consistência.

Definição 3.11 (Lógica **mCi**). A lógica proposicional **mCi** é obtida de **mbC** pelo acréscimo dos seguintes axiomas:

Axiomas Paraconsistentes Fortes:

$$\neg \circ \alpha \rightarrow (\alpha \wedge \neg \alpha) \quad (\text{Inc})$$

$$\circ \neg^n \circ \alpha \quad (\text{Con})$$

■

No axioma **(Con)** acima temos que $n \geq 0$, uma vez que definimos $\neg^0 \alpha = \alpha$ e $\neg^{n+1} \alpha = \neg \neg^n \alpha$. Assim, no caso $n = 0$ temos que $\vdash_{\mathbf{mCi}} \circ \circ \alpha$. Alguns resultados interessantes em **mCi** são os seguintes (ver [CCM07]):

Teorema 3.12. *Em **mCi** vale:*

$$(i) \neg \circ \alpha \vdash_{\mathbf{mCi}} (\alpha \wedge \neg \alpha) \text{ e então } \neg \circ \alpha \nvdash_{\mathbf{mCi}} (\alpha \wedge \neg \alpha),$$

mas o seguinte não vale:

$$(ii) \neg(\alpha \wedge \neg \alpha) \vdash_{\mathbf{mCi}} \circ \alpha;$$

$$(iii) \neg(\neg \alpha \wedge \alpha) \vdash_{\mathbf{mCi}} \circ \alpha.$$

Outras propriedades úteis de **mCi** são as seguintes:

Teorema 3.13. *Em **mCi** vale o seguinte:*

$$(i) \circ \alpha \nvdash_{\mathbf{mCi}} \neg \neg \circ \alpha;$$

$$(ii) \circ \alpha, \neg \circ \alpha \vdash_{\mathbf{mCi}} \beta;$$

$$(iii) (\Gamma, \beta \vdash_{\mathbf{mCi}} \circ \alpha) \text{ e } (\Delta, \beta \vdash_{\mathbf{mCi}} \neg \circ \alpha) \text{ implica } (\Gamma, \Delta \vdash_{\mathbf{mCi}} \neg \beta).$$

É importante observar que **mCi** os conectivos unários \neg e \circ também não preservam equivalências lógicas. Porém, pode ser provado que, se numa extensão de **mCi** a negação preserva equivalências lógicas, então o operador de consistência também o fará (ver [CCM07]).

Tanto a lógica **mbC** quanto a **mCi** possuem uma semântica de bivalorações, isto é, funções de verdade (não vero-funcionais) que atribuem, a cada sentença da linguagem, um valor de verdade 1 (verdadeiro) ou 0 (falso). É possível definir outras semânticas, como a semântica de traduções possíveis (ver [CCM07]) e a semântica de matrizes não determinísticas (ver [AZ06]), o que produz um método de decisão para estas lógicas. Porém, a diferença da abordagem semântica às **LFI**'s de primeira ordem apresentada em [AZ06], basearemos a semântica das **LFI**'s quantificadas em bivalorações paraconsistentes, seguindo a perspectiva de [Pod08].

3.3.2 LFI's quantificadas: os sistemas QmbC e QmCi

Uma vez que contamos com uma base proposicional paraconsistente sólida, os sistemas **mbC** e **mCi**, definiremos agora sua versão quantificada, como etapa natural seguinte do nosso projeto. Como mencionado anteriormente, adaptaremos a abordagem de [Pod08] através de algumas modificações e acréscimos.

O primeiro passo é definir linguagens de primeira ordem. As definições são as usuais mas, para deixar o texto auto-contido, descreveremos rapidamente os (algo aborrecidos, porém inevitáveis) detalhes técnicos das linguagens quantificadas, também chamadas de linguagens de primeira ordem.

Definição 3.14 (Assinatura de Primeira Ordem). Considere o conjunto de conectivos $\{\neg, \circ, \wedge, \vee, \rightarrow\}$ da lógica **mbC** e **mCi** (lembre da Definição 3.6), fixe um conjunto enumerável Var de variáveis individuais e novos símbolos \forall (quantificador universal) e \exists (quantificador existencial). Uma assinatura Σ para **LFI's** de primeira ordem consiste de um par $\Sigma = (\mathcal{F}, \mathcal{P})$ de famílias enumeráveis de conjuntos dois a dois disjuntos, $\mathcal{F} = \{F^n \mid 0 \leq n < \omega\}$ e $\mathcal{P} = \{P^n \mid 0 \leq n < \omega\}$.

■

Dada uma assinatura para **LFI's** de primeira ordem Σ , os elementos de F^n são chamados de símbolos de função n -ários, enquanto que os elementos de P^n são os símbolos de predicado n -ários. Adicionalmente, os elementos de F^0 são chamados de constantes, enquanto que os elementos de P^0 são constantes lógicas. Os conjuntos dos símbolos de funções e dos símbolos de predicados da assinatura são dados por

$$Func_{\Sigma} = \bigcup \{F^n \mid 0 \leq n < \omega\}$$

$$Pred_{\Sigma} = \bigcup \{P^n \mid 0 \leq n < \omega\}$$

respectivamente. Associado a uma assinatura temos o conjunto de seus *termos*, formado a partir das variáveis e símbolos funcionais.

Definição 3.15 (Termos de uma Assinatura). Dada uma assinatura Σ , o conjunto de termos de Σ , denotado por T_{Σ} , é o menor conjunto que satisfaz o seguinte:

- $Var \cup F^0 \subseteq T_{\Sigma}$;
- se $t_1, \dots, t_n \in T_{\Sigma}$ e $f \in F^n$ então $f(t_1, \dots, t_n) \in T_{\Sigma}$, para $n \geq 1$.

■

Na definição anterior, utilizamos vírgulas “,” e parênteses “(” e “)”. A rigor, estes símbolos não fazem parte da linguagem, mas são usados como recurso extralinguístico para facilitar nossa leitura.

Uma vez que definimos o conjunto de termos, podemos definir o conjunto de fórmulas.

Definição 3.16 (Fórmulas de uma Assinatura). Dada uma assinatura Σ , o conjunto de fórmulas de Σ , denotado por L_Σ , é o menor conjunto que satisfaz o seguinte:

- $P^0 \subseteq L_\Sigma$;
- se $t_1, \dots, t_n \in T_\Sigma$ e $P \in P^n$ então $P(t_1, \dots, t_n) \in L_\Sigma$, para $n \geq 1$;
- se $\alpha, \beta \in L_\Sigma$ então $\neg\alpha, \circ\alpha$ e $(\alpha\#\beta) \in L_\Sigma$, para $\# \in \{\wedge, \vee, \rightarrow\}$;
- se $\alpha \in L_\Sigma$ e $x \in Var$ então $\forall x\alpha, \exists x\alpha \in L_\Sigma$.

■

O conjunto At_Σ de fórmulas atômicas é dado por

$$At_\Sigma = P^0 \cup \{P(t_1, \dots, t_n) \mid t_1, \dots, t_n \in T_\Sigma, P \in P^n \text{ e } 1 \leq n < \omega\}$$

A noção de subfórmula é a usual. Para $Q = \forall, \exists$ dizemos que o *escopo* de uma ocorrência de Q da forma $Qx\alpha$ dentro de uma fórmula β é a fórmula α . Uma ocorrência de uma variável x numa fórmula α é *ligada* se x ocorre tanto numa expressão da forma Qx (para $Q = \forall, \exists$), ou se x ocorre no escopo de um quantificador Qx (para $Q = \forall, \exists$). Toda outra ocorrência de x em α é dita *livre*.

Definição 3.17 (Sentenças de uma Assinatura). Uma fórmula que não tem ocorrências livres de variáveis é chamada de *sentença*. Denotaremos por S_{L_Σ} o conjunto das sentenças sobre L_Σ .

■

Um termo t é livre para uma ocorrência livre de uma variável x numa fórmula α se, toda vez que x ocorre no escopo de um quantificador na variável y , então a variável y não ocorre em t . Um termo t é livre para uma variável x numa fórmula α se t é livre para cada ocorrência livre de x em α .

Obviamente x é livre para x em toda fórmula α ; por outro lado, se nenhuma variável que ocorre em t ocorre quantificada em α , então t é livre para x em α .

No exemplo abaixo, o termo t é livre para a variável x na fórmula α , mas não para a variável u :

$$t = f(y, g(z)); \quad \alpha = ((\forall x P(z) \vee Q(x, y)) \rightarrow \exists y R(y, u))$$

Observe que o escopo de \forall em α é $P(z)$ (e vemos assim que podem existir quantificações irrelevantes numa fórmula). Lendo de esquerda à direita, a primeira ocorrência de x é ligada, enquanto que a segunda é livre. A primeira ocorrência de y em α é livre, enquanto que a segunda e a terceira são ligadas. As únicas ocorrências das variáveis z e u em α são livres.

O conceito de substituição de variáveis livres por termos dentro de uma fórmula é fundamental nas linguagens de primeira ordem (e, em particular, na programação lógica e nas linguagens de programação em geral).

Notação 3.18 (Substituição). Denotaremos por $\varphi_x[t]$ a substituição de todas as ocorrências livres da variável x em φ pelo termo t (que deve ser livre para x em φ).

■

Definição 3.19 (Termo totalmente indicado). Dizemos que um termo t está *totalmente indicado* numa substituição $\varphi_x[t]$ se t não ocorre em φ . Caso contrário, dizemos que tal termo não está totalmente indicado nessa substituição.

■

Observação 3.20 (Negação Forte). Lembremos que, em **mbC**, a negação clássica \sim_β é definida pela abreviação $\sim_\beta \alpha = \alpha \rightarrow \perp_\beta$, em que $\perp_\beta = (\beta \wedge \neg \beta \wedge \circ \beta)$ é sempre uma fórmula trivializante, para qualquer fórmula β . No caso das linguagens de primeira ordem podemos repetir este processo, so que, por razões óbvias, a fórmula β deve ser escolhida como sendo uma sentença. A partir daqui usaremos a negação forte das **LFI**'s quantificadas (por exemplo, na Definição 3.21 e no Lema 3.33) tendo em mente que ela é definida para qualquer sentença β . Como foi feito para as **LFI**'s proposicionais, o índice β será omitido por simplicidade notacional (sendo que a sentença β escolhida é, de fato, irrelevante, e pode variar de negação forte para negação forte, ainda dentro da mesma fórmula).

■

Definiremos a seguir a versão quantificada de **mbC** acrescentando à versão originalmente apresentada em [Pod08] um axioma aparentemente faltante, o axioma **(PQ)**. Por outro lado, não identificaremos logicamente as fórmulas que variam apenas nas variáveis quantificadas e em quantificações irrelevantes, a diferença de [Pod08] (quem, por sua vez, inspirou-se em [AZ06]). Assim, as **LFI**'s propostas aqui diferem sutilmente das apresentadas na literatura.

Definição 3.21 (Lógica **QmbC**). Seja Σ uma assinatura para **LFI**'s de primeira ordem. A lógica **QmbC** é a versão quantificada de **mbC** definida sobre Σ da maneira seguinte:

Axiomas proposicionais:

Todas as instâncias em L_Σ dos axiomas de **mbC** **(mbC-Ax)**

Axioma ponte entre quantificadores (\sim denota uma negação forte):

$$\sim \forall x \alpha \rightarrow \exists x \sim \alpha \quad \textbf{(PQ)}$$

Axiomas Quantificacionais (t é um termo livre para x em α):

$$\alpha_x[t] \rightarrow \exists x \alpha \quad \textbf{(\exists-Ax)}$$

$$\forall x \alpha \rightarrow \alpha_x[t] \quad \textbf{(\forall-Ax)}$$

Regras de inferência

– Modus Ponens:

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta} \quad \textbf{(MP)}$$

– Introdução do Existencial (x não ocorre livre em β):

$$\frac{\alpha \rightarrow \beta}{\exists x \alpha \rightarrow \beta} \quad \textbf{(\exists-In)}$$

– Introdução do Universal (x não ocorre livre em α):

$$\frac{\alpha \rightarrow \beta}{\alpha \rightarrow \forall x \beta} \quad \textbf{(\forall-In)}$$

■

Definição 3.22 (Lógica **QmCi**). Seja Σ uma assinatura para **LFI**'s de primeira ordem. A lógica **QmCi** é a versão quantificada de **mCi** definida sobre Σ a partir de **QmbC**, acrescentando todas as instâncias sobre L_Σ dos Axiomas Paraconsistentes Fortes (**Inc**) e (**Con**).

■

Observação 3.23 (O axioma ponte (**PQ**)). Uma novidade das **LFI**'s quantificadas com relação às apresentadas por [Pod08] é a inclusão do axioma (**PQ**), que faz de ponte entre o quantificador existencial e o universal. A partir deste axioma podemos provar as seguintes propriedades fundamentais clássicas das **LFI**'s quantificadas:

$$\vdash (\forall x\alpha \rightarrow \beta) \rightarrow \exists x(\alpha \rightarrow \beta) \text{ se } x \text{ não ocorre livre em } \beta$$

$$\vdash (\alpha \rightarrow \exists x\beta) \rightarrow \exists x(\alpha \rightarrow \beta) \text{ se } x \text{ não ocorre livre em } \alpha$$

(ver Lema 3.33 abaixo). É fácil provar que se em **QmbC** e em **QmCi** substituirmos (**PQ**) pelo primeiro dos dois axiomas acima obteremos sistemas equivalentes.

É importante salientar que a partir de (**PQ**) obtemos a interdefinibilidade dos quantificadores existencial e universal, como acontece no caso clássico: $\forall x\alpha$ equivale a $\sim\exists x\sim\alpha$, enquanto que $\exists x\alpha$ equivale a $\sim\forall x\sim\alpha$. Assim, basta incluir um dos quantificadores (com seus axiomas e regras) na definição das **LFI**'s quantificadas, dado que o outro é definível por dualidade a partir da negação forte.

A interação entre os quantificadores estipulada pelo axioma (**PQ**) é uma propriedade fundamental que as **LFI**'s quantificadas devem satisfazer para ser completas com relação à semântica proposta. Aparentemente este axioma não pode ser demonstrado a partir dos outros axiomas e regras, daí a sua inclusão no nosso sistema. Fica como desafio determinar se (**PQ**) é ou não redundante em **QmbC** e **QmCi**.

■

Alguns Teoremas Úteis

Uma *teoria* numa dada lógica é, na nossa definição, um conjunto Γ de fórmulas na linguagem da lógica. Nesta seção serão enunciados resultados referentes a uma teoria Γ qualquer em **QmbC** ou **QmCi**. Logo, quando aparecer \vdash , estarão sendo representados tanto $\Gamma \vdash_{\mathbf{QmbC}}$ quanto $\Gamma \vdash_{\mathbf{QmCi}}$.

Teorema 3.24. *Para qualquer fórmula α , $\vdash \alpha \rightarrow \alpha$.*

Demonstração.

$\vdash \alpha \rightarrow \alpha \rightarrow \alpha$	Mon
$\vdash \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$	Mon
$\vdash (\alpha \rightarrow (\alpha \rightarrow \alpha)) \rightarrow (\alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$	Trans
$\vdash \alpha \rightarrow \alpha$	MP (2x)

□

Teorema 3.25 (Dedução por Casos). *Se conseguimos obter:*

$$\begin{array}{l} \Gamma \vdash \neg\alpha \rightarrow \beta \\ \Gamma \vdash \alpha \rightarrow \beta \end{array}$$

então, podemos deduzir também:

$$\Gamma \vdash \beta$$

Demonstração.

$\Gamma \vdash (\alpha \rightarrow \beta) \rightarrow ((\neg\alpha \rightarrow \beta) \rightarrow ((\alpha \vee \neg\alpha) \rightarrow \beta))$	(Axioma Ou2)
$\Gamma \vdash (\alpha \vee \neg\alpha) \rightarrow \beta$	(MP(2x) com Hipóteses)
$\Gamma \vdash \alpha \vee \neg\alpha$	(Axioma TND)
$\Gamma \vdash \beta$	(MP)

□

Teorema 3.26 (Generalização). *Podemos introduzir quantificações arbitrárias em quaisquer derivações:*

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall x \phi}$$

Demonstração.

$\Gamma \vdash \phi$	Hipótese
$\Gamma \vdash \phi \rightarrow (\neg\forall x \phi \rightarrow \phi)$	Mon
$\Gamma \vdash \neg\forall x \phi \rightarrow \phi$	MP
$\Gamma \vdash \neg\forall x \phi \rightarrow \forall x \phi$	\forall-In
$\Gamma \vdash \forall x \phi \rightarrow \forall x \phi$	Teorema 3.24
$\Gamma \vdash \forall x \phi$	Teorema 3.25

□

O passo seguinte é provar um dos mais importantes metateoremas que um sistema lógico com implicação pode ter: o Metateorema da Dedução, ou simplesmente o Teorema da Dedução. No caso proposicional sabemos que tanto **mbC** quanto **mCi** satisfazem esta propriedade. No caso das lógicas de primeira ordem, sabemos que este resultado não vale em geral, e que apenas versões restritas são válidas. Provaremos este resultado a seguir, adaptando a prova de [Men87] para a lógica clássica de predicados. Antes de provar este teorema, precisamos de algumas definições e resultados. A prova será feita para qualquer uma das duas **LFI**'s quantificadas em consideração.

Definição 3.27. Seja $d = \varphi_1, \dots, \varphi_n$ uma derivação numa **LFI** quantificada a partir de um conjunto Γ de fórmulas, e seja $\varphi \in \Gamma$. Dizemos que φ_i *depende* de φ em d se:

- $\varphi_i = \varphi$; ou
- φ_i é deduzida de φ_j e φ_k (com $j, k < i$) por (**MP**), sendo que φ_j ou φ_k dependem de φ em d ; ou
- φ_i é deduzida de φ_j (com $j < i$) por (**\exists -In**), sendo que φ_j depende de φ em d ; ou
- φ_i é deduzida de φ_j (com $j < i$) por (**\forall -In**), sendo que φ_j depende de φ em d .

■

O seguinte resultado é demonstrado a partir de considerações gerais sobre sistemas de Hilbert exatamente como o seu correspondente para a lógica clássica (ver [Men87]).

Lema 3.28. *Se ψ não depende de φ numa derivação de ψ a partir de $\Gamma \cup \{\varphi\}$ então $\Gamma \vdash \psi$.*

Teorema 3.29 (Teorema da Dedução). *Suponha que numa **LFI** quantificada existe uma derivação de ψ a partir de $\Gamma \cup \{\varphi\}$ tal que todas as aplicações das regras de quantificação (**\exists -In**) e (**\forall -In**) em fórmulas que dependem de φ são com relação a variáveis que não ocorrem livres em φ . Então $\Gamma \vdash \varphi \rightarrow \psi$.*

Demonstração. Seja $d = \varphi_1, \dots, \varphi_n$ uma derivação de ψ a partir de $\Gamma \cup \{\varphi\}$ nas condições do enunciado do teorema (logo, $\varphi_n = \psi$). Provaremos por indução em i que $\Gamma \vdash \varphi \rightarrow \varphi_i$ para todo $1 \leq i \leq n$. Em particular, teremos que $\Gamma \vdash \varphi \rightarrow \psi$ como desejado.

Se $n = 1$ então φ_1 é (instância de) um axioma, ou $\varphi_1 \in \Gamma \cup \{\varphi\}$. Em qualquer dos casos (e dado o Teorema 3.24) segue que $\Gamma \vdash \varphi \rightarrow \varphi_1$.

Suponha agora que $\Gamma \vdash \varphi \rightarrow \varphi_j$ para todo $1 \leq j < i$, e $i \geq 2$. Temos 4 casos para analisar:

- 1) φ_1 é um axioma, ou $\varphi_1 \in \Gamma \cup \{\varphi\}$. A prova é como acima.
- 2) existem $j, k < i$ tais que $\varphi_k = \varphi_j \rightarrow \varphi_i$ e φ_i é obtido em d de φ_k e φ_j por **(MP)**. Pela hipótese de indução, $\Gamma \vdash \varphi \rightarrow (\varphi_j \rightarrow \varphi_i)$ e $\Gamma \vdash \varphi \rightarrow \varphi_j$. Pelo axioma **(Trans)** e por **(MP)** obtemos que $\Gamma \vdash \varphi \rightarrow \varphi_i$.
- 3) existe $j < i$ tal que $\varphi_j = \alpha \rightarrow \beta$ e $\varphi_i = \exists x\alpha \rightarrow \beta$ (com x não livre em β) é obtida de φ_j pela regra **(\exists -In)**. Pela hipótese de indução, $\Gamma \vdash \varphi \rightarrow \varphi_j$ e, pela hipótese sobre d , φ_j não depende de φ ou x não ocorre livre em φ . Temos então dois subcasos para analisar:
 - 3.1) φ_j não depende de φ . Pelo Lema 3.28, $\Gamma \vdash \varphi_j$, isto é, $\Gamma \vdash \alpha \rightarrow \beta$. Aplicando a regra **(\exists -In)** temos que $\Gamma \vdash \exists x\alpha \rightarrow \beta$, isto é, $\Gamma \vdash \varphi_i$. Daqui, $\Gamma \vdash \varphi \rightarrow \varphi_i$.
 - 3.2) x não ocorre livre em φ . Dado que $\Gamma \vdash \varphi \rightarrow \varphi_j$, isto é, $\Gamma \vdash \varphi \rightarrow (\alpha \rightarrow \beta)$ então, pela regra **(\forall -In)**, temos que $\Gamma \vdash \varphi \rightarrow \forall x(\alpha \rightarrow \beta)$. Considere agora a seguinte derivação:
 1. $\forall x(\alpha \rightarrow \beta)$ (premissa)
 2. $\forall x(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta)$ (axioma)
 3. $\alpha \rightarrow \beta$ (**MP** 1,2)
 4. $\exists x\alpha \rightarrow \beta$ (**\exists -In** 3)

Isto mostra que $\forall x(\alpha \rightarrow \beta) \vdash \exists x\alpha \rightarrow \beta$. Pelo Teorema 3.10(vi) (adaptado às nossas lógicas) temos que $\varphi \rightarrow \forall x(\alpha \rightarrow \beta) \vdash \varphi \rightarrow (\exists x\alpha \rightarrow \beta)$. Pela transitividade da relação de derivabilidade temos que $\Gamma \vdash \varphi \rightarrow (\exists x\alpha \rightarrow \beta)$, isto é, $\Gamma \vdash \varphi \rightarrow \varphi_i$.

- 4) existe $j < i$ tal que $\varphi_j = \alpha \rightarrow \beta$ e $\varphi_i = \alpha \rightarrow \forall x\beta$ (com x não livre em α) é obtida de φ_j pela regra **(\forall -In)**. Pela hipótese de indução, $\Gamma \vdash \varphi \rightarrow \varphi_j$ e, pela hipótese sobre d , φ_j não depende de φ ou x não ocorre livre em φ . Temos novamente dois subcasos para analisar:
 - 4.1) φ_j não depende de φ . Pelo Lema 3.28, $\Gamma \vdash \varphi_j$, isto é, $\Gamma \vdash \alpha \rightarrow \beta$. Aplicando a regra **(\forall -In)** temos que $\Gamma \vdash \alpha \rightarrow \forall x\beta$, isto é, $\Gamma \vdash \varphi_i$. Daqui, $\Gamma \vdash \varphi \rightarrow \varphi_i$.
 - 4.2) x não ocorre livre em φ . Dado que $\Gamma \vdash \varphi \rightarrow \varphi_j$, isto é, $\Gamma \vdash \varphi \rightarrow (\alpha \rightarrow \beta)$ então, pela regra **(\forall -In)**, temos que $\Gamma \vdash \varphi \rightarrow \forall x(\alpha \rightarrow \beta)$. Considere agora a seguinte derivação:
 1. $\forall x(\alpha \rightarrow \beta)$ (premissa)
 2. $\forall x(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta)$ (axioma)

3. $\alpha \rightarrow \beta$ (**MP** 1,2)

4. $\alpha \rightarrow \forall x\beta$ (**\forall -In** 3)

Isto mostra que $\forall x(\alpha \rightarrow \beta) \vdash \alpha \rightarrow \forall x\beta$ ². Pelo Teorema 3.10(vi) (adaptado às nossas lógicas) temos que $\varphi \rightarrow \forall x(\alpha \rightarrow \beta) \vdash \varphi \rightarrow (\alpha \rightarrow \forall x\beta)$. Pela transitividade da relação de derivabilidade temos que $\Gamma \vdash \varphi \rightarrow (\alpha \rightarrow \forall x\beta)$, isto é, $\Gamma \vdash \varphi \rightarrow \varphi_i$.

Portanto $\Gamma \vdash \varphi \rightarrow \varphi_i$ para todo $1 \leq i \leq n$. Em particular, $\Gamma \vdash \varphi \rightarrow \psi$. \square

A versão demonstrada do Teorema da Dedução é bastante geral, mas por causa disso é um pouco difícil de saber de que modo aplicar. O seguinte caso particular é mais simples de aplicar e é suficiente para a maior parte das aplicações. De fato, vai ser esta a versão que usaremos nesta dissertação.

Corolário 3.30 (Teorema da Dedução, versão simplificada). *Suponha que numa LFI quantificada existe uma derivação de ψ a partir de $\Gamma \cup \{\varphi\}$ tal que todas as aplicações das regras de quantificação (**\exists -In**) e (**\forall -In**) são com relação a variáveis que não ocorrem livres em φ . Então $\Gamma \vdash \varphi \rightarrow \psi$.*

O seguinte resultado será de muita utilidade na prova do Teorema da Completude para LFI's quantificadas.

Teorema 3.31 (Teorema das Constantes). *Dada uma teoria Δ sobre uma linguagem L , e seja Δ' uma extensão sua por constantes, isto é, Δ' apenas acrescenta a Δ todas as instâncias dos axiomas lógicos utilizando os termos gerados por L junto com um certo conjunto de novas constantes. Logo, se tivermos, para $\varphi \in L$:*

$$\Delta' \vdash \varphi$$

então, igualmente é possível deduzir:

$$\Delta \vdash \varphi$$

Demonstração. É análoga ao caso clássico. Assim, considere uma derivação d de φ a partir de Δ' na lógica dada, e sejam c_1, \dots, c_n as constantes novas de Δ' (que não são parte da linguagem L) que ocorrem na derivação. Sejam x_1, \dots, x_n variáveis diferentes entre si que não ocorrem em d , e seja d' a sequência de fórmulas de L obtida de d substituindo toda ocorrência de c_i pela variável x_i , para $i = 1, \dots, n$, dado que as fórmulas de Δ' que não pertencem a Δ são instâncias de axiomas,

²A partir deste resultado, e aplicando o Teorema da Dedução, obtemos como teorema um dos axiomas presentes no sistema proposto em [Men87] para a lógica de primeira ordem clássica, não presente na nossa axiomatização das LFI's quantificadas.

as fórmulas de d' que correspondem a estas fórmulas serão agora instâncias de axiomas na linguagem L (pois as constantes novas foram trocadas por variáveis). Desta maneira, a sequência d' é uma derivação de φ a partir de Δ na lógica dada. \square

Teorema 3.32 (Regra de Conversão α). *Seja ϕ uma fórmula arbitrária e z uma variável livre para x em ϕ e totalmente indicada em $\phi_x[z]$ (isto é, z não ocorre em ϕ). Então:*

$$\vdash \exists x \phi \rightarrow \exists z \phi_x[z]$$

$$\vdash \forall x \phi \rightarrow \forall z \phi_x[z]$$

Demonstração. Para provar a primeira parte, devemos observar que $\phi = (\phi_x[z])_z[x]$, desde que z não ocorra em ϕ . A partir disto, obtemos a seguinte prova:

1. $\phi \rightarrow \exists z \phi_x[z]$ (\exists -**Ax**)
2. $\exists x \phi \rightarrow \exists z \phi_x[z]$ (\exists -**In**)

Para a segunda parte, considere a seguinte prova:

1. $\forall x \phi \rightarrow \phi_x[z]$ (\forall -**Ax**)
2. $\forall x \phi \rightarrow \forall z \phi_x[z]$ (\forall -**In**)

\square

Agora vamos demonstrar variações das regras \exists -**In** e \forall -**In**. Estes resultados são essenciais para a completude (Seção 3.3.4), mais precisamente na demonstração de que teorias não triviais podem ser estendidas *conservativamente* a teorias de Henkin não triviais (Teorema 3.60). Previamente precisamos provar um lema. Nos itens (ii), (iii) e na prova dos itens (iv) e (v) estaremos usando a negação forte (lembre da Observação 3.20).

Lema 3.33. *Nas duas lógicas consideradas vale o seguinte:*

- (i) $\vdash (\alpha \rightarrow \beta)$ implica $\vdash (\exists x \alpha \rightarrow \exists x \beta)$
- (ii) $\vdash \exists x \sim \sim \alpha \rightarrow \exists x \alpha$
- (iii) $\vdash \sim \forall x \sim \alpha \rightarrow \exists x \alpha$
- (iv) $\vdash (\forall x \alpha \rightarrow \beta) \rightarrow \exists x (\alpha \rightarrow \beta)$ se x não ocorre livre em β .
- (v) $\vdash (\alpha \rightarrow \exists x \beta) \rightarrow \exists x (\alpha \rightarrow \beta)$ se x não ocorre livre em α .

Demonstração. (i) Suponha que $\vdash (\alpha \rightarrow \beta)$. Dado que $\vdash (\beta \rightarrow \exists x\beta)$, por $(\exists\text{-Ax})$, então $\vdash (\alpha \rightarrow \exists x\beta)$, pela transitividade da implicação. O resultado segue pela regra $(\exists\text{-In})$.

(ii) Dado que $\vdash \sim\sim\alpha \rightarrow \alpha$, pelo Teorema 3.10(iii), o resultado segue do item (i).

(iii) Temos que $\vdash \sim\forall x\sim\alpha \rightarrow \exists x\sim\sim\alpha$, pelo axioma (\mathbf{PQ}) . O resultado segue pelo item (ii) e pela transitividade da implicação.

(iv) Usaremos o Teorema 3.10(iv) (adaptado às nossas lógicas) para provar primeiro que $(\forall x\alpha \rightarrow \beta) \vdash \exists x(\alpha \rightarrow \beta)$. Temos que $(\forall x\alpha \rightarrow \beta), \forall x\alpha \vdash \beta$. Mas $\vdash \beta \rightarrow (\alpha \rightarrow \beta)$ e $\vdash (\alpha \rightarrow \beta) \rightarrow \exists x(\alpha \rightarrow \beta)$. Logo, por transitividade, $(\forall x\alpha \rightarrow \beta), \forall x\alpha \vdash \exists x(\alpha \rightarrow \beta)$. Por outro lado, $\vdash \sim\alpha \rightarrow (\alpha \rightarrow \beta)$ e então $\vdash \exists x\sim\alpha \rightarrow \exists x(\alpha \rightarrow \beta)$, por (i). Dado que $\vdash \sim\forall x\alpha \rightarrow \exists x\sim\alpha$, pelo axioma (\mathbf{PQ}) , então segue que $\vdash \sim\forall x\alpha \rightarrow \exists x(\alpha \rightarrow \beta)$ e daí $(\forall x\alpha \rightarrow \beta), \sim\forall x\alpha \vdash \exists x(\alpha \rightarrow \beta)$. Portanto, pelo Teorema 3.10(iv) (adaptado às nossas lógicas), $(\forall x\alpha \rightarrow \beta) \vdash \exists x(\alpha \rightarrow \beta)$. O resultado segue pelo Teorema da Dedução, que pode ser aplicado pois x não ocorre livre em β .

(v) De $\alpha \rightarrow \exists x\beta, \forall x\sim(\alpha \rightarrow \beta)$ inferimos $\alpha \rightarrow \exists x\beta, \sim(\alpha \rightarrow \beta)$ e daí $\alpha \rightarrow \exists x\beta, \alpha, \sim\beta$, pelo Teorema 3.10(vii). Daqui segue $\exists x\beta, \sim\beta$. Mas $\sim\beta = \beta \rightarrow \perp$ e $\beta \rightarrow \perp \vdash \exists x\beta \rightarrow \perp$, pela regra $(\exists\text{-In})$. Isto é, $\sim\beta \vdash \sim\exists x\beta$. Combinando com a inferência acima temos que de $\alpha \rightarrow \exists x\beta, \forall x\sim(\alpha \rightarrow \beta)$ segue-se $\exists x\beta, \sim\exists x\beta$ e daí segue \perp . Portanto, $\alpha \rightarrow \exists x\beta \vdash \sim\forall x\sim(\alpha \rightarrow \beta)$, pelo Teorema da Dedução. Pelo item (iii), $\alpha \rightarrow \exists x\beta \vdash \exists x(\alpha \rightarrow \beta)$. Dado que x não ocorre livre em α , podemos aplicar o Teorema da Dedução para obter o resultado desejado. \square

Lema 3.34. *Se x não ocorre livre em λ e ψ , então valem as seguintes inferências nas duas lógicas consideradas:*

1.

$$\frac{\Gamma \vdash (\phi \rightarrow \lambda) \rightarrow \psi}{\Gamma \vdash (\forall x \phi \rightarrow \lambda) \rightarrow \psi}$$

1'

$$\frac{\Gamma \vdash (\phi \rightarrow \lambda)}{\Gamma \vdash (\forall x \phi \rightarrow \lambda)}$$

2.

$$\frac{\Gamma \vdash (\lambda \rightarrow \phi) \rightarrow \psi}{\Gamma \vdash (\lambda \rightarrow \exists x \phi) \rightarrow \psi}$$

2'

$$\frac{\Gamma \vdash (\lambda \rightarrow \phi)}{\Gamma \vdash (\lambda \rightarrow \exists x \phi)}$$

Demonstração. 1) Pelo Lema 3.33(iv) temos que $\vdash (\forall x\phi \rightarrow \lambda) \rightarrow \exists x(\phi \rightarrow \lambda)$. Logo, pelo Teorema 3.10(v), $\vdash (\exists x(\phi \rightarrow \lambda) \rightarrow \psi) \rightarrow ((\forall x\phi \rightarrow \lambda) \rightarrow \psi)$.

Suponha então que $\Gamma \vdash (\phi \rightarrow \lambda) \rightarrow \psi$. Logo $\Gamma \vdash \exists x(\phi \rightarrow \lambda) \rightarrow \psi$, pela regra (**\exists -In**). Portanto, $\Gamma \vdash (\forall x\phi \rightarrow \lambda) \rightarrow \psi$.

1') Considere a seguinte derivação:

1. $\phi \rightarrow \lambda$ (premissa)
2. $\forall x\phi$ (premissa)
3. $\forall x\phi \rightarrow \phi$ (**\forall -Ax**)
4. ϕ (**MP** 2,3)
5. λ (**MP** 1,4)

Assim, $\phi \rightarrow \lambda, \forall x\phi \vdash \lambda$ e então, pelo Teorema da Dedução (que pode ser aplicado pois x não ocorre livre em $\forall x\phi$) temos que $\phi \rightarrow \lambda \vdash \forall x\phi \rightarrow \lambda$. O resultado segue pela transitividade da derivabilidade.

2) Pelo Lema 3.33(v), $\vdash (\lambda \rightarrow \exists x \phi) \rightarrow \exists x(\lambda \rightarrow \phi)$. Aplicando o Teorema 3.10(v) obtemos $\vdash (\exists x(\lambda \rightarrow \phi) \rightarrow \psi) \rightarrow ((\lambda \rightarrow \exists x \phi) \rightarrow \psi)$.

Suponha então que $\Gamma \vdash (\lambda \rightarrow \phi) \rightarrow \psi$. Pela regra (**\exists -In**) inferimos que $\Gamma \vdash \exists x(\lambda \rightarrow \phi) \rightarrow \psi$. Logo, $\Gamma \vdash (\lambda \rightarrow \exists x \phi) \rightarrow \psi$.

2') Considere a seguinte derivação:

1. $\lambda \rightarrow \phi$ (premissa)
2. λ (premissa)
3. ϕ (**MP** 1,2)

4. $\phi \rightarrow \exists x\phi$ (\exists -**Ax**)

5. $\exists x\phi$ (**MP** 3,4)

Assim, $\lambda \rightarrow \phi, \lambda \vdash \exists x\phi$. Pelo Teorema da Dedução (dado que x não ocorre livre em λ), $\lambda \rightarrow \phi \vdash \lambda \rightarrow \exists x\phi$. Portanto, se $\Gamma \vdash \lambda \rightarrow \phi$ então $\Gamma \vdash \lambda \rightarrow \exists x\phi$.

□

3.3.3 Semântica de Estruturas Paraconsistentes

Esta seção é dedicada a provar a correção e a completude das **LFI**'s quantificadas **QmbC** e **QmCi** com relação à semântica de estruturas tarskianas munidas de valorações paraconsistentes. As definições semânticas serão feitas de maneira gradual. Uma novidade introduzida na semântica de valorações é a exigência da validade do Lema da Substituição, ver Observação 3.47 e Teorema 3.51.

Definição 3.35 (Estrutura Básica). Seja Σ uma assinatura de primeira ordem (veja a Definição 3.14), e seja L_Σ a linguagem associada (veja a Definição 3.16). Uma *estrutura básica* (ou tarskiana) sobre a linguagem L_Σ consiste num par

$$\mathfrak{A} = \langle A, I_{\mathfrak{A}} \rangle$$

em que A é um conjunto não-vazio (o domínio da estrutura) e $I_{\mathfrak{A}}$ é uma função de interpretação, isto é:

$$I_{\mathfrak{A}} : Func_\Sigma \cup Pred_\Sigma \rightarrow Func(A) \cup Pred(A),$$

tal que $Func(A)$ é o conjunto de todas as possíveis funções de aridade arbitrária com parâmetros e contradomínio em A e $Pred(A)$ o conjunto de todos os possíveis predicados (também de aridade arbitrária) sobre A , de maneira que a função $I_{\mathfrak{A}}$ preserva as aridades.

■

Uma estrutura básica \mathfrak{A} define uma função de interpretação $(\cdot)^{\mathfrak{A}} : TF_\Sigma \rightarrow A$ do conjunto TF_Σ dos termos fechados (isto é, sem variáveis livres) no conjunto A . Esta função é definida por indução na estrutura dos termos fechados da maneira seguinte:

- $c^{\mathfrak{A}} = I_{\mathfrak{A}}(c)$ se c é uma constante;
- $f(t_1, \dots, t_n)^{\mathfrak{A}} = I_{\mathfrak{A}}(f)(t_1^{\mathfrak{A}}, \dots, t_n^{\mathfrak{A}})$ se f é um símbolo de função n -ária (com $n \geq 1$) e $t_1, \dots, t_n \in TF_\Sigma$.

Para poder interpretar os quantificadores numa estrutura básica precisamos nomear cada um dos elementos do domínio da estrutura utilizando constantes novas, de maneira que possamos instanciar a variável da quantificação na fórmula quantificada em todos os elementos do domínio da estrutura utilizando estas constantes. Isto motiva a seguinte definição.

Definição 3.36 (Linguagem e Estrutura Diagrama). Considere uma estrutura básica \mathfrak{A} para uma linguagem L_Σ de primeira ordem. A *linguagem diagrama* de \mathfrak{A} , denotada por $L_\Sigma(\mathfrak{A})$, é definida sobre a assinatura Σ_A obtida de Σ pelo acréscimo de uma nova constante \bar{a} para cada elemento a do domínio A da estrutura \mathfrak{A} . Denotaremos por $T_\Sigma(\mathfrak{A})$, ou simplesmente por $T(\mathfrak{A})$, o conjunto de termos da linguagem diagrama. A estrutura básica

$$\mathfrak{A} = \langle A, I_{\mathfrak{A}} \rangle \quad \text{para } L_\Sigma$$

pode ser naturalmente estendida a uma estrutura básica

$$\widehat{\mathfrak{A}} = \langle A, \widehat{I}_{\mathfrak{A}} \rangle \quad \text{para } L(\mathfrak{A})$$

definindo $\widehat{I}_{\mathfrak{A}}(\bar{a}) = a$ para todo $a \in A$.

■

Notação 3.37. Toda vez que estivermos discutindo uma estrutura para certa linguagem, assumiremos implicitamente que sua função de interpretação já está estendida para os elementos da linguagem diagrama. Isto é, estaremos assumindo tacitamente a identificação entre \mathfrak{A} e $\widehat{\mathfrak{A}}$. Da mesma maneira, e por simplicidade, identificaremos \bar{a} com a , para todo $a \in A$.

■

Definição 3.38 (Estrutura). Uma estrutura é qualquer par $\langle \mathfrak{A}, v \rangle$, em que \mathfrak{A} é uma estrutura básica sobre alguma linguagem L de primeira ordem e v uma bivaloração $v : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$ que obedece à seguinte condição:

$$v(P(t_1, \dots, t_n)) = 1 \iff \langle t_1^{\widehat{\mathfrak{A}}}, \dots, t_n^{\widehat{\mathfrak{A}}} \rangle \in I_{\mathfrak{A}}(P) \quad (vPred)$$

para todo predicado n -ário P e termos fechados $t_1, \dots, t_n \in TF_{\Sigma_A}$.

■

Note que, em particular,

$$v(P(\bar{a}_1, \dots, \bar{a}_n)) = 1 \iff \langle a_1, \dots, a_n \rangle \in I_{\mathfrak{A}}(P)$$

para todo predicado n -ário P e indivíduos $a_1, \dots, a_n \in A$.

Quando não houver risco de confusão, designaremos a estrutura $\langle \mathfrak{A}, \nu \rangle$ por \mathfrak{A} , e ν por $\nu_{\mathfrak{A}}$. Se quisermos nos referir a estrutura básica sobre a qual ν está definida, falaremos da *pré-estrutura* de \mathfrak{A} .

A noção de satisfatibilidade será feita sobre sentenças da linguagem estendida (lembre da Definição 3.17 de sentença).

Definição 3.39 (Satisfatibilidade). Dizemos que uma estrutura \mathfrak{A} satisfaz a sentença $\varphi \in S_{L(\mathfrak{A})}$, o que é denotado por

$$\mathfrak{A} \models \varphi ,$$

se

$$\nu_{\mathfrak{A}}(\varphi) = 1 .$$

Podemos generalizar essa noção para um conjunto Γ de sentenças. Isto é, $\mathfrak{A} \models \Gamma$ se, e somente se, $\mathfrak{A} \models \gamma$ para toda $\gamma \in \Gamma$.

■

Definição 3.40 (Consequência Lógica). Dada uma classe \mathfrak{C} de estruturas sobre uma assinatura Σ , dizemos que uma sentença $\varphi \in S_{L_{\Sigma}}$ é consequência lógica (com relação a \mathfrak{C}) de um conjunto de sentenças $\Gamma \subseteq S_{L_{\Sigma}}$ se, e somente se, para toda estrutura \mathfrak{A} na classe \mathfrak{C} , toda vez que $\mathfrak{A} \models \Gamma$, temos que $\mathfrak{A} \models \varphi$. Em símbolos:

$$\Gamma \models_{\mathfrak{C}} \varphi$$

■

Definiremos agora o conceito de valoração paraconsistente, para interpretar as sentenças das linguagens paraconsistentes quantificadas. Como observamos antes, as **LFI**'s proposicionais tem um fragmento vero-funcional, formado pelos conectivos clássicos $\{\vee, \wedge, \rightarrow\}$. Por outro lado, os quantificadores também serão interpretados de maneira vero-funcional. Isto motiva separar a definição de valoração em duas etapas.

Definição 3.41 (Valoração positiva). Dada uma linguagem L_{Σ} sobre uma assinatura Σ , e uma estrutura \mathfrak{A} para essa linguagem, dizemos que a bivaloração $\nu_{\mathfrak{A}} : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$ é uma *valoração positiva* se obedece às seguintes condições, com $P \in P_{\Sigma}^n$, $\alpha, \beta, \phi \in S_{L(\mathfrak{A})}$ e \bar{a} o nome em $L(\mathfrak{A})$ para o indivíduo $a \in A$:

- Regras proposicionais:

$$\begin{aligned}
 v(\alpha \vee \beta) = 1 & \iff v(\alpha) = 1 \text{ ou } v(\beta) = 1 & (vOu) \\
 v(\alpha \wedge \beta) = 1 & \iff v(\alpha) = 1 \text{ e } v(\beta) = 1 & (vE) \\
 v(\alpha \rightarrow \beta) = 1 & \iff v(\alpha) = 0 \text{ ou } v(\beta) = 1 & (vImp)
 \end{aligned}$$

- Regras quantificacionais:

$$\begin{aligned}
 v(\exists x\phi) = 1 & \iff v(\phi_x[\bar{a}]) = 1 \text{ para algum indivíduo } a \text{ de } A & (vEx) \\
 v(\forall x\phi) = 1 & \iff v(\phi_x[\bar{a}]) = 1 \text{ para todo indivíduo } a \text{ de } A & (vUni)
 \end{aligned}$$

■

Finalmente introduzimos as cláusulas que definem uma valoração paraconsistente. Para isso, precisaremos generalizar as definições semânticas.

Definição 3.42 (Substituição múltipla). Seja \mathfrak{A} uma estrutura para uma linguagem L , $\vec{x} = x_1, \dots, x_n$ uma sequência de variáveis diferentes. O conjunto de fórmulas de $L(\mathfrak{A})$ cujas variáveis livres ocorrem na sequência \vec{x} é denotado por $L(\mathfrak{A})_{\vec{x}}$ (dizemos que \vec{x} é um *contexto* para as fórmulas de $L(\mathfrak{A})_{\vec{x}}$). Analogamente definimos o conjunto $L_{\vec{x}}$ de todas as fórmulas de L no contexto \vec{x} . Dada uma sequência $\vec{a} = a_1, \dots, a_n$ de elementos de A e $\varphi \in L(\mathfrak{A})_{\vec{x}}$, denotamos por $\varphi_{\vec{x}}[\vec{a}]$ a sentença de $S_{L(\mathfrak{A})}$ obtida de φ pela substituição simultânea de cada ocorrência livre de x_i em φ pela constante \bar{a}_i . Analogamente, se t é um termo sobre a assinatura Σ_A cujas variáveis ocorrem na lista \vec{x} então $t_{\vec{x}}[\vec{a}]$ é o termo fechado obtido de t substituindo uniformemente cada ocorrência da variável x_i pela constante \bar{a}_i associada a a_i . Denotaremos por $T(\mathfrak{A})_{\vec{x}}$ ao conjunto de todos os termos da assinatura de $L(\mathfrak{A})$ no contexto \vec{x} .

■

Definição 3.43 (Valoração estendida). Seja \mathfrak{A} uma estrutura básica para uma linguagem L , $\vec{x} = x_1, \dots, x_n$ uma sequência de variáveis diferentes e $\vec{a} = a_1, \dots, a_n$ uma sequência de elementos de A e $\varphi \in L(\mathfrak{A})_{\vec{x}}$. Se $v : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$ é uma valoração sobre \mathfrak{A} , definimos sua extensão $v_{\vec{x}}^{\vec{a}} : L(\mathfrak{A})_{\vec{x}} \rightarrow \{0, 1\}$ como segue: $v_{\vec{x}}^{\vec{a}}(\varphi) = v(\varphi_{\vec{x}}[\vec{a}])$ para toda $\varphi \in L(\mathfrak{A})_{\vec{x}}$.

■

Claramente, se $\varphi \in L(\mathfrak{A})_{\vec{x}}$ e $\vec{y} = (\vec{x}; \vec{z})$ com $\vec{z} = z_1, \dots, z_m$ então $v_{\vec{y}}^{\vec{a}; \vec{b}}(\varphi) = v_{\vec{x}}^{\vec{a}}(\varphi)$ para toda sequência $\vec{b} = b_1, \dots, b_m$ em A . Em particular, $v(\varphi) = v_{\vec{x}}^{\vec{a}}(\varphi)$ para toda \vec{x} e para toda \vec{a} , se $\varphi \in S_{L(\mathfrak{A})}$.

Definição 3.44 (Satisfatibilidade estendida). Dizemos que uma estrutura $\langle \mathfrak{A}, v \rangle$ satisfaz a fórmula $\varphi \in L(\mathfrak{A})_{\vec{x}}$, o que é denotado por

$$\mathfrak{A} \models_{\vec{x}} \varphi ,$$

se

$$v_{\vec{x}}^{\vec{d}}(\varphi) = 1 \text{ para toda sequência } \vec{d} \text{ em } A.$$

Podemos generalizar essa noção para um conjunto Γ de fórmulas em $L(\mathfrak{A})_{\vec{x}}$. Assim, $\mathfrak{A} \models_{\vec{x}} \Gamma$ se, e somente se, $\mathfrak{A} \models_{\vec{x}} \gamma$ para toda $\gamma \in \Gamma$. ■

Definição 3.45 (Consequência semântica estendida). Seja $\vec{x} = x_1, \dots, x_n$ uma sequência de variáveis diferentes e $\Gamma \cup \{\varphi\} \subseteq L_{\vec{x}}$. Seja \mathfrak{C} uma classe de estruturas sobre a linguagem L . Dizemos que φ é consequência semântica de Γ com relação a \mathfrak{C} no contexto \vec{x} , denotado por $\Gamma \models_{\mathfrak{C}}^{\vec{x}} \varphi$ se, para toda estrutura \mathfrak{A} em \mathfrak{C} , toda vez que $\mathfrak{A} \models_{\vec{x}} \Gamma$, temos que $\mathfrak{A} \models_{\vec{x}} \varphi$. ■

Observe que, no caso que $\Gamma \cup \{\varphi\} \subseteq S_L$, a noção $\models_{\mathfrak{C}}^{\vec{x}}$ coincide com $\models_{\mathfrak{C}}$.

Definição 3.46 (Valorações Paraconsistentes). Seja $v : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$ uma valoração positiva sobre uma estrutura \mathfrak{A} . Dizemos que v é uma valoração para **QmbC** se, além da condições da Definição 3.41, satisfaz a seguintes cláusulas, com $\alpha, \beta, \varphi \in S_{L(\mathfrak{A})}$:

$$\begin{aligned} v(\alpha) = 0 & \implies v(\neg\alpha) = 1 & (vNeg) \\ v(\circ\alpha) = 1 & \implies v(\alpha) = 0 \text{ ou } v(\neg\alpha) = 0 & (vCon) \\ v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\varphi_z[t]) = v_{\vec{x}, z}^{\vec{a}, b}(\varphi) & \implies v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\neg\varphi_z[t]) = v_{\vec{x}, z}^{\vec{a}, b}(\neg\varphi) & (sNeg) \\ v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\varphi_z[t]) = v_{\vec{x}, z}^{\vec{a}, b}(\varphi) & \implies v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\circ\varphi_z[t]) = v_{\vec{x}, z}^{\vec{a}, b}(\circ\varphi) & (sCon) \end{aligned}$$

desde que t seja livre para z em φ e $b = (t_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^{\mathfrak{A}}$.

Finalmente, dizemos que v é uma valoração para **QmCi** se é uma valoração para **QmbC** satisfazendo, adicionalmente, as seguintes cláusulas:

$$\begin{aligned} v(\neg \circ \alpha) = 1 & \implies v(\alpha) = 1 \text{ e } v(\neg\alpha) = 1 & (vInc) \\ v(\circ \neg^n \circ \alpha) = 1 & \implies (\text{para todo } n \geq 0) & (vCCon) \end{aligned}$$

Observação 3.47. As cláusulas $sNeg$ e $sCon$, de caráter puramente técnico, estabelecem que se duas fórmulas na linguagem diagrama envolvendo substituições tem seu valor de verdade coincidindo, então este valor deve ser preservado pelos conectivos não vero-funcionais. Por exemplo, seja P é um símbolo de predicado unário e f é um símbolo de função unária. Seja $\varphi = P(z)$ e $t = f(x)$. Seja $b = (t_x[a])^{\mathfrak{A}}$. Logo, $b = f(\bar{a})^{\mathfrak{A}} = f^{\mathfrak{A}}(a)$. Assim,

$$v_x^a(P(z)_z[t]) = v_x^a(P(f(x))) = v(P(f(x))_x[a]) = v(P(f(\bar{a})))$$

enquanto que

$$v_z^b(P(z)) = v(P(z)_z[b]) = v(P(\overline{f^{\mathfrak{A}}(a)})).$$

É claro que os dois valores de verdade das duas sentenças de $L(\mathfrak{A})$ acima têm o mesmo valor de verdade para toda valoração v sobre \mathfrak{A} satisfazendo $vPred$. Logo, por $sNeg$ e $sCon$ temos que

$$v(\neg P(f(\bar{a}))) = v(\neg P(\overline{f^{\mathfrak{A}}(a)})) \text{ e } v(\circ P(f(\bar{a}))) = v(\circ P(\overline{f^{\mathfrak{A}}(a)})) \quad (*)$$

o que deveria ser esperado. Esta propriedade será fundamental para provar o Lema da Substituição que, por sua vez, vai ser crucial para provar a correção de **QmbC** e de **QmCi** com relação à semântica de estruturas.

É importante salientar que, sem a exigência destas propriedades das valorações, podemos encontrar estruturas que não verifiquem as duas equações (*), falsificando em consequência os axiomas $\exists\text{-Ax}$ e $\forall\text{-Ax}$ (ver Observação 3.57).

Definição 3.48 (Estrutura **QmbC**). Uma estrutura para **QmbC** é uma estrutura \mathfrak{A} tal que $v_{\mathfrak{A}}$ é uma valoração para **QmbC**.

Definição 3.49 (Estrutura **QmCi**). Uma estrutura para **QmCi** é uma estrutura \mathfrak{A} tal que $v_{\mathfrak{A}}$ é uma valoração para **QmCi**.

Teorema 3.50. A Definição 3.46 continua definindo a mesma classe de estruturas se trocarmos:

$$v(\alpha) = 0 \implies v(\neg\alpha) = 1 \quad (vNeg)$$

por

$$v(\neg\alpha) = 0 \implies v(\alpha) = 1 \quad (vNeg')$$

Ou:

$$v(\circ\alpha) = 1 \implies v(\alpha) = 0 \text{ ou } v(\neg\alpha) = 0 \quad (vCon)$$

por

$$v(\alpha) = 1 \text{ e } v(\neg\alpha) = 1 \implies v(\circ\alpha) = 0 \quad (vCon')$$

Ou ainda, para estruturas que obedecem à condição $vCCon$, $vNeg$ e $vCon$, da Definição 3.46, se trocarmos:

$$v(\neg \circ \alpha) = 1 \implies v(\alpha) = 1 \text{ e } v(\neg\alpha) = 1 \quad (vInc)$$

por

$$v(\alpha) = 0 \text{ ou } v(\neg\alpha) = 0 \implies v(\circ\alpha) = 1 \quad (vInc')$$

Demonstração. Aplicação direta da contrapositiva (que vale para a metalinguagem). \square

Tanto nas estruturas para **QmbC** quanto para **QmCi**, a negação (\neg) e o conectivo de consistência (\circ) não apresentam uma semântica composicional (verofuncional). No entanto, diferentemente do que ocorre em **QmbC**, o valor de verdade de $\circ\alpha$ está *em função* dos de α e $\neg\alpha$ em **QmCi**:

$$v(\circ\alpha) = 1 \text{ sse } v(\alpha) \neq v(\neg\alpha).$$

O seguinte resultado é fundamental para a prova da correção das **LFI**'s quantificadas com relação à sua semântica, e é provado da mesma maneira que na lógica clássica: por indução na complexidade da fórmula φ .

Teorema 3.51 (Lema da Substituição). *Seja t um termo livre para a variável z na fórmula φ . Suponha que $(\vec{x}; z)$ e $(\vec{x}; \vec{y})$ são contextos para φ e $\varphi_z[t]$, respectivamente. Sejam \mathfrak{A} uma **LFI** quantificada (**QmbC** ou **QmCi**) e $\langle \mathfrak{A}, v \rangle$ uma estrutura em que v é uma valoração paraconsistente sobre \mathfrak{A} (na lógica dada). Se $b = (t_{\vec{x}; \vec{y}}[\vec{a}; \vec{b}])^{\mathfrak{A}}$ então:*

$$v_{\vec{x}; \vec{y}}^{\vec{a}; \vec{b}}(\varphi_z[t]) = v_{\vec{x}; z}^{\vec{a}; b}(\varphi).$$

Demonstração. A prova é idêntica com a prova clássica, por indução na complexidade de $\varphi \in L(\mathfrak{A})_{\vec{x};z}$.

(a) $\varphi = P(t_1, \dots, t_k)$ com P predicado k -ário e t_1, \dots, t_k termos em $T(\mathfrak{A})_{\vec{x};z}$. Logo, $\varphi_z[t] = P((t_1)_z[t], \dots, (t_k)_z[t])$. Por definição de valoração estendida temos que:

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\varphi_z[t]) = 1 \text{ sse } \langle (((t_1)_z[t])_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}}, \dots, (((t_k)_z[t])_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}} \rangle \in I_{\mathfrak{A}}(P).$$

Por indução na complexidade do termo $u \in T(\mathfrak{A})_{\vec{x};z}$ é fácil provar que

$$((u_z[t])_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}} = (u_{\vec{x};z}[\vec{a};b])^{\widehat{\mathfrak{A}}}$$

para $b = (t_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}}$, logo

$$\langle (((t_1)_z[t])_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}}, \dots, (((t_k)_z[t])_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}} \rangle \in I_{\mathfrak{A}}(P)$$

se e somente se

$$\langle ((t_1)_{\vec{x};z}[\vec{a};b])^{\widehat{\mathfrak{A}}}, \dots, ((t_k)_{\vec{x};z}[\vec{a};b])^{\widehat{\mathfrak{A}}} \rangle \in I_{\mathfrak{A}}(P).$$

Dado que

$$\langle ((t_1)_{\vec{x};z}[\vec{a};b])^{\widehat{\mathfrak{A}}}, \dots, ((t_k)_{\vec{x};z}[\vec{a};b])^{\widehat{\mathfrak{A}}} \rangle \in I_{\mathfrak{A}}(P) \text{ sse } v_{\vec{x};z}^{\vec{a};b}(\varphi) = 1$$

então

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\varphi_z[t]) = 1 \text{ sse } v_{\vec{x};z}^{\vec{a};b}(\varphi) = 1.$$

(b) $\varphi = (\alpha \# \beta)$, com $\# \in \{\vee, \wedge, \rightarrow\}$. Assumindo que α e β satisfazem a propriedade (por hipótese de indução), então φ também satisfaz a propriedade, por causa de que v é vero-funcional para estes conectivos.

(c) $\varphi = \forall x \psi$. Logo, $\varphi_z[t] = \forall x(\psi_z[t])$. Dado que t é livre para a variável z na fórmula φ , então x não ocorre em t . Por definição de valoração,

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\varphi_z[t]) = v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\forall x(\psi_z[t])) = 1 \text{ sse } v_{\vec{x};\vec{y};x}^{\vec{a};\vec{b};a}(\psi_z[t]) = 1 \text{ para todo } a.$$

Por hipótese de indução, e dado que $b = (t_{\vec{x};\vec{y}}[\vec{a};\vec{b}])^{\widehat{\mathfrak{A}}} = (t_{\vec{x};\vec{y};x}[\vec{a};\vec{b};a])^{\widehat{\mathfrak{A}}}$ (pois x não ocorre em t),

$$v_{\vec{x};\vec{y};x}^{\vec{a};\vec{b};a}(\psi_z[t]) = 1 \text{ sse } v_{\vec{x};z;x}^{\vec{a};b;a}(\psi) = 1.$$

Por outro lado,

$$v_{\vec{x};z;x}^{\vec{a};b;a}(\psi) = 1 \text{ para todo } a \text{ sse } v_{\vec{x};z}^{\vec{a};b}(\forall x \psi) = v_{\vec{x};z}^{\vec{a};b}(\varphi) = 1$$

e assim

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\varphi_z[t]) = 1 \text{ sse } v_{\vec{x};z}^{\vec{a};b}(\varphi) = 1 .$$

(d) $\varphi = \exists x\psi$. É uma consequência de que $v(\exists x\delta) = v(\sim\forall x\sim\delta)$.

(e) $\varphi = \#\psi$, com $\# \in \{\neg, \circ\}$. Pela hipótese de indução,

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\psi_z[t]) = v_{\vec{x};z}^{\vec{a};b}(\psi)$$

e então, pelos axiomas *sNeg* e *sCon*,

$$v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\varphi_z[t]) = v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\#\psi_z[t]) = v_{\vec{x};z}^{\vec{a};b}(\#\psi) = v_{\vec{x};z}^{\vec{a};b}(\varphi) .$$

□

A importância dos axiomas *sNeg* e *sCon* fica patente na prova do Lema da Substituição apresentada acima.

Agora, vamos traçar alguns paralelos entre as estruturas no sentido clássico e as paraconsistentes.

Definição 3.52 (Valoração Clássica). Uma valoração clássica é uma valoração positiva $v : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$ sobre \mathfrak{A} , sendo L uma linguagem clássica (isto é, sem o conectivo \circ), que verifica também:

$$v(\alpha) = 0 \iff v(\neg\alpha) = 1 \quad (vTND)$$

■

Podemos entender as estruturas paraconsistentes como generalizações das estruturas clássicas, tendo em vista que o comportamento de todos os conectivos são vero-funcionais (o valor de verdade da expressão complexa depende do valor de suas subfórmulas) no ambiente clássico, enquanto nas paraconsistentes isso é um caso particular.

Teorema 3.53. *Dada uma estrutura \mathfrak{A} , sobre ela existe uma única valoração clássica.*

Demonstração. Por indução na complexidade das fórmulas. □

Logo, podemos identificar as estruturas, no sentido clássico, como um caso particular das nossas estruturas.

Definição 3.54 (Estrutura Clássica). Uma estrutura \mathfrak{A} é considerada uma estrutura clássica, dado que determina unicamente uma valoração clássica $v : S_{L(\mathfrak{A})} \rightarrow \{0, 1\}$.

■

Teorema 3.55. *Duas estruturas \mathfrak{A} e \mathfrak{B} clássicas, de igual assinatura, com mesmo domínio, que interpretem seus símbolos funcionais da mesma maneira e que coincidam sobre as bases de Herbrand (Definição 2.27) de suas linguagens diagrama:*

$$v_{\mathfrak{A}} \upharpoonright_{B_{L(\mathfrak{A})}} = v_{\mathfrak{B}} \upharpoonright_{B_{L(\mathfrak{B})}}$$

são, na realidade, a mesma estrutura:

$$\mathfrak{A} = \mathfrak{B}.$$

No entanto, podemos ter diferentes estruturas paraconsistentes que coincidam na sua base de Herbrand em comum.

Demonstração. A parte correspondente às duas estruturas clássicas é decorrência do Teorema 3.53.

Por outro lado, duas estruturas paraconsistentes que coincidam em uma base de Herbrand em comum às duas devem coincidir apenas em todas as fórmulas geradas a partir das fórmulas da base e das constantes lógicas que são vero-funcionais, ou seja, \wedge , \vee , \rightarrow , \exists e \forall . □

Pode ser observado, no entanto, que, duas estruturas para **QmCi** que coincidam na sua base de Herbrand em comum e, adicionalmente, coincidem em todas as fórmulas negadas, então coincidem em todas as fórmulas, isto é, acabam sendo a mesma estrutura (como no caso clássico).

3.3.4 Correção e completude das lógicas **QmbC** e **QmCi**

Uma vez que introduzimos a semântica para as **LFI**'s quantificadas, devemos provar que a relação de consequência sintática e semântica coincidem. O primeiro resultado que devemos provar é a correção dos cálculos **QmbC** e **QmCi** com relação às suas estruturas. Isto é, se $\Delta \cup \{\varphi\}$ é um conjunto de sentenças, então

$$\Delta \vdash \varphi \quad \Longrightarrow \quad \Delta \vDash \varphi$$

nas duas **LFI**'s quantificadas. Devemos observar que, embora as premissas (o conjunto Δ) e a conclusão (a fórmula φ) sejam sentenças, uma derivação de φ a partir de Δ pode (e em geral, esse é o caso) envolver fórmulas com variáveis livres.

Para ilustrar esta afirmação, considere as derivações de $\forall x(\alpha \rightarrow \beta) \vdash \exists x\alpha \rightarrow \beta$ e de $\forall x(\alpha \rightarrow \beta) \vdash \alpha \rightarrow \forall x\beta$ apresentadas nos itens 3.2) e 4.2) da prova do Teorema da Dedução 3.29. Isto justifica a utilização de valorações estendidas. Como observamos antes, a novidade introduzida nesta dissertação na definição de valorações com relação ao Lema da Substituição (Teorema 3.51) é fundamental para obter a prova de correção dos sistemas Hilbertianos, ver Observação 3.57.

Teorema 3.56 (Correção de **QmbC** e **QmCi**). *Seja \mathcal{L} uma LFI quantificada (**QmbC** ou **QmCi**) e seja $\vDash_{\mathcal{L}}$ a relação de consequência semântica associada usando a classe de estruturas correspondente (ver Definição 3.40). Para todo conjunto de sentenças $\Delta \cup \{\varphi\}$ temos que:*

$$\Delta \vDash_{\mathcal{L}} \varphi \quad \Longrightarrow \quad \Delta \vDash_{\mathcal{L}} \varphi .$$

Demonstração. Por indução no comprimento n de uma derivação $\varphi_1, \dots, \varphi_n$ de φ a partir de Δ , provaremos que, fixada uma estrutura básica \mathfrak{A} então, cada valoração paraconsistente v sobre \mathfrak{A} (na lógica dada) tal que $v(\gamma) = 1$ para todo $\gamma \in \Delta$ satisfaz o seguinte: $v_{\vec{x}}^{\vec{a}}(\varphi_i) = 1$ para toda sequência \vec{a} em A e para todo $i \leq i \leq n$, em que \vec{x} é um contexto para todas as fórmulas φ_i ($1 \leq i \leq n$). Em particular, teremos que $v(\varphi) = 1$ como desejado.

É importante observar que, para provar o resultado previsto, é suficiente demonstrar a seguinte propriedade semântica de \mathcal{L} :

- i $v_{\vec{x}}^{\vec{a}}(\psi) = 1$ para toda \vec{a} e para toda instância ψ de um axioma de \mathcal{L} ;
- ii se $v_{\vec{x}}^{\vec{a}}(\psi_1) = 1$ e $v_{\vec{x}}^{\vec{a}}(\psi_1 \rightarrow \psi_2) = 1$ para toda \vec{a} então $v_{\vec{x}}^{\vec{a}}(\psi_2) = 1$ para toda \vec{a} ;
- iii se $v_{\vec{x};y}^{\vec{a};b}(\psi_1 \rightarrow \psi_2) = 1$ para toda $(\vec{a}; b)$, e se a variável y não ocorre livre em ψ_1 , então $v_{\vec{x}}^{\vec{a}}(\psi_1 \rightarrow \forall y\psi_2) = 1$ para toda \vec{a} ;
- iv se $v_{\vec{x};y}^{\vec{a};b}(\psi_1 \rightarrow \psi_2) = 1$ para toda $(\vec{a}; b)$, e se a variável y não ocorre livre em ψ_2 , então $v_{\vec{x}}^{\vec{a}}(\exists y\psi_1 \rightarrow \psi_2) = 1$ para toda \vec{a} .

Para provar (i) é claro que é suficiente analisar os axiomas envolvendo quantificação: os outros axiomas são obviamente válidos pela sua natureza proposicional e pelo fato de que as **LFI**'s proposicionais são corretas para a semântica de valorações paraconsistentes (ver [CCM07]). O mesmo se aplica para o item (ii) (que trata de **MP**). Assim, considere os seguintes casos:

(i.1) $\psi = \forall z\gamma \rightarrow \gamma_z[t]$ em que t é livre para z em γ . Sejam \vec{x} as variáveis livres ocorrendo em $\forall z\gamma$ e $(\vec{x}; \vec{y})$ as variáveis livres ocorrendo em $\gamma_z[t]$. Considere uma sequência $(\vec{a}; \vec{b})$ de elementos de A . Se $v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\forall z\gamma) = 0$ então $v_{\vec{x};\vec{y}}^{\vec{a};\vec{b}}(\psi) = 1$. Se, por

outro lado, $v_{\vec{x},\vec{y}}^{\vec{a},\vec{b}}(\forall z\gamma) = v_{\vec{x}}^{\vec{a}}(\forall z\gamma) = 1$ então $v_{\vec{x},z}^{\vec{a},b}(\gamma) = 1$ para todo $b \in A$. Em particular, $v_{\vec{x},z}^{\vec{a},b}(\gamma) = 1$ para $b = (t_{\vec{x},\vec{y}}[\vec{a}; \vec{b}])^{\mathfrak{A}}$. Pelo Lema da Substituição (Teorema 3.51), $v_{\vec{x},\vec{y}}^{\vec{a},\vec{b}}(\gamma_z[t]) = v_{\vec{x},z}^{\vec{a},b}(\gamma)$, pois t é livre para z em γ . Daqui segue que $v_{\vec{x},\vec{y}}^{\vec{a},\vec{b}}(\gamma_z[t]) = 1$, como requerido.

(i.2) $\psi = \gamma_z[t] \rightarrow \exists z\gamma$ em que t é livre para z em γ . A prova é análoga à do item anterior.

(i.3) $\psi = \sim\forall x\alpha \rightarrow \exists x\sim\alpha$. Basta observar que $v(\sim\gamma) = 1$ sse $v(\gamma) = 0$ para toda γ . Para provar (iii), suponha que $v_{\vec{x},\vec{y}}^{\vec{a},b}(\psi_1 \rightarrow \psi_2) = 1$ para toda $(\vec{a}; b)$, sendo que a variável y não ocorre livre em ψ_1 . Fixe \vec{a} . Se $v_{\vec{x}}^{\vec{a}}(\psi_1) = 0$ então $v_{\vec{x}}^{\vec{a}}(\psi_1 \rightarrow \forall y\psi_2) = 1$. Por outro lado, se $v_{\vec{x}}^{\vec{a}}(\psi_1) = v_{\vec{x},\vec{y}}^{\vec{a},b}(\psi_1) = 1$ então, por hipótese, $v_{\vec{x},\vec{y}}^{\vec{a},b}(\psi_2) = 1$, para todo b . Daqui, $v_{\vec{x}}^{\vec{a}}(\forall y\psi_2) = 1$.

O item (iv) é provado de maneira análoga. \square

Observação 3.57. Como comentamos na Observação 3.47, as cláusulas *sNeg* e *sCon* permitem provar o Lema da Substituição e assim, por exemplo, vale que

$$v(\neg P(\overline{f^{\mathfrak{A}}(a, b)})) = v(\neg P(f(\vec{a}, \vec{b}))) \quad (*)$$

No exemplo acima, estamos considerando $\gamma = \neg P(z)$ e $t = f(x, y)$, com P um predicado unário. Assim, reproduzindo a prova acima do item (i.1), suponhamos que $v_{\vec{x},\vec{y}}^{\vec{a},b}(\forall z\gamma) = v(\forall z\neg P(z)) = 1$. Logo,

$$v(\neg P(\vec{e})) = 1 \text{ para todo } e \in A. \quad (1)$$

Em particular,

$$v(\neg P(\overline{f^{\mathfrak{A}}(a, b)})) = 1. \quad (2)$$

Por outro lado, em virtude de (*)

$$v_{\vec{x},\vec{y}}^{\vec{a},b}(\gamma_z[t]) = v_{\vec{x},\vec{y}}^{\vec{a},b}(\neg P(f(x, y))) = v(\neg P(f(\vec{a}, \vec{b}))) = 1. \quad (3)$$

Assim, se não temos a propriedade (*) assegurada pelo Lema da Substituição, não teremos nenhuma garantia de que (3) seja o caso, mesmo na presença de (1) e (2) e tendo ainda que, por definição, $v(P(\overline{f^{\mathfrak{A}}(a, b)})) = v(P(f(\vec{a}, \vec{b})))$. Noutras palavras, a instância

$$\forall z\neg P(z) \rightarrow \neg P(f(x, y))$$

do axioma $\forall\text{-Ax}$ pode ser falsificada numa estrutura cuja valoração não satisfaça a cláusula *sNeg*. Um argumento análogo pode ser feito com relação à cláusula *sCon*, falsificando nesse caso a instância

$$\forall z\circ P(z) \rightarrow \circ P(f(x, y))$$

do axioma \forall -Ax.

Acreditamos que esta observação nunca foi feita nas abordagens às lógicas paraconsistentes de primeira ordem prévias na literatura (por exemplo [Pod08], [AZ06], [dCBB98]), constituindo portanto uma importante contribuição conceitual original da presente dissertação para o entendimento destas lógicas não-clássicas.

■

A partir de agora procederemos nesta seção a provar a completude das **LFI**'s quantificadas com relação à sua semântica. Tentaremos fazer a prova da maneira mais genérica possível, para facilitar a extensão do método para outras **LFI**'s.

Na completude clássica, o que se demonstra é que teorias *não contraditórias* têm modelos. Para as lógicas paraconsistentes, o foco não está na *não contradição*, mas na *não trivialidade*, dado que admitem inconsistências sem a trivialização (a diferença com a lógica clássica). O método clássico, seja utilizando o teorema da compacidade provado em [Gö30], ou diretamente com a demonstração de [Hen49]), decorre da seguinte maneira:

$$\begin{aligned} \Delta \not\models \varphi &\implies \Delta, \neg\varphi \text{ não é contraditória} \\ \Delta, \neg\varphi \text{ não é contraditória} &\implies \Delta, \neg\varphi \text{ é satisfável} \\ \Delta, \neg\varphi \text{ é satisfável} &\implies \Delta \models \varphi \end{aligned}$$

Para a completude das **LFI**'s tomaremos um caminho mais direto, tendo em vista que algumas dessas implicações podem não funcionar (ver a discussão na Seção 5.1). A partir de uma teoria que não deduza uma fórmula φ qualquer (não sendo, portanto, trivial), encontraremos uma estrutura paraconsistente que satisfaça Δ mas não φ :

$$\Delta \not\models \varphi \implies \Delta \not\models \varphi$$

Por sucessivas extensões, encontraremos um modelo de Δ que não satisfaz φ , a partir das fórmulas e termos das linguagens das extensões da teoria Δ . Primeiramente, faremos a extensão por constantes de uma teoria qualquer e garantiremos, dessa maneira, a presença de constantes testemunhas (caracterizando uma teoria de Henkin, cf. Definição 3.59). Será provado, também, que tal extensão não deduz nenhuma fórmula a mais da linguagem original. Para realizar esse passo, nada exigimos da teoria e, com ele, garantiremos que as regras quantificacionais da Definição 3.41 funcionarão na estrutura final.

Na sequência, assumindo que uma dada teoria Δ não deduz a sentença φ , encontraremos $\bar{\Delta}$, extensão maximal sua que também não deduz φ , conforme o método clássico de Lindenbaum-Asser. Para encontrar tal extensão, faremos uso fundamental do axioma da escolha, sob a forma do teorema de Teichmüller-Tukey.

Dessa maneira, garantimos um contramodelo de φ que seja modelo de Δ , demonstrando que $\Delta \not\models \varphi$. Seguiremos de perto a demonstração de completude apresentada em [Sho67], e assim nossa prova de completude para **LFI**'s quantificadas é original e diferente daquela apresentada em [Pod08].

Teorias de Henkin

Uma noção de fundamental importância nas provas de completude para a lógica clássica e para algumas não-clássicas é a de conjunto de Henkin. Usaremos estes conjuntos na nossa prova de completude para **LFI**'s quantificadas. Previamente, precisamos estabelecer o seguinte resultado.

Teorema 3.58. *Qualquer teoria Δ de alguma lógica \mathcal{L} , considerada sobre uma linguagem $L = L_{\Sigma}$ que tenha alguma constante, induz uma estrutura (Definição 3.38) \mathfrak{D} tal que, para toda sentença $\varphi \in S_L$:*

$$\mathfrak{D} \models \varphi \quad \iff \quad \Delta \vdash_{\mathcal{L}} \varphi$$

Demonstração. O domínio $D = \text{dom}(\mathfrak{D})$ da estrutura a ser construída é formado pelo conjunto FT_{Σ} dos termos fechados (ou seja, sem variáveis) da linguagem L :

$$D = FT_{\Sigma}.$$

Definem-se, então, as interpretações das funções e predicados de maneira que a interpretação de f aplicada aos termos t_n é o próprio termo que simboliza essa aplicação, $f(t_1, \dots, t_n)$:

$$I_{\mathfrak{D}}(f)(t_1, \dots, t_n) = f(t_1, \dots, t_n);$$

em particular, $I_{\mathfrak{D}}(c) = c$ se c é uma constante. Desta maneira, pode ser provado por indução na complexidade de t que $t^{\mathfrak{D}} = t$, para todo termo fechado t .

Por outro lado, os predicados são interpretados como sendo verdadeiros exatamente naquelas n -uplas de termos que podem ser deduzidos sintaticamente, a partir de Δ . Formalmente,

$$\langle t_1, \dots, t_n \rangle \in I_{\mathfrak{D}}(P) \quad \iff \quad \Delta \vdash_{\mathcal{L}} P(t_1, \dots, t_n).$$

Finalmente, devemos definir a estrutura $\widehat{\mathfrak{D}}$ para a linguagem diagrama $L(\mathfrak{D})$ de \mathfrak{D} , assim como a bivaloração $v_{\mathfrak{D}} : S_{L(\mathfrak{D})} \rightarrow \{0, 1\}$ satisfazendo a cláusula $vPred$ (lembre da Definição 3.38). Observe que a assinatura de $L(\mathfrak{D})$ acrescenta uma constante \bar{t} para cada termo fechado $t \in FT_{\Sigma}$ (visto como um indivíduo de \mathfrak{D}), de maneira a ter $\widehat{I}^{\widehat{\mathfrak{D}}}(\bar{t}) = t$. Denotando por $TF(\mathfrak{D})$ o conjunto de termos fechados da linguagem diagrama $L(\mathfrak{D})$, defina agora uma função $*$: $TF(\mathfrak{D}) \rightarrow TF_{\Sigma}$ como segue:

- $(\bar{t})^* = t$ se $t \in FT_\Sigma$;
- $c^* = c$ se c é constante de Σ ;
- $(f(t_1, \dots, t_n))^* = f((t_1)^*, \dots, (t_n)^*)$ se f é símbolo de função de Σ .

É claro que $t^* = \widehat{t^\mathfrak{D}}$ para todo $t \in TF(\mathfrak{D})$. Estendemos agora a função $*$ para sentenças. Assim, seja $*$: $S_{L(\mathfrak{D})} \rightarrow S_L$ definida como segue:

- i $(P(t_1, \dots, t_n))^* = P(t_1^*, \dots, t_n^*)$ se $P(t_1, \dots, t_n)$ é sentença atômica;
- ii $(\varphi \# \psi)^* = (\varphi^* \# \psi^*)$ se $\# \in \{\wedge, \vee, \rightarrow\}$;
- iii $(\# \psi)^* = \#(\psi^*)$ se $\# \in \{\neg, \circ\}$;
- iv $(Qx \psi)^* = Qx(\psi^*)$ se $Q \in \{\forall, \exists\}$.

Definimos finalmente $v_{\mathfrak{D}} : S_{L(\mathfrak{D})} \rightarrow \{0, 1\}$ como segue:

$$v_{\mathfrak{D}}(\varphi) = 1 \quad \iff \quad \Delta \vdash_{\mathfrak{Q}} \varphi^* .$$

Assim, se $P(t_1, \dots, t_n)$ é sentença atômica de $L(\mathfrak{D})$, temos que

$$v_{\mathfrak{D}}(P(t_1, \dots, t_n)) = 1 \quad \iff \quad \Delta \vdash_{\mathfrak{Q}} P(t_1^*, \dots, t_n^*),$$

por definição de $v_{\mathfrak{D}}$ e pelo item (i) acima. Mas isto acontece se e somente se $\langle t_1^*, \dots, t_n^* \rangle \in I_{\mathfrak{D}}(P)$, pela definição de $I_{\mathfrak{D}}(P)$. Dado que $t^* = \widehat{t^\mathfrak{D}}$ para todo $t \in TF(\mathfrak{D})$, então obtemos finalmente que

$$v_{\mathfrak{D}}(P(t_1, \dots, t_n)) = 1 \quad \iff \quad \langle \widehat{t_1^\mathfrak{D}}, \dots, \widehat{t_n^\mathfrak{D}} \rangle \in I_{\mathfrak{D}}(P)$$

e então $v_{\mathfrak{D}}$ satisfaz a cláusula $vPred$ da Definição 3.38. Pela construção de $v_{\mathfrak{D}}$, é imediato que, para toda sentença $\varphi \in S_L$:

$$\mathfrak{D} \models \varphi \quad \iff \quad \Delta \vdash_{\mathfrak{Q}} \varphi .$$

□

Agora, munidos de uma estrutura, com interpretação para símbolos de predicados e funções e uma valoração incipiente, daremos mais um passo rumo à definição de uma estrutura paraconsistente para Δ .

Definição 3.59 (Teoria de Henkin). Dada uma teoria $\Delta \subseteq S_L$ de uma lógica \mathcal{L} , sobre uma linguagem L de primeira ordem, diz-se que ela é *teoria de Henkin* se satisfaz o seguinte: Para toda sentença da forma $\exists x \phi$ em S_L , existe uma constante c na linguagem de L tal que:

$$\Delta \vdash_{\mathcal{L}} \exists x \phi \quad \Longrightarrow \quad \Delta \vdash_{\mathcal{L}} \phi_x[c]$$

■

Como consequência da Definição 3.59, e pelo axioma **PQ**, temos que Δ é uma teoria de Henkin se e somente se, para toda sentença da forma $\forall x \phi$ em S_L , existe uma constante na linguagem de L tal que:

$$\Delta \not\vdash_{\mathcal{L}} \forall x \phi \quad \Longrightarrow \quad \Delta \not\vdash_{\mathcal{L}} \phi_x[c]$$

Esta característica das Teorias de Henkin é importante dado que, por causa do axioma **PQ**, as **LFI**'s quantificadas poderiam ser apresentadas numa linguagem com apenas um quantificador, sendo que o outro é introduzido como abreviação, exatamente como acontece com a lógica clássica de primeira ordem.

O próximo passo é provar que toda teoria pode ser estendida conservativamente a uma teoria de Henkin. A idéia da prova é ir adicionando constantes ao longo de vários níveis (ω níveis) que façam as vezes de testemunhas para fórmulas quantificadas do nível anterior. Por sua vez, essas constantes novas engendram novas fórmulas em cada nível, que serão testemunhadas por constantes do próximo nível. Tais testemunhas agem como comprovações das sentenças existenciais.

A construção é feita em duas etapas. Primeiro, estende-se Δ por meio das constantes, dando origem à teoria Δ' , com os mesmos axiomas não lógicos, mas com axiomas lógicos a mais: aquelas instâncias dos axiomas lógicos que envolvem as constantes novas. Pelo teorema das constantes, Δ' é extensão conservativa de Δ .

Na sequência, adicionam-se outros axiomas não lógicos à Δ' que envolvem as constantes novas. Desse processo resulta a teoria Δ^H . Tais axiomas garantirão que a teoria final seja uma teoria de Henkin. Por último, prova-se que os axiomas introduzidos podem ser eliminados de qualquer prova de uma fórmula da linguagem original, garantindo que Δ^H é uma extensão conservativa de Δ .

Teorema 3.60. *Toda teoria $\Delta \subseteq S_L$ de uma lógica $\mathcal{L} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$ pode ser estendida conservativamente a uma teoria de Henkin Δ^H . Ademais, qualquer extensão de Δ^H formada por sentenças na mesma linguagem de Δ^H também é uma teoria de Henkin.*

Demonstração. Começaremos definindo uma sequência enumerável crescente de assinaturas

$$\Sigma_0 \subseteq \Sigma_1 \subseteq \dots \subseteq \Sigma_n \subseteq \dots$$

tal que a linguagem gerada por Σ_n será denotada por L_n (isto é, $L_n = L_{\Sigma_n}$), de maneira a ter uma sequência enumerável crescente de linguagens

$$L(\Delta) = L_0 \subseteq L_1 \subseteq \dots \subseteq L_n \subseteq \dots$$

A definição das assinaturas é realizada da maneira seguinte:

- Σ_0 é a assinatura da linguagem $L(\Delta)$ de Δ ; logo, $L_0 = L_{\Sigma_0} = L(\Delta)$
- Σ_1 é obtida de Σ_0 acrescentando as novas constantes

$$\{ c_{\exists x\alpha} \mid \exists x\alpha \text{ é fórmula fechada de } L_0 \}$$

- para $n \geq 1$, Σ_{n+1} é obtida de Σ_n acrescentando as novas constantes

$$\{ c_{\exists x\alpha} \mid \exists x\alpha \text{ é fórmula fechada de } L_n \setminus L_{n-1} \}$$

Finalmente, definimos $\Sigma_\omega = \bigcup_{n \in \omega} \Sigma_n$ e $L_\omega = L_{\Sigma_\omega}$.

Considere agora a extensão Δ' de Δ , na qual são acrescentadas todas as instâncias dos axiomas lógicos na linguagem L_ω .

O seguinte passo é estender Δ' de maneira a introduzir axiomas que garantam que as novas constantes são de fato testemunhas para as sentenças existenciais correspondentes de L_ω ³.

Considere então a seguinte sequência de axiomas não-lógicos:

$$AX_0 = \emptyset$$

$$AX_n = \{ \exists x\phi \rightarrow \phi_x[c_{\exists x\phi}] \mid \exists x\phi \in S_{L_n} \} \quad (\text{para } n \geq 1)$$

Definimos finalmente o conjunto desejado:

$$\Delta^H = \Delta' \cup \bigcup_{n \in \omega} AX_n.$$

³ Dada uma sentença existencial de L_ω da forma $\exists x\phi$, pela finitude do comprimento das fórmulas, ou existe uma ocorrência em ϕ de uma constante em Σ_n , com n máximo, ou a fórmula é original de $L(\Delta)$. Existem, então, testemunhas para tal quantificação em Σ_{n+1} ou em Σ_1 , respectivamente. Tal processo independe da cardinalidade de $L(\Delta)$: depende apenas da finitude do comprimento das fórmulas, motivo pelo qual não é preciso definir recursões além de ω para obter teorias de Henkin.

Observe que $L(\Delta^H) = L_\omega$, isto é, Δ^H está definido na linguagem L_ω . Mais ainda, $\Delta^H \subseteq S_{L_\omega}$. Provaremos que Δ^H é uma extensão conservativa de Δ . Assim, seja $\phi \in S_{L(\Delta)}$ uma sentença da linguagem de Δ tal que

$$\Delta^H \vdash_{\mathcal{Q}} \phi .$$

Pela finitude do processo de derivação nos sistemas de Hilbert aqui considerados (nos quais as regras de inferência são finitárias, isto é, tem finitas premissas), temos que de fato apenas um número finito de fórmulas de Δ^H são utilizadas como premissas numa derivação em \mathcal{Q} de ϕ a partir de Δ^H . Portanto, existe um conjunto finito $\Gamma \subseteq \bigcup_{n \in \omega} AX_n$ tal que

$$\Delta', \Gamma \vdash_{\mathcal{Q}} \phi .$$

Seja $\exists x\psi \rightarrow \psi_x[c_{\exists x\psi}]$ em Γ , e seja $\Gamma' = \Gamma \setminus \{\exists x\psi \rightarrow \psi_x[c_{\exists x\psi}]\}$. Dado que Γ está constituído apenas por sentenças então podemos utilizar o Teorema da Dedução para inferir

$$\Delta', \Gamma' \vdash_{\mathcal{Q}} (\exists x\psi \rightarrow \psi_x[c_{\exists x\psi}]) \rightarrow \phi .$$

Observe que a constante $c_{\exists x\psi}$ somente aparece na conclusão e então, usando uma técnica análoga à utilizada na prova do Teorema 3.31, podemos substituir a constante por uma variável nova, digamos y :

$$\Delta', \Gamma' \vdash_{\mathcal{Q}} (\exists x\psi \rightarrow \psi_x[y]) \rightarrow \phi .$$

Pelo Lema 3.34(2),

$$\Delta', \Gamma' \vdash_{\mathcal{Q}} (\exists x\psi \rightarrow \exists y\psi_x[y]) \rightarrow \phi .$$

Por outro lado, $\vdash_{\mathcal{Q}} \exists x\psi \rightarrow \exists y\psi_x[y]$, pelo Teorema 3.32. Daqui,

$$\Delta', \Gamma' \vdash_{\mathcal{Q}} \phi .$$

Repetindo o processo, eliminamos, em finitos passos, uma a uma todas as sentenças de Γ' , provando assim que $\Delta' \vdash_{\mathcal{Q}} \phi$. Pelo Teorema 3.31, obtemos finalmente que $\Delta \vdash_{\mathcal{Q}} \phi$. Isto mostra que Δ^H é uma extensão conservativa de Δ .

Por fim, considere alguma extensão $\Delta^{H'}$ de Δ^H (em particular, podemos tomar $\Delta^{H'} = \Delta^H$), formada por sentenças de L_ω . Suponha que, para alguma sentença $\exists x\varphi \in L_\omega$, temos que $\Delta^{H'} \vdash_{\mathcal{Q}} \exists x\varphi$. Dado que $\Delta^{H'}$ estende Δ^H , então:

$$\Delta^{H'} \vdash_{\mathcal{Q}} \exists x\varphi \rightarrow \varphi_x[c_{\exists x\varphi}]$$

e assim

$$\Delta^{H'} \vdash_{\mathcal{Q}} \varphi_x[c_{\exists x\varphi}] .$$

Portanto, $\Delta^{H'}$ é teoria de Henkin. □

Teorema 3.61. *Toda teoria de Henkin Δ de $\mathcal{Q} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$, sobre uma linguagem L de primeira ordem, induz uma estrutura \mathfrak{D} sobre a mesma linguagem tal que $v_{\mathfrak{D}}$ obedece à regra quantificacional vEx da Definição 3.41. Tal estrutura satisfaz, restrita às sentenças de L , exatamente as consequências lógicas de Δ . Isto é, para toda sentença $\varphi \in S_L$:*

$$\mathfrak{D} \models \varphi \quad \iff \quad \Delta \vdash_{\mathcal{Q}} \varphi$$

Demonstração. Seja $\exists x \phi \in S_L$, e seja Σ a assinatura de L . Dado que Δ é uma teoria de Henkin, então

$$\Delta \vdash_{\mathcal{Q}} \exists x \phi \quad \implies \quad \Delta \vdash_{\mathcal{Q}} \phi_x[c] \text{ para alguma constante } c \text{ de } \Sigma.$$

Por outro lado, em virtude do axioma $(\exists\text{-Ax})$ e da regra **MP** temos que, em \mathcal{Q} :

$$\Delta \vdash_{\mathcal{Q}} \exists x \phi \quad \longleftarrow \quad \Delta \vdash_{\mathcal{Q}} \phi_x[c] \text{ para alguma constante } c \text{ de } \Sigma.$$

Logo:

$$\Delta \vdash_{\mathcal{Q}} \exists x \phi \quad \iff \quad \Delta \vdash_{\mathcal{Q}} \phi_x[c] \text{ para alguma constante } c \text{ de } \Sigma.$$

Agora, pelo Teorema 3.58, podemos obter uma estrutura \mathfrak{D} para Δ , cujo domínio é constituído pelo conjunto FT_{Σ} dos termos fechados (sem variáveis) de L , e satisfaz exatamente as consequências lógicas de Δ : para toda sentença $\varphi \in S_L$,

$$\mathfrak{D} \models \varphi \quad \iff \quad \Delta \vdash_{\mathcal{Q}} \varphi.$$

Mais ainda, pela construção de $v_{\mathfrak{D}}$ (ver prova do Teorema 3.58),

$$v_{\mathfrak{D}}(\varphi) = 1 \quad \iff \quad \Delta \vdash_{\mathcal{Q}} \varphi^*$$

para toda sentença φ da linguagem diagrama $L(\mathfrak{D})$ de \mathfrak{D} , em que φ^* é a sentença de L obtida de φ substituindo todas as ocorrências de constantes da forma \bar{t} (que nomeam termos fechados $t \in TF_{\Sigma}$, vistos como indivíduos de \mathfrak{D}) pelo próprio termo t . Seja agora $\exists x \phi$ uma sentença de $L(\mathfrak{D})$. Então:

$$v_{\mathfrak{D}}(\exists x \phi) = 1 \quad \iff \quad \Delta \vdash_{\mathcal{Q}} (\exists x \phi)^*.$$

Mas $(\exists x \phi)^* = \exists x(\phi)^*$, logo

$$v_{\mathfrak{D}}(\exists x\phi) = 1 \quad \iff \quad \Delta \vdash_{\mathfrak{D}} \exists x(\phi)^*.$$

Dado que Δ é teoria da Henkin, inferimos que

$$v_{\mathfrak{D}}(\exists x\phi) = 1 \quad \iff \quad \Delta \vdash_{\mathfrak{D}} ((\phi)^*)_x[c] \text{ para alguma constante } c \text{ de } \Sigma.$$

É muito simples provar por indução na complexidade de ϕ que

$$((\phi)^*)_x[c] = (\phi_x[\bar{c}])^*.$$

Logo,

$$v_{\mathfrak{D}}(\exists x\phi) = 1 \quad \iff \quad \Delta \vdash_{\mathfrak{D}} (\phi_x[\bar{c}])^* \text{ para alguma constante } c \text{ de } \Sigma.$$

Podemos então concluir que:

$$v_{\mathfrak{D}}(\exists x\phi) = 1 \quad \iff \quad v_{\mathfrak{D}}(\phi_x[\bar{c}]) \text{ para algum indivíduo } c \text{ de } TF_{\Sigma}.$$

Isto prova que $\langle \mathfrak{D}, v_{\mathfrak{D}} \rangle$ é uma estrutura que satisfaz a regra quantificacional vEx da Definição 3.41. □

Com isso, temos uma estrutura \mathfrak{D} para toda teoria de Henkin Δ . Tal estrutura satisfaz, restrito às sentenças de L , exatamente as consequências lógicas de Δ , e além disso, sua bivaloração $v_{\mathfrak{D}}$ obedece à regra quantificacional vEx da Definição 3.41.

Na próxima seção, partiremos de uma teoria que não deduz uma fórmula φ . Obteremos, então, uma extensão completa sua que também não deduzirá φ . A função característica de tal extensão obedecerá todas as regras da Definição 3.41 e, conforme seja uma teoria sobre **QmbC** ou **QmCi**, obedecerá às regras das Definições 3.48 ou 3.49. Pelo teorema anterior, então, haverá uma estrutura para tal extensão que satisfaz Δ , mas não φ , sendo também paraconsistente.

Se a teoria Δ for de Henkin, então tal extensão também o será, tendo em vista que está sobre a mesma linguagem (é um extensão simples).

Extensões Maximais: Argumento de Lindenbaum-Asser

Para poder estabelecer a completude precisamos de outra noção importante: a de conjunto maximal não trivial com relação a uma fórmula. A partir destes conjuntos obteremos contramodelos apropriados para provar a completude das lógicas, usando um argumento contrapositivo.

Definição 3.62 (Teoria não trivial em relação a γ). Dizemos que uma teoria Δ é não trivial em relação a γ se $\Delta \not\vdash \gamma$.

■

Dada uma teoria Δ não trivial em relação a γ , a possibilidade de construir extensões maximais de Δ com essa mesma característica é fundamental para se estabelecer a completude da semântica proposta.

A estrutura final \mathfrak{D} que estamos buscando tem três características importantes:

(i) satisfaz Δ :

$$\mathfrak{D} \models \Delta$$

(ii) não satisfaz γ :

$$\mathfrak{D} \not\models \gamma$$

(iii) é de natureza sintática:

$$\mathfrak{D} \models \phi \iff \phi \in D$$

em que D é o domínio de \mathfrak{D} . Para uma lógica \mathfrak{L} qualquer, a prova de ela é completa para uma dada semântica de estruturas \mathcal{L} significa estabelecer uma relação entre sua derivabilidade e a consequência semântica em questão (veja Definição 3.40). Significa estabelecer que, para toda teoria Γ , todas as consequências semânticas de Γ são demonstráveis a partir de Γ . Em particular, tudo que é logicamente válido é demonstrável. Logo, para provar a completude da lógica \mathfrak{L} para a semântica \mathcal{L} , é suficiente demonstrar que todas as suas teorias não triviais em relação a γ têm modelos em \mathcal{L} que são contra-modelos para γ .

Lema 3.63 (Equivalente do Teorema da Completude). *A completude da lógica \mathfrak{L} em relação à uma semântica de estruturas \mathcal{L} , em símbolos:*

$$\Gamma \models_{\mathcal{L}} \gamma \iff \Gamma \vdash_{\mathfrak{L}} \gamma,$$

é equivalente a se demonstrar que, para toda φ não deduzida por uma teoria Γ ($\Gamma \not\vdash_{\mathfrak{L}} \varphi$), existe uma estrutura \mathfrak{D} tal que:

$$\mathfrak{D} \models_{\mathcal{L}} \Gamma \quad e \quad \mathfrak{D} \not\models_{\mathcal{L}} \varphi.$$

Demonstração. O resultado é consequência imediata da definição de consequência lógica (Definição 3.40) e da contrapositiva do teorema da completude:

$$\Gamma \not\vdash_{\mathcal{Q}} \gamma \quad \implies \quad \Gamma \not\vdash_{\mathcal{L}} \gamma .$$

□

A próxima demonstração faz uso do teorema de Teichmüller-Tukey, um equivalente do axioma da escolha, conforme [Sho67, p.47]. Tal teorema afirma que se uma família J de subconjuntos de algum conjunto U tiver *carácter finito*, então nessa família deve haver um elemento maximal com relação a inclusão (\subseteq). Ter carácter finito significa que um subconjunto de U pertence à tal família J exatamente quando todos seus subconjuntos finitos também pertencem.

Teorema 3.64 (Extensão Maximal não Trivial em Relação a γ). *Seja \mathcal{Q} uma lógica monotônica e compacta definida sobre uma linguagem L , e seja S_L o conjunto de todas as sentenças da linguagem. Para toda teoria $\Delta \subseteq S_L$ tal que Δ não é trivial em relação a γ , existe uma extensão de Δ não trivial em relação a γ , indicada por $\bar{\Delta}$, definida sobre a mesma linguagem L e que é maximal no seguinte sentido:*

se $\bar{\Delta} \subseteq \Delta' \subseteq S_L$ com Δ' não trivial em relação a γ então:

$$\Delta' = \bar{\Delta} .$$

Demonstração. Define-se $J(\Delta) \subseteq \mathcal{P}(S_L)$ como a família de todos os subconjuntos de S_L tais que unidos com Δ não derivam γ :

$$J(\Delta) = \left\{ A \subseteq S_L \mid \Delta \cup A \not\vdash_{\mathcal{Q}} \gamma \right\} .$$

Vamos demonstrar que $J(\Delta)$ tem carácter finito. Seja $A \in \mathcal{P}(S_L)$:

1. Se $A \in J(\Delta)$ e $\varphi \in S_L$ não é derivável a partir de $\Delta \cup A$:

$$\Delta \cup A \not\vdash_{\mathcal{Q}} \varphi ,$$

devido à **monotonicidade** de \mathcal{Q} , para todo $A^\circ \subseteq A$ finito:

$$\Delta \cup A^\circ \not\vdash_{\mathcal{Q}} \varphi .$$

Logo $\Delta \cup A^\circ$ não deduz nenhuma sentença a mais, incluindo γ . Portanto, $A^\circ \in J(\Delta)$. Logo, para todo $A^\circ \subseteq A$ finito:

$$A^\circ \in J(\Delta) .$$

2. Se $A \notin J(\Delta)$, pela definição de $J(\Delta)$:

$$\Delta \cup A \vdash_{\mathcal{L}} \gamma .$$

Devido à **compacidade** de \mathcal{L} , existe um $A^\circ \subseteq A$ finito tal que:

$$\Delta \cup A^\circ \vdash_{\mathcal{L}} \gamma .$$

Logo, existe um $A^\circ \subseteq A$ finito tal que:

$$A^\circ \notin J(\Delta) .$$

Por esses dois casos, podemos concluir então que, para todo $A \in \mathcal{P}(S_L)$:

$$A \in J(\Delta) \iff \forall A^\circ \subseteq A, A^\circ \text{ finito} : A^\circ \in J(\Delta) .$$

Isso significa que $J(\Delta)$ tem caráter finito. Logo, pelo Teorema de Teichmüller-Tukey, existe algum \bar{A} maximal em $J(\Delta)$. Para completar a demonstração, basta tomar:

$$\bar{\Delta} = \Delta \cup \bar{A}$$

□

Teorema 3.65 (Fechamento da Extensão Maximal). *Seja Δ uma teoria maximal não trivial em relação a γ numa lógica \mathcal{L} em que valem as leis da reflexividade e do corte. Então:*

$$\Delta \vdash_{\mathcal{L}} \phi \iff \phi \in \Delta .$$

Demonstração. Um dos sentidos dessa equivalência ($\phi \in \Delta$ implica $\Delta \vdash_{\mathcal{L}} \phi$) é decorrência direta da reflexividade de \mathcal{L} . Para provar o outro sentido da equivalência, suponha $\Delta \vdash_{\mathcal{L}} \phi$. Vamos demonstrar que $\Delta \cup \{\phi\}$ também não é trivial em relação a γ . Suponha, por absurdo, que $\Delta \cup \{\phi\}$ é trivial em relação a γ , ou seja:

$$\Delta, \phi \vdash_{\mathcal{L}} \gamma .$$

Como estamos supondo que $\Delta \vdash_{\mathcal{L}} \phi$, pela *lei do corte* teríamos que:

$$\Delta \vdash_{\mathcal{L}} \gamma ,$$

uma contradição. Logo, $\Delta \cup \{\phi\}$ é não trivial em relação a γ . Portanto, como $\Delta \subseteq \Delta \cup \{\phi\}$ e Δ é maximal não trivial em relação a γ , temos:

$$\Delta = \Delta \cup \{\phi\} ,$$

que só pode acontecer se:

$$\phi \in \Delta .$$

□

Teorema 3.66 (Extensões Maximais e Valorações). *Seja Δ uma teoria de $\mathcal{L} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$, maximal não trivial em relação a γ na lógica \mathcal{L} e definida sobre uma linguagem L . Sua função característica $v : S_L \rightarrow \{0, 1\}$, definida por:*

$$v(\phi) = 1 \quad \iff \quad \phi \in \Delta$$

obedece todas as Regras Proposicionais da Definição 3.41, junto com as cláusulas $vNeg$ e $vCon$ da Definição 3.46. Adicionalmente, no caso de ser maximal não trivial em relação a γ na lógica \mathbf{QmCi} , então v satisfaz também as cláusulas $vInc$ e $vCCon$.

Demonstração. Procederemos por vários casos, um para cada item das Definições 3.41 e 3.46 a ser demonstrado (lembre-se que $v(\varphi) = 1$ é equivalente a $\varphi \in \Delta$):

- Para o item vOu , se $\alpha \in \Delta$ ou $\beta \in \Delta$, pelos Axiomas **Ou0** ou **Ou1**, por **MP** temos que $\alpha \vee \beta \in \Delta$ (que é fechada sobre a relação de dedução de \mathcal{L} , pelo Teorema 3.65):

$$\alpha \in \Delta \quad \mathbf{ou} \quad \beta \in \Delta \quad \implies \quad \alpha \vee \beta \in \Delta$$

Agora, vamos demonstrar a outra parte da equivalência pela contrapositiva. Suponha $\alpha \notin \Delta$ e $\beta \notin \Delta$. Temos, pela maximalidade de Δ :

$$\Delta, \alpha, \vdash_{\mathcal{L}} \gamma$$

$$\Delta, \beta \vdash_{\mathcal{L}} \gamma$$

E, pelo teorema da dedução (Teorema 3.29), que sempre vale para sentenças:

$$\Delta \vdash_{\mathcal{L}} \alpha \rightarrow \gamma$$

$$\Delta \vdash_{\mathcal{L}} \beta \rightarrow \gamma$$

Pelo axioma **Ou2**:

$$\Delta \vdash_{\mathcal{L}} (\alpha \rightarrow \gamma) \rightarrow ((\beta \rightarrow \gamma) \rightarrow (\alpha \vee \beta \rightarrow \gamma))$$

e por duas aplicações de **MP**:

$$\Delta \vdash_{\mathcal{L}} \alpha \vee \beta \rightarrow \gamma$$

Obtemos portanto que $\Delta, \alpha \vee \beta \vdash_{\mathcal{L}} \gamma$, o que significa que $\alpha \vee \beta \notin \Delta$. Temos, então, a segunda parte da equivalência requerida pelo item vOu :

$$\alpha \notin \Delta \quad \mathbf{e} \quad \beta \notin \Delta \quad \implies \quad \alpha \vee \beta \notin \Delta$$

- Considere agora νE . Se $\alpha \in \Delta$ e $\beta \in \Delta$, pelo axioma **E0** e duas aplicações de **MP**, temos $\alpha \wedge \beta \in \Delta$. Suponha, agora, $\alpha \wedge \beta \in \Delta$. Pelos axiomas **E1** e **E2**, temos que $\alpha \in \Delta$ e $\beta \in \Delta$.
- Agora, νImp , a última das regras proposicionais de 3.41 (que devem valer tanto para teorias de **QmbC** quanto de **QmCi**). Se supormos $\alpha \rightarrow \beta \in \Delta$ e $\alpha \in \Delta$, deveremos ter também (por **MP**) $\beta \in \Delta$. Logo, se $\alpha \rightarrow \beta \in \Delta$, ou $\alpha \notin \Delta$ ou $\beta \in \Delta$.

Suponha que $\alpha \notin \Delta$ ou $\beta \in \Delta$. Temos dois possíveis casos:

1. Se $\beta \in \Delta$, pelo axioma **Mon** $\beta \rightarrow (\alpha \rightarrow \beta)$ e **MP** obtemos $\alpha \rightarrow \beta \in \Delta$.
2. Se $\beta \notin \Delta$, necessariamente $\alpha \notin \Delta$. Suponha, então que $\alpha \rightarrow \beta \notin \Delta$. Logo:

$$\begin{array}{l}
 \Delta, \alpha \vdash_{\mathcal{Q}} \gamma \\
 \Delta, \alpha \rightarrow \beta \vdash_{\mathcal{Q}} \gamma \\
 \Delta \vdash_{\mathcal{Q}} \alpha \rightarrow \gamma \\
 \Delta \vdash_{\mathcal{Q}} (\alpha \rightarrow \beta) \rightarrow \gamma \\
 \Delta \vdash_{\mathcal{Q}} (\alpha \vee (\alpha \rightarrow \beta)) \rightarrow \gamma \quad (\text{Ou2 e MP}) \\
 \Delta \vdash_{\mathcal{Q}} \alpha \vee (\alpha \rightarrow \beta) \quad (\text{OuI}) \\
 \Delta \vdash_{\mathcal{Q}} \gamma \quad (\text{MP})
 \end{array}$$

Por contradição, $\alpha \rightarrow \beta \in \Delta$.

- Partiremos agora para as regras da Definição 3.46. A regra νNeg deve valer tanto para teorias de **QmbC**, quanto de **QmCi**. Suponha então que $\alpha \notin \Delta$ e logo $\Delta, \alpha \vdash_{\mathcal{Q}} \gamma$. Se fosse o caso de $\neg\alpha \notin \Delta$, teríamos:

$$\begin{array}{l}
 \Delta \vdash_{\mathcal{Q}} \alpha \rightarrow \gamma \quad (\text{Teorema 3.29}) \\
 \Delta \vdash_{\mathcal{Q}} \neg\alpha \rightarrow \gamma \quad (\text{Teorema 3.29}) \\
 \Delta \vdash_{\mathcal{Q}} (\alpha \vee \neg\alpha) \rightarrow \gamma \quad (\text{Axioma Ou2 e MP}) \\
 \Delta \vdash_{\mathcal{Q}} \alpha \vee \neg\alpha \quad (\text{Axioma TND}) \\
 \Delta \vdash_{\mathcal{Q}} \gamma \quad (\text{Axioma MP})
 \end{array}$$

O que não é possível. Logo:

$$\alpha \notin \Delta \quad \implies \quad \neg\alpha \in \Delta$$

- Como no ítem anterior, νCon deve valer para teorias de **QmbC** e **QmCi**. Suponha que $\circ\alpha \in \Delta$ e $\alpha \in \Delta$. Logo, caso também tivéssemos $\neg\alpha \in \Delta$, pela

explosividade fraca dessas lógicas (codificada pelo Axioma **Exp**), teríamos, entre tantas outras, $\Delta \vdash_{\mathcal{Q}} \gamma$, absurdo. Portanto:

$$\circ\alpha \in \Delta \quad \Longrightarrow \quad \alpha \notin \Delta \quad \text{ou} \quad \neg\alpha \notin \Delta$$

- Este e o próximo ítem valem apenas para teorias de **QmCi**. Primeiramente νInc . Pelos Axiomas **Inc**, **E1** e **E2** devemos, necessariamente, ter:

$$\neg \circ\alpha \in \Delta \quad \Longrightarrow \quad \alpha \in \Delta \quad \text{e} \quad \neg\alpha \in \Delta$$

- Por último, $\nu CCon$. Como Δ é fechada sobre a relação de dedução de **QmCi** e temos o axioma **Con** presente nesse cálculo, para todo $n \geq 0$:

$$\circ\neg^n \circ\alpha \in \Delta$$

□

A partir da prova do Teorema 3.66 obtemos imediatamente o seguinte:

Corolário 3.67 (Propriedades das Teorias Maximais). *Seja Δ uma teoria de $\mathcal{Q} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$, maximal não trivial em relação a γ na lógica \mathcal{Q} . Então Δ satisfaz o seguinte:*

- (i) $\alpha \wedge \beta \in \Delta$ sse $\alpha \in \Delta$ e $\beta \in \Delta$;
- (ii) $\alpha \vee \beta \in \Delta$ sse $\alpha \in \Delta$ ou $\beta \in \Delta$;
- (iii) $\alpha \rightarrow \beta \in \Delta$ sse $\alpha \notin \Delta$ ou $\beta \in \Delta$;
- (iv) $\alpha \notin \Delta$ implica que $\neg\alpha \in \Delta$;
- (v) $\circ\alpha \in \Delta$ implica que $\alpha \notin \Delta$ ou $\neg\alpha \notin \Delta$.

No caso da lógica **QmCi** temos que, adicionalmente,

- (vi) $\neg\circ\alpha \in \Delta$ implica que $\alpha \in \Delta$ e $\neg\alpha \in \Delta$;
- (vii) $\circ\neg^n \circ\alpha \in \Delta$.

Teorema 3.68 (Teorema da completude). *As lógicas **QmbC** e **QmCi** são completas, respectivamente, para as semânticas de estruturas paraconsistentes dadas pelas Definições 3.48 e 3.49.*

Demonstração. Suponha que uma teoria Δ de uma lógica $\mathcal{Q} \in \{\mathbf{QmCi}, \mathbf{QmbC}\}$ é não trivial em relação a δ :

$$\Delta \not\vdash_{\mathcal{Q}} \delta.$$

Pelo Teorema 3.60, podemos encontrar uma extensão de Henkin Δ^H para Δ definida numa linguagem L_{ω} sobre a assinatura Σ_{ω} que a estende conservativamente,

isto é, não deduz nenhuma fórmula da linguagem original a mais. Por causa disto, Δ^H também é não trivial em relação a δ :

$$\Delta^H \not\vdash_{\mathcal{Q}} \delta .$$

Pelo Teorema 3.64, existe uma extensão $\overline{\Delta^H}$ de Δ^H não trivial em relação a δ que é maximal. Como $\overline{\Delta^H}$ está definida sobre a mesma linguagem L_{ω} de Δ^H , é também uma teoria de Henkin.

Pelo Teorema 3.61, podemos definir uma estrutura \mathfrak{D} para $\overline{\Delta^H}$, cujo domínio é formado pelo conjunto $FT_{\Sigma_{\omega}}$ dos termos fechados de L_{ω} , junto com uma bivaloração v tal que, para toda sentença φ da linguagem diagrama $L(\mathfrak{D})$ de \mathfrak{D} :

$$v(\varphi) = 1 \quad \iff \quad \overline{\Delta^H} \vdash_{\mathcal{Q}} \varphi^*$$

em que φ^* é a sentença de L_{ω} obtida de φ substituindo todas as ocorrências de constantes da forma \bar{t} (que nomeam termos fechados $t \in TF_{\Sigma}$, vistos como indivíduos de \mathfrak{D}) pelo próprio termo t . A valoração v satisfaz as cláusulas $vPred$ (da Definição 3.38) e vEx (da Definição 3.41).

Provaremos agora que v satisfaz o Lema da Substituição 3.51 e então as cláusulas $sNeg$ e $sCon$ serão automaticamente satisfeitas.

Fatos: Seja t um termo livre para a variável z na fórmula φ , e seja $b = (t_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^*$.

(i) $((u_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^* = (u_{\vec{x}, z}[\vec{a}; b])^*$, para todo termo $u \in T(\mathfrak{D})_{\vec{x}, z}$.

(ii) $((\varphi_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^* = (\varphi_{\vec{x}, z}[\vec{a}; b])^*$.

O item (i) é provado por indução na complexidade de u . Este fato já foi mencionado na prova do Teorema 3.51, observando que $u^{\widehat{\mathfrak{D}}} = u^*$ para todo termo u (ver prova do Teorema 3.58).

O item (ii) é provado por indução na complexidade de φ . Se φ é atômica, o resultado sai pelo item (i). A propagação da hipótese de indução pelos conectivos $\wedge, \vee, \rightarrow, \neg, \circ$ é óbvia. A propagação da hipótese de indução pelos quantificadores é uma consequência do fato de que t é livre para z em φ , logo x não ocorre em t no caso em que $\varphi = Qx\psi$, com $Q \in \{\forall, \exists\}$. Assim,

$$((Qx\psi)_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}] = Qx((\psi_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}]) .$$

Agora, sejam φ uma fórmula de $L(\mathfrak{D})$, t um termo livre para a variável z em φ e $b = (t_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^*$. Temos que

$$v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\varphi_z[t]) = v((\varphi_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])$$

e então

$$v_{\vec{x}, \vec{y}}^{\vec{a}, \vec{b}}(\varphi_z[t]) = 1 \quad \iff \quad \overline{\Delta^H} \vdash_{\mathcal{Q}} ((\varphi_z[t])_{\vec{x}, \vec{y}}[\vec{a}; \vec{b}])^* .$$

Por outro lado,

$$v_{\vec{x};z}^{\vec{a};b}(\varphi) = v(\varphi_{\vec{x};z}[\vec{a}; b])$$

portanto

$$v_{\vec{x};z}^{\vec{a};b}(\varphi) = 1 \quad \iff \quad \overline{\Delta^H} \vdash_{\mathcal{Q}} (\varphi_{\vec{x};z}[\vec{a}; b])^* .$$

Pelo **Fato(ii)**, temos que

$$v_{\vec{x};y}^{\vec{a};\vec{b}}(\varphi_z[t]) = v_{\vec{x};z}^{\vec{a};b}(\varphi)$$

e então as cláusulas *sNeg* e *sCon* são trivialmente satisfeitas.

Para provar que v satisfaz as Regras Proposicionais da Definição 3.41, assim como as cláusulas de paraconsistência da Definição 3.46 lembremos que, pela definição de v ,

$$v(\varphi) = 1 \quad \iff \quad \overline{\Delta^H} \vdash_{\mathcal{Q}} \varphi^* .$$

Mais ainda, pelo Teorema 3.65 temos que

$$v(\varphi) = 1 \quad \iff \quad \varphi^* \in \overline{\Delta^H} .$$

Pela prova do Teorema 3.58 sabemos que $(\varphi\#\psi)^* = (\varphi^*\#\psi^*)$ se $\# \in \{\wedge, \vee, \rightarrow\}$, e que $(\#\psi)^* = \#(\psi^*)$ se $\# \in \{\neg, \circ\}$. A partir disto, da definição de v e das propriedades de $\overline{\Delta^H}$ listadas no Corolário 3.67 obtemos imediatamente que v satisfaz as cláusulas *vE*, *vOu*, *vImp* (Definição 3.41), assim como *vNeg* e *vCon* (Definição 3.46). No caso da lógica subjacente ser **QmCi**, então v satisfaz, adicionalmente, as cláusulas *vInc* e *vCCon* da Definição 3.46.

A partir destas propriedades podemos provar facilmente que, se \sim denota a negação forte, então $v(\sim\varphi) = 1$ sse $v(\varphi) = 0$. Por outro lado, temos que $v(\forall x\varphi) = v(\sim\exists x\sim\varphi)$, pois $\vdash_{\mathcal{Q}} \forall x\varphi \rightarrow \sim\exists x\sim\varphi$ e $\vdash_{\mathcal{Q}} \sim\exists x\sim\varphi \rightarrow \forall x\varphi$ (e por causa do Teorema da Correção). Usando estas propriedades vemos facilmente que, dado que v satisfaz a cláusula *vEx* da Definição 3.41, então satisfaz a cláusula *vUni* desta Definição.

Temos assim que $\langle \mathfrak{D}, v \rangle$ é uma estrutura para $\mathcal{Q} \in \{\mathbf{QmCi}, \mathbf{QmbC}\}$ tal que, para toda $\gamma \in S_L$,

$$v(\gamma) = 1 \quad \iff \quad \gamma \in \overline{\Delta^H}$$

(pois $\gamma^* = \gamma$ se $\gamma \in S_L$). Isto é, pela definição de satisfatibilidade (Definição 3.39):

$$\mathfrak{D} \vDash \gamma \quad \iff \quad \gamma \in \overline{\Delta^H},$$

Dado que $\Delta \subseteq \overline{\Delta^H}$ então $\mathfrak{D} \vDash \Delta$. Por outro lado, $\delta \notin \overline{\Delta^H}$, portanto $\mathfrak{D} \not\vDash \delta$. Isto mostra que $\Delta \not\vDash \delta$.

Isto equivale, pelo Lema 3.63, à completude da lógica \mathcal{Q} para a semântica \mathcal{L} :

$$\Delta \vDash_{\mathcal{L}} \delta \quad \implies \quad \Delta \vdash_{\mathcal{Q}} \delta .$$

□

Dessa maneira fica provada a completude de **QmbC** e **QmCi** para as semânticas de bivalorações. Com isso temos um arcabouço semântico claro à nossa disposição que terá um papel importante na construção dos sequentes do próximo capítulo. O Teorema da Completude é parte constitutiva da demonstração de equivalência entre os formalismos originais das **LFI**'s e o dos sequentes, introduzidos a seguir.

Capítulo 4

Rumo à Programação Lógica Paraconsistente II: Um Teorema de Herbrand

Neste capítulo demonstraremos um resultado que está nos fundamentos da programação lógica: o Teorema de Herbrand. Conforme argumentamos na Seção 3.2, o papel de tal resultado para o presente projeto de pesquisa está em garantir a possibilidade de uma programação lógica genuinamente paraconsistente. Ele garante que há uma forma de reduzir a derivabilidade de uma lógica de primeira ordem a um caso mais simples: à derivabilidade em seu fragmento proposicional. Dessa maneira, para encontrar demonstrações (por um método automático, por exemplo) para o caso geral, pode-se restringir a busca às demonstrações proposicionais. Para a lógica clássica, são possíveis duas versões deste teorema: sua versão em relação a *insatisfatibilidade* e em relação a *derivabilidade*.

Conforme explicado na Seção 1.1 (Página 8), para a lógica clássica de primeira ordem, o teorema foi demonstrado primeiramente por Skolem, que fez uso da noção de *insatisfatibilidade*. Ou seja, Skolem demonstrou (de maneira não construtiva) que, para determinar se alguma fórmula de primeira ordem é insatisfatível, é necessário, e suficiente, determinar a insatisfatibilidade de uma fórmula livre de quantificadores e fechada. Tal coisa pode ser feita através da análise verofuncional por tabelas-verdade, por exemplo. Através do teorema da completude, pode-se chegar (por meios não construtivos novamente) à versão que apela à noção de *derivabilidade*, tendo em vista que a insatisfatibilidade é equivalente à validade universal (pela lei da dupla negação) que é equivalente a derivabilidade por meio do teorema da completude.

A vantagem da demonstração de Herbrand consiste em levar a cabo, por méto-

dos estritamente construtivos, a transformação de uma demonstração de uma fórmula de primeira ordem na demonstração de uma fórmula livre de quantificadores e fechada, com a possibilidade de fazer a transformação reversa construtivamente também. Logo, tal fórmula é uma *tautologia* e sua demonstrabilidade pode ser estabelecida por métodos totalmente proposicionais.

Nas **LFI**'s perde-se a dualidade que existe na lógica de primeira ordem clássica entre *insatisfatibilidade* e *validade*, conforme será ressaltado na Seção 5.1. Faz-se, então, necessário seguir métodos da teoria da prova para se chegar ao Teorema de Herbrand. Na sua forma clássica, o resultado pode ser obtido como uma consequência do teorema da eliminação do corte. Tendo isso em mente, primeiramente será obtida uma formulação por seqüentes para duas **LFI**'s fundamentais: **QmbC** e **QmCi**. Na Seção 4.1.4, será demonstrada a eliminação do corte para **QmbC** e, na Seção 4.2, um teorema de Herbrand restrito a um fragmento grande o suficiente para abarcar os programas lógicos estendidos definidos, introduzidos no Capítulo 3.1.

Até onde sabemos, são resultados novos tanto a formulação por seqüentes para lógicas baseadas em **mbC** e **mCi**, quanto o teorema de Herbrand para alguma **LFI** quantificada.

4.1 Seqüentes para **QmbC** e **QmCi**

A seguinte formulação se baseia na construção de **BC** e **CI**, conforme aparecem [Gen10] e [Gen07]. Tais cálculos são versões quantificadas, elaboradas por meio de seqüentes, para as lógicas da inconsistência formal **bC** e **ci**, respectivamente. Em tais artigos, tanto para **BC** quanto para **CI**, o autor substitui a regra clássica de introdução de negação à esquerda:

$$\frac{\Gamma \mapsto \Delta, \alpha}{\Gamma, \neg\alpha \mapsto \Delta},$$

por uma versão que demanda uma *fórmula de suporte* ($\circ\alpha$), à esquerda do seqüente da hipótese:

$$\frac{\Gamma, \circ\alpha \mapsto \Delta, \alpha}{\Gamma, \circ\alpha, \neg\alpha \mapsto \Delta}.$$

Demanda também, para **BC** e **CI**, a introdução de negações duplas à esquerda:

$$\frac{\Gamma, \alpha \mapsto \Delta}{\Gamma, \neg\neg\alpha \mapsto \Delta}.$$

Especificamente para **CI**, existem duas regras diferentes, uma para a introdução de negações de consistências à esquerda, e outra para a introdução de consistências à direita:

$$\frac{\Gamma \mapsto \Delta, \circ\alpha}{\Gamma, \neg \circ \alpha \mapsto \Delta} \text{ e } \frac{\alpha \wedge \neg\alpha, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \circ\alpha}.$$

Nesta seção, serão introduzidos os cálculos **MBC** e **MCI**, versões por sequentes de **QmbC** e **QmCi**. É utilizada, para os dois cálculos, a mesma regra de introdução de negação com suporte, tal como enunciada acima, ao invés da regra clássica de introdução de negação à esquerda. Não há regra para introdução de dupla negação.

Para **MCI**, ligeiras alterações das regras específicas para **CI** se fazem necessárias:

$$\frac{\Gamma \mapsto \Delta, \neg^n \circ \alpha}{\Gamma, \neg^{n+1} \circ \alpha \mapsto \Delta} \text{ e } \frac{\alpha, \neg\alpha, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \circ\alpha}.$$

4.1.1 Definição

Aqui vamos reformular os cálculos apresentados nas Seções 3.3.1 e 3.3.2 (*a la Hilbert*) através de sequentes. Note-se que o sentido intuitivo de se afirmar um sequente $\Gamma \mapsto \Delta$ é que alguma $\gamma \in \Gamma$ é falsa ou alguma $\delta \in \Delta$ é verdadeira.

Os sequentes aqui tratados são duplas de *multiconjuntos* finitos de fórmulas. Multiconjuntos são conjuntos que admitem a repetição de elementos, mas não levam em conta sua ordem. Representaremos a dupla de multiconjuntos Γ e Δ por $\Gamma \mapsto \Delta$. Começaremos com a contrapartida por sequentes de **QmbC** (**MBC**) e na sequência apresentaremos a extensão necessária para **QmCi** (**MCI**).

Definição 4.1 (MBC). Dada uma linguagem de primeira ordem L (veja-se Definição 3.16), os axiomas e regras de inferência de **MBC**, a versão por sequentes de **QmbC**, são os seguintes:

- **Axiomas**

$$\alpha \mapsto \alpha \quad (\mathbf{Ax})$$

Para qualquer fórmula $\alpha \in L$.

• **Enfraquecimento**

$$\frac{\Gamma \mapsto \Delta}{\alpha, \Gamma \mapsto \Delta} \text{ (Enf-E)} \quad \frac{\Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \alpha} \text{ (Enf-D)}$$

• **Contração**

$$\frac{\Gamma, \alpha, \alpha \mapsto \Delta}{\Gamma, \alpha \mapsto \Delta} \text{ (Cont-E)} \quad \frac{\Gamma \mapsto \Delta, \alpha, \alpha}{\Gamma \mapsto \Delta, \alpha} \text{ (Cont-D)}$$

• **Introdução de \rightarrow**

$$\frac{\Gamma \mapsto \Delta, \alpha \quad \beta, \Gamma' \mapsto \Delta'}{\alpha \rightarrow \beta, \Gamma, \Gamma' \mapsto \Delta, \Delta'} \text{ (}\rightarrow\text{E)} \quad \frac{\Gamma, \alpha \mapsto \beta, \Delta}{\Gamma \mapsto \alpha \rightarrow \beta, \Delta} \text{ (}\rightarrow\text{D)}$$

• **Introdução de \wedge**

$$\frac{\Gamma, \alpha, \beta \mapsto \Delta}{\Gamma, \alpha \wedge \beta \mapsto \Delta} \text{ (}\wedge\text{E)} \quad \frac{\Gamma \mapsto \alpha, \Delta \quad \Gamma' \mapsto \beta, \Delta'}{\Gamma, \Gamma' \mapsto \alpha \wedge \beta, \Delta, \Delta'} \text{ (}\wedge\text{D)}$$

• **Introdução de \vee**

$$\frac{\alpha, \Gamma \mapsto \Delta \quad \beta, \Gamma' \mapsto \Delta'}{\alpha \vee \beta, \Gamma, \Gamma' \mapsto \Delta, \Delta'} \text{ (}\vee\text{E)} \quad \frac{\Gamma \mapsto \Delta, \alpha, \beta}{\Gamma \mapsto \Delta, \alpha \vee \beta} \text{ (}\vee\text{D)}$$

- **Introdução de \neg ¹**

$$\frac{\circ\alpha, \Gamma \mapsto \Delta, \alpha}{\circ\alpha, \neg\alpha, \Gamma \mapsto \Delta} \quad (\neg E) \qquad \frac{\alpha, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \neg\alpha} \quad (\neg D)$$

- **Regras para Quantificadores**

$$\frac{\alpha_x[t], \Gamma \mapsto \Delta}{\forall x \alpha, \Gamma \mapsto \Delta} \quad (\forall E) \qquad \frac{\Gamma \mapsto \Delta, \alpha}{\Gamma \mapsto \Delta, \forall x \alpha} \quad (\forall D)$$

$$\frac{\alpha, \Gamma \mapsto \Delta}{\exists x \alpha, \Gamma \mapsto \Delta} \quad (\exists E) \qquad \frac{\Gamma \mapsto \Delta, \alpha_x[t]}{\Gamma \mapsto \Delta, \exists x \alpha} \quad (\exists D)$$

Para $\forall E$ e $\exists D$, t está restrito aos **termos** livres para x em α e pode aparecer ainda em $Qx\alpha$ (isto é, não é necessário que t esteja *completamente indicado*² em $\alpha_x[t]$). Já x é uma **variável** que não ocorre livre³ em Γ e Δ das regras $\forall D$ e $\exists E$, designada na literatura por *autovariável* (*Eigenvariable*).

- **Regra do Corte**

$$\frac{\Gamma \mapsto \Delta, \alpha \quad \alpha, \Gamma' \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} \quad (\text{Corte})$$

■

Na sequência, serão dados alguns exemplos para que fiquem claras as duas formas de se introduzir cada quantificador e a diferença entre um termo totalmente e não totalmente indicado.

¹É de se notar que **LK** (o cálculo de predicados clássico) pode ser obtido se substituirmos $\neg E$ por $\frac{\Gamma \mapsto \Delta, \alpha}{\Gamma, \neg\alpha \mapsto \Delta}$, cf. [Gen10, p.7].

²Cf. Definição 3.18 e [Tak75, p.8].

³Conforme [GTL89, p.32].

Exemplo 4.2. Seja P um símbolo de predicado ternário. Pode-se simbolizar $P(x, y, z)$ por φ e, logo, $\varphi_x[z] \equiv P(z, y, z)$. Portanto, é possível realizar a seguinte dedução:

$$\begin{aligned} P(z, y, z) \mapsto P(z, y, z) &\equiv \varphi_x[z] \mapsto \varphi_x[z] && (\text{Ax.}) \\ \forall x P(x, y, z) \mapsto P(z, y, z) &\equiv \forall x \varphi \mapsto \varphi_x[z] && (\forall E) \end{aligned}$$

Neste caso, diz-se que z não está totalmente indicada em $\varphi_x[z]$, dado que ocorre em φ .

Exemplo 4.3. Da mesma forma, é possível denominar $P(x, y, x)$ por ψ , $\psi_x[z] \equiv P(z, y, z)$ e continuar a dedução do exemplo anterior com a introdução de um existencial do lado direito, do seguinte modo:

$$\begin{aligned} \forall x P(x, y, z) \mapsto P(z, y, z) &\equiv \forall x \varphi \mapsto \psi_x[z] \\ \forall x P(x, y, z) \mapsto \exists x P(x, y, x) &\equiv \forall x \varphi \mapsto \exists x \psi && (\exists D) \end{aligned}$$

Neste caso, diz-se que z está totalmente indicada em $\psi_x[z]$. Note-se que não seria possível introduzir um $\forall y$ à direita com uma aplicação de $\forall D$, pois y está livre à esquerda do último sequente. No entanto, é possível introduzir um $\exists z$ à esquerda:

$$\begin{aligned} \forall x P(x, y, z) \mapsto \exists x P(x, y, x) &\equiv \forall x \varphi \mapsto \exists x \psi \\ \exists z \forall x P(x, y, z) \mapsto \exists x P(x, y, x) &\equiv \exists z \forall x \varphi \mapsto \exists x \psi && (\exists E) \end{aligned}$$

dado que tal variável não ocorre à direita (bastaria que não ocorresse livre).

Definição 4.4. MCI estende MBC pelas seguintes regras:

$$\frac{\Gamma \mapsto \Delta, \neg^n \circ \alpha}{\neg^{n+1} \circ \alpha, \Gamma \mapsto \Delta} \quad (\neg \circ E) \qquad \frac{\alpha, \neg \alpha, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \circ \alpha} \quad (\circ D)$$

■

Teorema 4.5 (Corte da Negação). *Em MBC e MCI são possíveis as seguintes derivações:*

$$\frac{\circ \alpha, \Gamma \mapsto \Delta, \neg \alpha}{\circ \alpha, \alpha, \Gamma \mapsto \Delta} \qquad \frac{\neg \alpha, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \alpha} .$$

Demonstração.

$$\frac{\circ\alpha, \Gamma \mapsto \Delta, \neg\alpha \quad \frac{\circ\alpha, \alpha \mapsto \alpha}{\circ\alpha, \alpha, \neg\alpha \mapsto} : \neg E}{\circ\alpha, \alpha, \Gamma \mapsto \Delta} : \text{Corte de } \neg\alpha \text{ e Cont-E em } \circ\alpha$$

$$\frac{\neg\alpha, \Gamma \mapsto \Delta \quad \frac{\alpha \mapsto \alpha}{\mapsto \alpha, \neg\alpha} : \neg D}{\Gamma \mapsto \Delta, \alpha} : \text{Corte de } \neg\alpha$$

□

4.1.2 Completude dos Sequentes em relação à Formulação Hilbertiana

Para que fique claro que nossa formulação por sequentes é adequada, devemos demonstrar sua equivalência com relação às **LFI**'s em questão. Por completude dos sequentes em relação aos cálculos hilbertianos, queremos dizer que tudo o que é derivado por meio dos axiomas e das regras de inferência hilbertianos é possível ser derivado também usando sequentes.

Se $\mathcal{L} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$, teremos respectivamente $\mathfrak{S} \in \{\mathbf{MBC}, \mathbf{MCI}\}$ e, ainda respectivamente, \mathcal{E} será a semântica das estruturas para **QmbC** ou **QmCi**. Representaremos por Γ° um subconjunto finito de Γ . Como completude, demonstraremos o seguinte resultado:

$$\Gamma \vdash_{\mathcal{L}} \varphi \quad \Longrightarrow \quad \text{para algum } \Gamma^\circ: \quad \vdash_{\mathfrak{S}} \Gamma^\circ \mapsto \varphi.$$

Note que, devido à compacidade de \mathcal{L} e ao teorema da dedução, isso é equivalente a demonstrar, para todo Γ finito:

$$\vdash_{\mathcal{L}} \bigwedge \Gamma \rightarrow \varphi \quad \Longrightarrow \quad \vdash_{\mathfrak{S}} \Gamma \mapsto \varphi$$

A correção dos sequentes será estabelecida em relação à semântica de valorações paraconsistentes na Seção 4.1.3:

$$\vdash_{\mathfrak{S}} \Gamma \mapsto \varphi \quad \Longrightarrow \quad \Gamma \models_{\mathcal{L}} \varphi$$

No Corolário 4.12 abaixo, obteremos uma nova prova do Teorema de Correção 3.56, como uma consequência da completude dos sequentes em relação aos sistemas hilbertianos e de sua correção em relação à semântica de valorações. E então, pela completude das semânticas \mathcal{E} em relação às lógicas \mathcal{L} , provada na Seção 3.3.4, tem-se a equivalência entre os três sistemas dedutivos.

A completude dos seqüentes será provada por indução no tamanho das derivações em \mathfrak{L} que terminem na fórmula $\wedge \Gamma \rightarrow \varphi$.

Teorema 4.6. *Os axiomas de **QmbC** e **QmCi** (Definições 3.6, 3.11 e 3.21) são demonstráveis por seqüentes (Definição 4.1).*

Demonstração. Note que os axiomas **Inc** e **Con** são demonstrados apenas em **MCI** e que podem ser feitas permutações à vontade, pois seqüentes duplas de *multiconjuntos*.

Ax. **OuI**

$$\begin{array}{ll}
 \alpha \mapsto \alpha & (\text{Ax}) \\
 \alpha \mapsto \alpha, \beta & (\text{Enf-D}) \\
 \alpha \mapsto \beta, \alpha & \\
 \mapsto \alpha \rightarrow \beta, \alpha & (\rightarrow D) \\
 \mapsto \alpha, \alpha \rightarrow \beta & \\
 \mapsto \alpha \vee (\alpha \rightarrow \beta) & (\vee D)
 \end{array}$$

Ax. **TND**

$$\begin{array}{ll}
 \alpha \mapsto \alpha & (\text{Ax}) \\
 \mapsto \alpha, \neg \alpha & (\neg D) \\
 \mapsto \alpha \vee \neg \alpha & (\vee D)
 \end{array}$$

Ax. **Exp**

$$\begin{array}{ll}
 \alpha \mapsto \alpha & (\text{Ax}) \\
 \circ \alpha, \alpha \mapsto \alpha & (\text{Enf-E}) \\
 \circ \alpha, \alpha \mapsto \alpha, \beta & (\text{Enf-D}) \\
 \circ \alpha, \alpha, \neg \alpha \mapsto \beta & (\neg E) \\
 \mapsto \circ \alpha \rightarrow \alpha \rightarrow \neg \alpha \rightarrow \beta & (\rightarrow D)
 \end{array}$$

Ax. **Inc**

$$\begin{array}{ll}
 \alpha \wedge \neg \alpha \mapsto \alpha \wedge \neg \alpha & (\text{Ax}) \\
 \mapsto \circ \alpha, \alpha \wedge \neg \alpha & (\circ D) \\
 \neg \circ \alpha \mapsto \alpha \wedge \neg \alpha & (\neg \circ E) \\
 \mapsto \neg \circ \alpha \rightarrow (\alpha \wedge \neg \alpha) & (\rightarrow D)
 \end{array}$$

Ax. **Con**

$$\begin{aligned} \neg^n \circ \alpha &\vdash \neg^n \circ \alpha && \text{(Ax)} \\ \neg^n \circ \alpha, \neg^{n+1} \circ \alpha &\vdash && (\neg \circ E) \\ &\vdash \circ \neg^n \circ \alpha && (\circ D) \end{aligned}$$

Ax. \exists -Ax

$$\begin{aligned} \alpha_x[t] &\vdash \alpha_x[t] && \text{(Ax)} \\ \alpha_x[t] &\vdash \exists x \alpha && (\exists D) \\ &\vdash \alpha_x[t] \rightarrow \exists x \alpha && (\rightarrow D) \end{aligned}$$

Ax. \forall -Ax

$$\begin{aligned} \alpha_x[t] &\vdash \alpha_x[t] && \text{(Ax)} \\ \forall x \alpha &\vdash \alpha_x[t] && (\forall E) \\ &\vdash \forall x \alpha \rightarrow \alpha_x[t] && (\rightarrow D) \end{aligned}$$

Ax. **PQ**

Primeiramente, temos que $\perp \vdash$ é teorema ($\perp \equiv \beta \wedge \neg\beta \wedge \circ\beta$):

$$\begin{aligned} \beta &\vdash \beta && \text{(Ax.)} \\ \beta, \circ\beta &\vdash \beta && \text{(Enf-E)} \\ \beta, \circ\beta, \neg\beta &\vdash && (\neg E) \end{aligned}$$

Agora:

$$\begin{aligned} \alpha &\vdash \alpha && \text{(Ax.)} \\ \alpha &\vdash \perp, \alpha && \text{(Enf-D)} \\ &\vdash \alpha \rightarrow \perp, \alpha && (\rightarrow D) \\ &\vdash \exists x(\alpha \rightarrow \perp), \alpha && (\exists D) \\ &\vdash \exists x(\alpha \rightarrow \perp), \forall x\alpha && (\forall D) \\ &\vdash \exists x(\alpha \rightarrow \perp), \forall x\alpha && \perp \vdash \\ &\forall x\alpha \rightarrow \perp &\vdash \exists x(\alpha \rightarrow \perp) && (\rightarrow E) \\ &\vdash (\forall x\alpha \rightarrow \perp) \rightarrow \exists x(\alpha \rightarrow \perp) && (\rightarrow D) \end{aligned}$$

□

Para demonstrar a completude de **MBC** e **MCI**, faz-se necessário que nesses formalismos também seja possível simular as regras de inferência de **QmbC** e **QmCi** (que são as mesmas para os dois cálculos). Para tanto, faremos uso de um lema intermediário:

Teorema 4.7. *Eliminação do conectivo “ \rightarrow ”:*

$$\Gamma \mapsto \alpha \rightarrow \beta \quad \vdash_{\varepsilon} \quad \Gamma, \alpha \mapsto \beta.$$

Demonstração. Com efeito, suponha $\Gamma \mapsto \alpha \rightarrow \beta$. Usaremos, então, a regra $\rightarrow E$ com os axiomas $\alpha \mapsto \alpha$ e $\beta \mapsto \beta$:

$$\frac{\alpha \mapsto \alpha \quad \beta \mapsto \beta}{\alpha \rightarrow \beta, \alpha \mapsto \beta}$$

Por fim, com o **Corte** desse último sequente com a hipótese, obtemos o resultado desejado:

$$\frac{\Gamma \mapsto \alpha \rightarrow \beta \quad \alpha \rightarrow \beta, \alpha \mapsto \beta}{\Gamma, \alpha \mapsto \beta}$$

□

Teorema 4.8. ***MBC** e **MCI** podem simular as regras de inferência de **QmbC** e **QmCi** (vejam-se as Definições 3.6 e 3.21).*

Demonstração. São apenas três as regras, comuns a **QmbC** e **QmCi**, que devemos ser capazes de simular nossos sequentes:

1. Modus Ponens

Suponha:

$$\begin{array}{l} \mapsto \alpha \\ \mapsto \alpha \rightarrow \beta \end{array}$$

Logo:

$$\frac{\begin{array}{l} \mapsto \alpha \rightarrow \beta \\ \alpha \mapsto \beta \end{array} \quad \begin{array}{l} \text{Lema 4.7} \\ \text{Hipótese} \end{array}}{\mapsto \beta} \quad \text{Corte}$$

2. Introdução de Existencial

Suponha, sem que x ocorra livre em β :

$$\vdash \alpha \rightarrow \beta$$

Logo

$$\frac{\begin{array}{l} \vdash \alpha \rightarrow \beta \\ \alpha \vdash \beta \\ \hline \exists x \alpha \vdash \beta \end{array}}{\vdash \exists x \alpha \rightarrow \beta} \quad \begin{array}{l} \text{Lema 4.7} \\ \exists\text{-E} \\ \rightarrow\text{-D} \end{array}$$

3. Introdução de Universal

Suponha, sem que x ocorra livre em α :

$$\vdash \alpha \rightarrow \beta$$

Logo

$$\frac{\begin{array}{l} \vdash \alpha \rightarrow \beta \\ \alpha \vdash \beta \\ \hline \alpha \vdash \forall x \beta \end{array}}{\vdash \alpha \rightarrow \forall x \beta} \quad \begin{array}{l} \text{Lema 4.7} \\ \forall\text{-D} \\ \rightarrow\text{-D} \end{array}$$

□

Temos, então, a completude dos formalismo por sequentes em relação às **LFI**'s:

Corolário 4.9 (Completude dos Sequentes). *Seja $\mathfrak{S} \in \{\mathbf{MBC}, \mathbf{MCI}\}$ e, respectivamente, $\mathfrak{Q} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$. Então:*

$$\Gamma \vdash_{\mathfrak{Q}} \varphi \quad \Longrightarrow \quad \text{para algum } \Gamma^{\circ} \subseteq \Gamma \text{ finito: } \vdash_{\mathfrak{S}} \Gamma^{\circ} \vdash \varphi$$

Os axiomas usados em demonstrações em **MBC** podem se limitar, sem que se percam demonstrações, a três tipos de fórmulas, enquanto os de **MCI** a dois tipos.

Teorema 4.10. *MBC não perde demonstrações se seus axiomas se limitarem aos seguintes casos: (“At” indica uma fórmula atômica e “ φ ” uma fórmula qualquer):*

1.

$$At \vdash At$$

2.

$$\neg\varphi \mapsto \neg\varphi$$

3.

$$\circ\varphi \mapsto \circ\varphi$$

E para MCI vale o mesmo com a limitação de seus axiomas aos itens 1 e 2 acima.

Demonstração. A demonstração é por dupla indução no número de instâncias, em uma demonstração qualquer, de axiomas com a fórmula complexa e na complexidade dessas fórmulas. O passo indutivo principal consiste em substituir o uso de uma instância do axioma com a fórmula em questão por demonstrações que terminem nessa fórmula e que façam uso apenas de axiomas com fórmulas de complexidade menor. Temos 6 casos, conforme a fórmula φ na instância de “ $\varphi \mapsto \varphi$ ” seja:

1. $\alpha \rightarrow \beta$

$$\begin{array}{l} \alpha \mapsto \alpha \quad \beta \mapsto \beta \quad \text{Hipóteses de Indução} \\ \alpha \rightarrow \beta, \alpha \mapsto \beta \quad \quad \quad \rightarrow E \\ \hline \alpha \rightarrow \beta \mapsto \alpha \rightarrow \beta \quad \quad \quad \rightarrow D \end{array}$$

2. $\alpha \wedge \beta$

$$\begin{array}{l} \alpha \mapsto \alpha \quad \beta \mapsto \beta \quad \text{Hipóteses de Indução} \\ \alpha \wedge \beta \mapsto \alpha \quad \alpha \wedge \beta \mapsto \beta \quad \quad \quad \mathbf{Enf-E} \text{ e } \wedge E \\ \hline \alpha \wedge \beta \mapsto \alpha \wedge \beta \quad \quad \quad \wedge D \text{ e } \mathbf{Cont-E} \end{array}$$

3. $\alpha \vee \beta$

$$\begin{array}{l} \alpha \mapsto \alpha \quad \beta \mapsto \beta \quad \text{Hipóteses de Indução} \\ \alpha \mapsto \alpha \vee \beta \quad \beta \mapsto \alpha \vee \beta \quad \quad \quad \mathbf{Enf-D} \text{ e } \vee E \\ \hline \alpha \vee \beta \mapsto \alpha \vee \beta \quad \quad \quad \vee E \text{ e } \mathbf{Cont-D} \end{array}$$

4. $\forall x \alpha$

$$\begin{array}{l} \alpha \mapsto \alpha \quad \text{Hipótese de Indução} \\ \forall x \alpha \mapsto \alpha \quad \quad \quad \forall E \\ \forall x \alpha \mapsto \forall x \alpha \quad \quad \quad \forall D \end{array}$$

5. $\exists x \alpha$

$$\begin{array}{ll} \alpha \mapsto \alpha & \text{Hipótese de Indução} \\ \alpha \mapsto \exists x \alpha & \exists D \\ \exists x \alpha \mapsto \exists x \alpha & \exists E \end{array}$$

6. $\circ\alpha$ (apenas para MCI)

$$\begin{array}{ll} \alpha \mapsto \alpha & \text{Hipótese de Indução} \\ \circ\alpha, \alpha \mapsto \alpha & \text{Enf-E} \\ \circ\alpha, \alpha, \neg\alpha \mapsto & \neg E \\ \circ\alpha \mapsto \circ\alpha & \circ D \end{array}$$

□

Note que por esse processo não foram introduzidos cortes a mais.

4.1.3 Correção em Relação às Valorações Paraconsistentes

Conforme \mathfrak{S} seja **MBC** ou **MCI**, \mathcal{E} será, respectivamente, a semântica de estruturas para **QmbC** ou **QmCi**. Demonstraremos, então, que toda vez que:

$$\vdash_{\mathfrak{S}} \Gamma \mapsto \varphi,$$

teremos também:

$$\Gamma \vDash_{\mathcal{E}} \varphi.$$

Teorema 4.11 (Correção dos Sequentes). *Todo sequente $\Gamma \mapsto \Delta$ derivado em \mathfrak{S} tem a propriedade de que, em toda estrutura \mathcal{E} para a linguagem das fórmulas do sequente, ou alguma fórmula de Γ é falsa, ou alguma fórmula de Δ é verdadeira. Logo:*

$$\vdash_{\mathfrak{S}} \Gamma \mapsto \varphi \quad \Longrightarrow \quad \Gamma \vDash_{\mathcal{E}} \varphi.$$

Demonstração. Pode-se ver que os axiomas das Definições 4.1 e 4.4 obedecem tal propriedade e suas regras a preservam. □

Corolário 4.12 (Correção das LFI's em Relação às Semânticas Paraconsistentes). *Seja $\mathfrak{Q} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$ e, respectivamente, \mathcal{E} a semântica de bivalorações para as lógicas **QmbC** e **QmCi**. Então:*

$$\Gamma \vdash_{\mathfrak{Q}} \varphi \quad \Longrightarrow \quad \Gamma \vDash_{\mathcal{E}} \varphi.$$

Demonstração. Com efeito, suponha que:

$$\Gamma \not\vdash_{\mathcal{E}} \varphi .$$

Logo, temos que, para todo Γ° , subconjunto finito de Γ :

$$\Gamma^\circ \not\vdash_{\mathcal{E}} \varphi ,$$

pois todo modelo de Γ é também modelo de Γ° . Logo, pela correção dos sequentes, para todo $\Gamma^\circ \subseteq \Gamma$ finito:

$$\not\vdash_{\mathcal{E}} \Gamma^\circ \mapsto \varphi$$

e, pela sua completude:

$$\Gamma \not\vdash_{\mathcal{Q}} \varphi .$$

Com isso, temos a correção de \mathcal{Q} em relação à semântica paraconsistente \mathcal{E} :

$$\Gamma \vdash_{\mathcal{Q}} \varphi \quad \Longrightarrow \quad \Gamma \vDash_{\mathcal{E}} \varphi$$

□

4.1.4 Eliminação do Corte para QmbC

A eliminação do corte consiste em mostrar que o corte é, em princípio, *dispensável*. Isto é, em sistemas para os quais vale o teorema da eliminação do corte, pode-se excluir essa regra sem que se percam teoremas, em troca de um aumento superexponencial no comprimento das demonstrações:

Roughly speaking, a cut-free proof is a proof from which all formulas which are “too general” have been banished. General formulas in a proof carry the ideas of the proof: it is only because we have general formulas, that we can have short and understandable proofs; when (in the Hauptsatz) we eliminate all these general formulas, we increase the length and obscurity of the proof: for that reason, cut-free proofs are unlikely objects for mathematical practice. Their interest lies somewhere else: these proofs are very interesting to study, because their general properties are very important. [Gir87, p.95]

Quanto às propriedades interessantes de demonstrações sem corte, faremos uso delas mais adiante, quando for demonstrado o teorema de Herbrand para uma das **LFI**s no Capítulo 4.2.

Podemos ainda entender o resultado de outra maneira: que os sistemas obtidos pela supressão da regra de corte são fechados em relação a ela, ou seja, a regra de

corte é *redundante*. No entanto, se admitimos axiomas não lógicos, nem sempre é possível eliminar todos os cortes (sem que percamos possíveis demonstrações feitas a partir de tais axiomas).

A demonstração começa com os casos chaves (Lema 4.17): nele demonstra-se que o corte de uma fórmula complexa pode ser trocado por cortes sobre fórmulas de menor complexidade, quando as duas subprovas de ambos os sequentes das premissas deste corte terminem em uma regra lógica que introduza o conectivo principal da fórmula de corte. O Lema Principal (4.19) demonstra a possibilidade de se fazer o que ficou conhecido como *multi-corte*. A eliminação dos cortes feita indutivamente esbarra em um problema quando as últimas regras de inferência antes de um corte são contrações: a subprova contém mais instâncias da fórmula a ser cortada que a prova. Esse caso em particular levou Gentzen, em [Gen35], a introduzir os *multi-cortes*. Tal regra, que permite cortes simultâneos, é equivalente à regra de corte simples. A demonstração original consistia, na verdade, em demonstrar que os multi-cortes poderiam ser eliminados (cf. [TS96, 4.1.4]).

Algumas definições são necessárias a fim de serem usadas como variáveis de indução. A demonstração apresentada aqui foi adaptada de [GTL89] seguindo os métodos de [Gen10] para lidar com as características não clássicas.

Definição 4.13 (Grau de uma Fórmula). O grau de uma fórmula é o número total de ocorrências de conectivos nela contidos⁴. Logo, se uma fórmula tem grau 0, se trata de uma fórmula atômica.

■

Definição 4.14 (Grau do Corte e Grau da Demonstração). O grau de um corte é o grau da fórmula que está sendo cortada. O grau de uma demonstração é o grau do corte de maior grau.

■

As demonstrações por sequentes podem ser vistas como árvores, em que cada nó tem um ou dois filhos e a raiz é a última regra aplicada. As folhas da árvore são os axiomas usados na demonstração (ou hipóteses, quando admitidas).

Definição 4.15 (Altura da Demonstração). A altura de uma demonstração é a altura de sua árvore. Convencionaremos que demonstrações constituídas unicamente por axiomas terão altura 0.

■

⁴Nesta dissertação nos referimos, em capítulos anteriores, à *complexidade* de uma fórmula como sinônimo de *grau*.

Definição 4.16. Vamos simbolizar por $\Gamma - \varphi$ o multiconjunto obtido a partir de Γ pela supressão de todas as fórmulas φ .

■

Ao longo da demonstração, em vários momentos, certas aplicações de regras ou demonstrações serão substituídas por outras. Quando ocorrer esse tipo de substituição, estará indicado por um sinal de dupla barra. Por exemplo:

$$\frac{\varpi}{\varpi'},$$

indica que a demonstração ϖ foi substituída pela demonstração ϖ' .

Casos Chaves

O objetivo é eliminar cortes:

$$\frac{\Gamma \mapsto \Delta, \varphi \quad \varphi, \Gamma' \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} \quad : \text{Corte de } \varphi$$

em que φ é a fórmula principal das últimas regras nas duas subprovas das premissas e que essas regras são lógicas. Tais cortes são substituídos por demonstrações das mesmas conclusões que fazem uso de cortes com fórmulas de menor complexidade.

Lema 4.17 (Casos Chaves). *Seja ϖ uma demonstração que termine em um corte sobre uma fórmula φ que tenha acabado de ser introduzida por regras lógicas nos dois sequentes do corte. Será encontrada uma demonstração ϖ' que conclua o mesmo sequente de ϖ e que introduza cortes apenas sobre fórmulas de grau estritamente menor que o de φ . Tal procedimento será simbolizado por: $\frac{\varpi}{\varpi'}$.*

Demonstração. Procederemos de diferentes maneiras, conforme φ seja:

1. $\alpha \wedge \beta$

$$\frac{\frac{\Gamma \mapsto \alpha, \Delta \quad \Gamma' \mapsto \beta, \Delta'}{\Gamma, \Gamma' \mapsto \alpha \wedge \beta, \Delta, \Delta'} \quad : \wedge D \quad \frac{\Gamma'', \alpha, \beta \mapsto \Delta''}{\Gamma'', \alpha \wedge \beta \mapsto \Delta''} \quad : \wedge E}{\Gamma, \Gamma', \Gamma'' \mapsto \Delta, \Delta', \Delta''} \quad : \text{Corte de } \alpha \wedge \beta$$

$$\frac{\Gamma' \mapsto \beta, \Delta' \quad \frac{\Gamma \mapsto \alpha, \Delta \quad \Gamma'', \alpha, \beta \mapsto \Delta''}{\Gamma, \Gamma'', \beta \mapsto \Delta, \Delta''} \quad : \text{Corte de } \alpha}{\Gamma, \Gamma', \Gamma'' \mapsto \Delta, \Delta', \Delta''} \quad : \text{Corte de } \beta$$

2. $\neg\alpha$

$$\frac{\frac{\Gamma, \alpha \mapsto \Delta}{\Gamma \mapsto \neg\alpha, \Delta} : \neg D \quad \frac{\Gamma', \circ\alpha \mapsto \alpha, \Delta'}{\Gamma', \circ\alpha, \neg\alpha \mapsto \Delta'} : \neg E}{\Gamma, \Gamma', \circ\alpha \mapsto \Delta, \Delta'} : \text{Corte de } \neg\alpha$$

$$\frac{\Gamma', \circ\alpha \mapsto \alpha, \Delta' \quad \Gamma, \alpha \mapsto \Delta}{\Gamma, \Gamma', \circ\alpha \mapsto \Delta, \Delta'} : \text{Corte de } \alpha$$

 3. $\forall x \alpha$

$$\frac{\frac{\Gamma \mapsto \Delta, \alpha}{\Gamma \mapsto \Delta, \forall x \alpha} : \forall D \quad \frac{\Gamma', \alpha_x[t] \mapsto \Delta'}{\Gamma', \forall x \alpha \mapsto \Delta'} : \forall E}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} : \text{Corte de } \forall x \alpha$$

$$\frac{\Gamma \mapsto \Delta, \alpha_x[t] \quad \Gamma', \alpha_x[t] \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} : \text{Corte de } \alpha_x[t]$$

E obtemos a demonstração de $\Gamma \mapsto \Delta, \alpha_x[t]$ (usada no **Corte** de $\alpha_x[t]$), a partir da substituição de x por t por toda a subprova de $\Gamma \mapsto \Delta, \alpha$, que é a hipótese de $\forall D$ (usada no **Corte** de $\forall x \alpha$). Uma vez que x não ocorre livre em Γ e Δ (condições da regra $\forall D$), a substituição não altera tais conjuntos.

 4. $\exists x \alpha$

$$\frac{\frac{\Gamma \mapsto \Delta, \alpha_x[t]}{\Gamma \mapsto \Delta, \exists x \alpha} : \exists D \quad \frac{\Gamma', \alpha \mapsto \Delta'}{\Gamma', \exists x \alpha \mapsto \Delta'} : \exists E}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} : \text{Corte de } \exists x \alpha$$

$$\frac{\Gamma \mapsto \Delta, \alpha_x[t] \quad \Gamma', \alpha_x[t] \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} : \text{Corte de } \alpha_x[t]$$

E obtemos a demonstração de $\Gamma', \alpha_x[t] \mapsto \Delta'$ (usada no **Corte** de $\alpha_x[t]$), a partir da substituição de x por t por toda a subprova de $\Gamma', \alpha \mapsto \Delta'$, que é a hipótese de $\exists E$ (usada no **Corte** de $\exists x \alpha$). Como no caso anterior, a substituição não altera Γ' e Δ' , pois x não ocorre livre nesses conjuntos.

As demonstrações dos outros dois casos ($\varphi \in \{\alpha \rightarrow \beta, \alpha \vee \beta\}$) não diferem em nada do caso em que $\varphi = \alpha \wedge \beta$. \square

Exemplo 4.18. Considere a seguinte demonstração que termina em um **Corte** de $\exists xA$, em que A é um predicado monádico:

$$\frac{\frac{A(t) \mapsto A(t)}{A(t) \mapsto \exists xA(x)} : \exists D \quad \frac{\frac{A(x) \mapsto A(x)}{A(x), \circ A(x) \mapsto A(x)} : \text{Enf-E} \quad \frac{A(x), \circ A(x), \neg A(x) \mapsto}{A(x), \forall x \circ A(x), \forall x \neg A(x) \mapsto} : \forall E}{\frac{A(x), \forall x \circ A(x), \forall x \neg A(x) \mapsto}{\exists xA(x), \forall x \circ A(x), \forall x \neg A(x) \mapsto} : \exists E}}{A(t), \forall x \circ A(x), \forall x \neg A(x) \mapsto} : \exists E$$

Ela pode ser substituída pela seguinte demonstração, que termina em um corte de $A(t)$:

$$\frac{A(t) \mapsto A(t) \quad \frac{\frac{A(t) \mapsto A(t)}{A(t), \circ A(t) \mapsto A(t)} : \text{Enf-E} \quad \frac{A(t), \circ A(t), \neg A(t) \mapsto}{A(t), \forall x \circ A(x), \forall x \neg A(x) \mapsto} : \forall E}}{A(t), \forall x \circ A(x), \forall x \neg A(x) \mapsto} : \exists E$$

Note-se que as únicas regras que poderiam ser quebradas pelas substituições de t por x , feitas nos ítems 3 ou 4 do Lema 4.17, seriam $\forall D$ ou $\exists E$, respectivamente:

$$\frac{\Gamma \mapsto \Delta, \varphi[x]}{\Gamma \mapsto \Delta, \forall x \varphi[x]} : \forall D \quad \frac{\Gamma', \varphi[x] \mapsto \Delta'}{\Gamma', \exists x \varphi[x] \mapsto \Delta'} : \exists E,$$

pois ficariam desta maneira:

$$\frac{\Gamma \mapsto \Delta, \varphi[t]}{\Gamma \mapsto \Delta, \forall x \varphi[x]} : \forall D! \quad \frac{\Gamma', \varphi[t] \mapsto \Delta'}{\Gamma', \exists x \varphi[x] \mapsto \Delta'} : \exists E!$$

e as regras não seriam aplicáveis nesses casos (por isso, as marcamos com “!”). Porém, como pudemos ver no exemplo acima, em todos os sequentes das sub-provas a serem alteradas pela substituição, há sempre alguma fórmula com a variável x livre, o que garante que não pode ocorrer aplicações de tais regras nas sub-provas.

Lema Principal

Lema 4.19 (Lema Principal). *Seja φ uma fórmula de grau d , e π e π' provas de $\Gamma \mapsto \Delta$ e de $\Gamma' \mapsto \Delta'$ com grau estritamente menor que d . Podemos construir uma prova ϖ de $\Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'$ de grau estritamente menor que d :*

$$\begin{array}{c}
 \begin{array}{ccc}
 \pi & & \pi' \\
 \vdots \vdots \vdots \vdots & & \vdots \vdots \vdots \vdots \\
 \Gamma \mapsto \Delta & & \Gamma' \mapsto \Delta'
 \end{array} \\
 \hline \hline
 \varpi \\
 \vdots \vdots \vdots \vdots \\
 \Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'
 \end{array}$$

Demonstração. Constrói-se ϖ por indução em $h(\pi) + h(\pi')$. As regras de sequentes podem ter uma ou duas premissas. Assim, se chamarmos a última regra da demonstração π de r , sua(s) premissa(s) será(serão) designadas por $\Gamma_i \mapsto \Delta_i$, com $i = 1$ ou $i \in \{1, 2\}$. Paralelamente, a última regra de π' será r' com premissa(s) $\Gamma'_j \mapsto \Delta'_j$, $j = 1$ ou $j \in \{1, 2\}$.

Portanto, conforme tenhamos uma ou duas premissas em π ou π' , I e J serão respectivamente o número de premissas de cada demonstração:

$$\begin{array}{ccc}
 \begin{array}{c}
 \pi \\
 \vdots \vdots \vdots \vdots \\
 \begin{array}{cc}
 \pi_1 & \pi_I \\
 \vdots & \vdots \\
 \Gamma_1 \mapsto \Delta_1 & \Gamma_I \mapsto \Delta_I
 \end{array} \\
 \hline
 \Gamma \mapsto \Delta
 \end{array}
 & :r &
 \begin{array}{c}
 \pi' \\
 \vdots \vdots \vdots \vdots \\
 \begin{array}{cc}
 \pi'_1 & \pi'_J \\
 \vdots & \vdots \\
 \Gamma'_1 \mapsto \Delta'_1 & \Gamma'_J \mapsto \Delta'_J
 \end{array} \\
 \hline
 \Gamma' \mapsto \Delta'
 \end{array}
 & :r' &
 \end{array}$$

Seguem-se, então, todos os casos para encontrar a demonstração ϖ de $\Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'$:

1. π é um axioma. Temos dois casos:

- π é $\varphi \mapsto \varphi$. Basta tomar ϖ como a demonstração de $\varphi, \Gamma' - \varphi \mapsto \Delta'$ feita a partir de π' por meio de regras estruturais. Note-se que $g(\varpi) = g(\pi') < g(\varphi)$.
- π é $\alpha \mapsto \alpha$, com $\alpha \neq \varphi$. Tomemos então ϖ como a demonstração de $\alpha, \Gamma' - \varphi \mapsto \alpha, \Delta'$ por meio de enfraquecimentos tendo $\alpha \mapsto \alpha$ como axioma. Obviamente $g(\varpi) = 0$.

2. π' é um axioma. A construção de ϖ é similar ao item 1.

3. r é regra estrutural. Pela hipótese de indução para π_1 e π' , há uma demonstração ϖ_1 de $\Gamma_1, \Gamma' - \varphi \mapsto \Delta_1 - \varphi, \Delta'$. Obtem-se, então, ϖ a partir de ϖ_1 por regras estruturais. Note-se que se r for **Cont-D** sobre φ , haverá um número

maior de ocorrências de φ em Δ_1 do que em Δ . Neste caso, quando há exatamente uma ocorrência de φ em Δ_1 , pode-se já tomar $\varpi_1 = \varpi$. Claramente $g(\varpi) = g(\varpi_1)$.

4. r' é regra estrutural. A construção é similar ao item anterior.
5. r é alguma regra lógica que não introduza φ à direita. Pela hipótese de indução, para π_i e π' temos demonstração(ões) ϖ_i de $\Gamma_i, \Gamma' - \varphi \mapsto \Delta_i - \varphi, \Delta'$ com grau estritamente menor que d . Se aplicarmos apropriadamente a mesma regra r à(s) ϖ_i (obtida(s) pela(s) hipótese(s) de indução) e possivelmente algumas regras estruturais, conforme dois casos principais:

$I = 1$

$$\frac{\frac{\Gamma_1 \mapsto \Delta_1}{\Gamma \mapsto \Delta} : r \quad \Gamma' \mapsto \Delta'}{\quad} : r$$

$$\frac{\Gamma_1, \Gamma' - \varphi \mapsto \Delta_1 - \varphi, \Delta'}{\Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'} : r$$

Com $r \in \{\wedge \mathbf{E}, \vee \mathbf{D}, \rightarrow \mathbf{D}, \neg \mathbf{E}, \neg \mathbf{D}, \forall \mathbf{E}, \forall \mathbf{D}, \exists \mathbf{E}, \exists \mathbf{D}\}$.

$I = 2$

$$\frac{\frac{\Gamma_1 \mapsto \Delta_1 \quad \Gamma_2 \mapsto \Delta_2}{\Gamma \mapsto \Delta} : \wedge \mathbf{D}, \vee \mathbf{E}, \rightarrow \mathbf{E} \quad \Gamma' \mapsto \Delta'}{\quad} : \wedge \mathbf{D}, \vee \mathbf{E}, \rightarrow \mathbf{E}$$

$$\frac{\Gamma_1, \Gamma' - \varphi \mapsto \Delta_1 - \varphi, \Delta' \quad \Gamma_2, \Gamma' - \varphi \mapsto \Delta_2 - \varphi, \Delta'}{\Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'} : \wedge \mathbf{D}, \vee \mathbf{E}, \rightarrow \mathbf{E}$$

obteremos ϖ sem maiores problemas, pois todas as regras estão previstas na demonstração do caso clássico com exceção de $\neg \mathbf{E}$:

$$\frac{\frac{\Gamma'_1, \circ\alpha \mapsto \Delta, \alpha}{\Gamma'_1, \circ\alpha, \neg\alpha \mapsto \Delta} : \neg \mathbf{E} \quad \Gamma' \mapsto \Delta'}{\quad} : \neg \mathbf{E}$$

$$\frac{\Gamma'_1, \circ\alpha, \Gamma' - \varphi \mapsto (\Delta, \alpha) - \varphi, \Delta'}{\Gamma'_1, \circ\alpha, \neg\alpha, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'} : \neg \mathbf{E}$$

Para essa regra, pode-se facilmente ver que a fórmula de restrição $\circ\alpha \in \Gamma_1$ ainda se mantém em $\Gamma_1, \Gamma' - \varphi$ (que, nesse caso, é idêntico a $\Gamma'_1, \circ\alpha, \neg\alpha, \Gamma' - \varphi$), permitindo aplicar a regra novamente ao sequente final de ϖ_1 caso a fórmula principal de $\neg \mathbf{E}$ não seja $\neg\varphi$ (isto é, não seja o caso de que $\alpha \equiv \varphi$). Caso seja, basta **Enf-E** para introduzir $\neg\varphi$ à esquerda ($\neg\varphi \in \Gamma$).

6. r' é alguma regra lógica que não introduz φ à esquerda. Pela hipótese de indução, para π e π'_j temos demonstração(ões) ϖ'_j de $\Gamma, \Gamma'_j - \varphi \mapsto \Delta - \varphi, \Delta'_j$. O único caso que nos impede de proceder como no item anterior é quando r' é $\neg E$, com fórmula principal $\neg\phi$ e $\varphi \equiv \circ\phi$:

$$\frac{\pi' \quad \vdots \quad \frac{\Gamma''_1, \circ\phi \mapsto \Delta', \phi}{\Gamma''_1, \circ\phi, \neg\phi \mapsto \Delta'} : r' = \neg E}{\Gamma, \Gamma''_1 - \circ\phi \mapsto \Delta - \circ\phi, \Delta', \phi} \quad \text{HI} \quad (\star)$$

pois teríamos que demonstrar:

$$\frac{\Gamma, \Gamma''_1 - \circ\phi \mapsto \Delta - \circ\phi, \Delta', \phi}{\Gamma, \Gamma''_1 - \circ\phi, \neg\phi \mapsto \Delta - \circ\phi, \Delta'} \equiv \frac{\text{HI}}{\text{QED}} \quad (\star)$$

usando apenas $\neg E$ e regras estruturais. Pode ser que não haja uma $\circ\phi$ em Γ que sirva de fórmula de restrição para a aplicação de $\neg E$.

Logo, para chegar em uma demonstração ϖ que termine em (\star) , o caso em que $\circ\phi \notin \Delta$ é trivial, pois *QED* pode ser obtido a partir de π (que termina em $\Gamma \mapsto \Delta$) através de repetidos enfraquecimentos.

Agora, suponha que $\circ\phi \in \Delta$. Tendo em vista que em **MBC** não há regras para a introdução de \circ , os ancestrais de $\circ\phi$ estão todos ao lado direito dos sequentes de π e a eles se aplicaram apenas regras estruturais. Temos dois possíveis casos:

- (a) Se entre tais ancestrais não encontramos nenhuma fórmula introduzida por axioma, elimine-se de π todas as regras estruturais, obtendo então uma demonstração de $\Gamma \mapsto \Delta - \circ\phi$. Para chegar à ϖ bastam regras estruturais e temos que $g(\varpi) = g(\pi)$.
- (b) Se encontrarem-se ancestrais das $\circ\phi$ que ocorrem em Δ entre axiomas de π , usando a hipótese de indução encontramos uma demonstração ϖ_0 de $\circ\phi, \Gamma''_1 - \circ\phi, \neg\phi \mapsto \Delta'$ a partir do axioma e de π' . Substituindo-se tais demonstrações pelos axiomas $\circ\phi \mapsto \circ\phi$ em π e apagando todos herdeiros da $\circ\phi$ à direita, bem como os enfraquecimentos ancestrais a qualquer $\circ\phi \in \Delta$ que não tenha se originado em axioma, obtemos a demonstração ϖ desejada, com $g(\varpi) = g(\pi)$.

7. Ambas r e r' são regras lógicas, r introduz φ à direita e r' introduz φ à esquerda. Pela hipótese de indução aplicada respectivamente a:

(a) π_i e π'_i , obtemos a(s) demonstraçã(o)es ϖ_i de $\Gamma_i, \Gamma' - \varphi \mapsto \Delta_i - \varphi, \Delta'$.

(b) π e π' , obtemos a(s) demonstraçã(o)es ϖ'_j de $\Gamma, \Gamma'_j - \varphi \mapsto \Delta - \varphi, \Delta'_j$.

Note-se que as demonstrações recém obtidas pela hipótese de indução têm grau menor que o grau d de φ . Aplicando, então, a regra r (e algumas regras estruturais) a ϖ_i obtem-se uma derivação ρ de:

$$\Gamma, \Gamma' - \varphi \mapsto \varphi, \Delta - \varphi, \Delta'.$$

Igualmente pode-se aplicar r' (e regras estruturais) a ϖ'_j e obter uma derivação ρ' de:

$$\Gamma, \Gamma' - \varphi, \varphi \mapsto \Delta - \varphi, \Delta'.$$

Agora, faz-se o **Corte** da φ que está à direita na conclusão de ρ com a φ que está à esquerda na conclusão de ρ' e obtem-se:

$$\Gamma, \Gamma' - \varphi, \Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta', \Delta - \varphi, \Delta'.$$

No entanto, o corte feito tem grau d e foi efetuado sobre demonstrações de grau estritamente menor que d . Essas são exatamente as hipóteses do Lema 4.17, que garante uma demonstração desse último sequente com cortes de grau estritamente menor que d .

Por repetidos enfraquecimentos, pode-se obter uma demonstração ϖ de grau estritamente menor que d do seguinte sequente:

$$\Gamma, \Gamma' - \varphi \mapsto \Delta - \varphi, \Delta'.$$

□

A Eliminação do Corte

Lema 4.20. *Se π é uma demonstração com grau $d > 0$ para um sequente, então é possível construir uma nova prova ϖ para o mesmo sequente com grau estritamente menor que d .*

Demonstração. Por indução em $h(\pi)$. Seja r a última regra de π e π_i a(s) subprova(s) correspondente(s) à(s) premissa(s) de r . Temos dois casos:

1. r não é um corte de grau d . Pela hipótese de indução, temos ϖ_i de grau $< d$. Obtemos ϖ aplicando r à(s) ϖ_i .
2. r é um corte de grau d :

$$\frac{\Gamma \mapsto \varphi, \Delta \quad \Gamma', \varphi \mapsto \Delta'}{\Gamma, \Gamma' \mapsto \Delta, \Delta'} : r$$

A hipótese de indução nos dá ϖ_i de grau estritamente menor que d , para as premissas de r . Aplicando o Lema Principal (4.19), obtemos uma demonstração ϖ de $\Gamma, \Gamma' \mapsto \Delta, \Delta'$ com grau $< d$.

□

Teorema 4.21 (Eliminação do Corte). *A regra de corte é redundante em MBC.*

Demonstração. Pela iteração do Lema 4.20.

□

4.2 Teorema de Herbrand para QmbC

A demonstração apresentada aqui é de uma versão restrita do teorema, no sentido de que estamos tratando apenas fórmulas Π_2 , consequências lógicas de conjuntos de fórmulas não quantificadas. Em sua tese de doutorado [Her30], Herbrand apresentou a forma geral, aplicável a quaisquer sentenças logicamente válidas. Note-se que, no entanto, Herbrand tratava como primitivos apenas “ \neg ”, “ \vee ” e “ \exists ”, e também “ \forall ” para fórmulas prenexas (cf. [WSBA09, p.20]), o que não pode ser feito no contexto das LFI’s. Nessas lógicas não é sempre possível definir todos os conectivos em termos de conjuntos de conectivos que para as lógicas clássicas são suficientes (cf. [CCM07]).

A presente demonstração foi baseada em [Bus98], em que está provado também o caso mais geral, dadas as devidas correções de [Mck]. Em tais artigos também, quando tratam da versão geral, assumem sem perda de generalidade fórmulas restritas à forma normal negativa (em que a negação aplica-se apenas às fórmulas atômicas) e ao fragmento gerado a partir de “ \neg ”, “ \vee ”, “ \wedge ” e os quantificadores. Tal caminho não é possível para as LFI’s, por causa dos conectivos não vero-funcionais \neg e \circ .

Lema 4.22. *Para toda demonstração por sequentes, mesmo que admitindo um conjunto arbitrário de hipóteses como sequentes iniciais, toda fórmula de todo sequente da demonstração só não aparece no sequente final (como fórmula ou subfórmula de alguma fórmula presente em tal sequente), se for eliminada por um corte.*

Demonstração. Basta reparar que, nos formalismos de sequentes, não temos regras para eliminação de fórmula alguma que não a introduza como subfórmula de uma mais complexa – excetuando-se, é claro, o **Corte**. \square

Teorema 4.23 (Teorema de Herbrand Restrito). *Seja T um conjunto de sequentes do tipo $\vdash \varphi$, em que φ são fórmulas sem quantificadores, possivelmente com variáveis livres. Se for possível uma derivação em **MBC** de um sequente $\vdash \exists \vec{y} \psi(\vec{x}, \vec{y})$, que admita os sequentes de T como hipóteses:*

$$T \vdash_{\text{MBC}} \vdash \exists \vec{y} \psi(\vec{x}, \vec{y}),$$

então existe uma sequência finita de termos $t_{i,j} = t_{i,j}(\vec{x})$, com $1 \leq i \leq r$ e $1 \leq j \leq k$, para a qual é possível também a seguinte dedução:

$$T \vdash_{\text{MBC}} \vdash \psi(\vec{x}, t_{1,1}, \dots, t_{1,k}), \dots, \psi(\vec{x}, t_{r,1}, \dots, t_{r,k})$$

Demonstração. O teorema da eliminação do corte não considera o caso em que se admitam premissas, mas é possível ver que, quando as admitimos, os únicos cortes que não poderão ser eliminados são aqueles que envolvem algum sequente obtido a partir das premissas por uma substituição⁵ de variáveis por termos que abranja todas as suas fórmulas. Podemos, então, encontrar uma demonstração ϖ da hipótese:

$$\frac{\begin{array}{c} \varpi \\ \vdots \\ \pi \\ \vdots \\ \Gamma \vdash \Delta \\ \vdash \exists \vec{y} \psi(\vec{x}, \vec{y}) \end{array}}{\vdash \exists \vec{y} \psi(\vec{x}, \vec{y})} : \text{R}$$

que faça uso de cortes apenas com sequentes obtidos a partir de substituições dos sequentes de T . Logo, todos os cortes são sobre fórmulas sem quantificadores.

Observe-se que o sequente final de ϖ tem uma única fórmula em que todos seus quantificadores precedem uma matriz livre de quantificações:

$$(\exists y_1) \dots (\exists y_k) \psi(x_1, \dots, x_m, y_1, \dots, y_k)$$

e a única maneira de alguma fórmula de algum sequente de ϖ não aparecer como fórmula, ou subfórmula de alguma fórmula do sequente final é através de cortes (Lema 4.22). Logo, todas as fórmulas quantificadas em ϖ são ancestrais da conclusão de ϖ .

⁵Veja [GTL89, p.111].

Temos, então, que todas as fórmulas quantificadas têm o seguinte formato (com $1 \leq j \leq k$):

$$(\exists y_j) \dots (\exists y_k) \psi(\vec{x}, t_1, \dots, t_{j-1}, y_j, \dots, y_k),$$

estão à direita dos sequentes e as possíveis regras em que estavam envolvidas são $\exists D$ ou **Cont-D**.

Pode-ser ver, então, que todos os sequentes da demonstração de ϖ têm o seguinte formato:

$$\Gamma \mapsto \Delta, \Delta',$$

de maneira que, em Γ e Δ , há apenas fórmulas sem quantificadores e, em Δ' , apenas existenciais.

Por indução no comprimento da demonstração ϖ , demonstraremos que, para cada subprova de ϖ para um sequente $\Gamma \mapsto \Delta, \Delta'$, existe um $r \geq 0$ e uma demonstração:

$$T \vdash_{\text{MBC}} \Gamma \mapsto \Delta, \psi(\vec{x}, t_{1,1}, \dots, t_{1,k}), \dots, \psi(\vec{x}, t_{r,1}, \dots, t_{r,k}).$$

Considere-se os dois possíveis tipos de subprovas de ϖ :

$$\frac{\begin{array}{c} \pi \\ \vdots \\ \pi_1 \quad \pi_2 \\ \vdots \end{array} \quad \frac{\Gamma_1 \mapsto \Delta_1, \Delta'_1 \quad \Gamma_2 \mapsto \Delta_2, \Delta'_2}{\Gamma \mapsto \Delta, \Delta'} : \mathbf{R}}{\Gamma \mapsto \Delta, \Delta'} : \mathbf{R} \quad \frac{\begin{array}{c} \pi' \\ \vdots \\ \pi'' \\ \vdots \end{array} \quad \frac{\Gamma' \mapsto \Delta', \Delta''}{\Gamma \mapsto \Delta, \Delta'} : \mathbf{R}'}{\Gamma \mapsto \Delta, \Delta'} : \mathbf{R}'$$

Note-se que \mathbf{R} não influi com a parte existencial e devemos ter $\Delta' = \Delta'_1 \cup \Delta'_2$, além de que as seguintes são as únicas regras possíveis (sendo \mathbf{R}^P regras proposicionais e, portanto, também não atuam, em ϖ , sobre as fórmulas existenciais):

$$\mathbf{R} \in \{ \rightarrow E, \wedge D, \vee E, \text{Corte} \}$$

$$\mathbf{R}' \in \{ \text{Cont-D}, \exists D \} \cup \mathbf{R}^P$$

Para os casos em que se trata de uma regra atuando sobre fórmulas sem quantificadores ou **Cont-D** sobre alguma fórmula existencial, basta aplicar as mesmas regras às hipóteses de indução e obter o resultado.

Se \mathbf{R} for um **Corte**, com alguma substituição dos sequentes de T :

$$\begin{array}{c}
 \pi \\
 \vdots \\
 \pi_1 \qquad \qquad \qquad \pi_2 \\
 \vdots \qquad \qquad \qquad \vdots \\
 \frac{\vdash \varphi_{\vec{x}}[\vec{t}] \quad \Gamma'_2, \varphi_{\vec{x}}[\vec{t}] \vdash \Delta_2, \Delta'_2}{\Gamma'_2 \vdash \Delta_2, \Delta'_2} : \mathbf{Corte}
 \end{array}$$

podemos obter o resultado pelo **Corte** da substituição com a hipótese de indução aplicada a subprova π_2 . Também não precisamos considerar o caso de axiomas com fórmulas quantificacionais devido ao Corolário 4.22 e ao Teorema 4.10.

Por fim, considere $\mathbf{R}' = \exists \mathbf{D}$:

$$\begin{array}{c}
 \pi' \\
 \vdots \\
 \pi'' \\
 \vdots \\
 \frac{\Gamma \vdash \Delta, \Delta'', (\exists y_j) \dots (\exists y_k) \psi(\vec{x}, t_1, \dots, t_{j-1}, y_j, \dots, y_k)}{\Gamma \vdash \Delta, \Delta'', (\exists y_{j-1}) \dots (\exists y_k) \psi(\vec{x}, t_1, \dots, t_{j-2}, y_{j-1}, \dots, y_k)} : \exists \mathbf{D}
 \end{array}$$

A hipótese de indução também satisfaz a condição para o seqüente final de π' . \square

Com o teorema de Herbrand restrito, terminamos as primeiras duas fases propostas para a fundamentação da programação lógica paraconsistente. Já está claro que é possível encarar tal assunto por uma abordagem baseada nas **LFI**'s. Na seqüência, iremos delinear os caminhos que nos parecem mais naturais para dar continuidade à proposta de fundamentar um amplo ambiente de programação lógica paraconsistente.

Capítulo 5

Próximos Passos

Neste último capítulo serão indicados os próximos passos necessários para dar sequência ao programa delineado ao longo da dissertação, qual seja, o de estabelecer um amplo formalismo para programação lógica paraconsistente com base nas **LFI**'s. Conforme foi discutido na Seção 3.1, a programação lógica paraconsistente está baseada nos chamados *programas lógicos estendidos*, que admitem dois tipos de negação: uma negação *explícita* (\neg), monotônica e paraconsistente, e uma não-monotônica e explosiva ($\overset{\circ}{\neg}$). A negação não-monotônica em programação lógica surgiu no contexto dos chamados programas lógicos gerais, que fazem uso da regra da negação por falha para deduzir literais negados (com $\overset{\circ}{\neg}$). Para uma brevíssima descrição, veja-se a Seção 2.3. Hoje em dia, os programas lógicos estendidos admitem várias caracterizações semânticas dentro da perspectiva da programação lógica paraconsistente. Um bom apanhado dessas abordagens semânticas encontra-se em [DP98].

A literatura atual em programação lógica paraconsistente está focada nas semânticas não-monotônicas para os programas estendidos, tendo em vista que é este o caso geral. Mas é de se notar a falta de uma base lógica bem definida, com uma formulação axiomática explícita, que sirva mesmo ao seu fragmento monotônico: os programas lógicos estendidos definidos. Como visto na Seção 3.2, as caracterizações atuais deste fragmento apelam a noções extra-lógicas, como a suposição de que a tradução da Definição 3.3 transfira o resultado clássico do teorema de Herbrand à lógica inerente aos programas estendidos definidos, necessariamente paraconsistente. Acreditamos que as **LFI**'s podem cumprir o papel de uma legítima lógica de base para a programação lógica estendida definida, que explicita suas características paraconsistentes inerentes. Com a explicitação de uma lógica básica, poderia-se considerar extensões da programação lógica paraconsistente que admitissem fórmulas mais complexas nas regras como, por exemplo, fórmulas atô-

micas precedidas pelo conectivo de consistência. Partiria-se, também, com a posse de uma lógica base clara, de uma perspectiva privilegiada para abordar o caso geral dos programas lógicos estendidos.

Antes de finalizar com algumas conjecturas e indicações de futuros rumos do presente programa de pesquisa, serão ressaltadas, na próxima seção, algumas dificuldades inerentes a qualquer abordagem que vise métodos automáticos de demonstração para lógicas paraconsistentes. Na sequência, haverá ainda uma seção lidando brevemente com uma possibilidade de resolução restrita a fórmulas na forma normal conjuntiva, para as lógicas **QmbC** e **QmCi**.

5.1 Problemas com a Dualidade entre Satisfatibilidade e Validade: Considerações Semânticas

Está claro que, pela própria definição de consequência e validade lógica, a validade universal é um subproblema do problema mais geral da consequência lógica. No contexto clássico, graças à lei da dupla negação, tem-se que a validade está reduzida à insatisfatibilidade e, graças também à completude e à compacidade, a consequência lógica pode ser estabelecida por métodos sintáticos de refutação:

$$\begin{aligned} \varphi \text{ é válida} & \iff \emptyset \vDash \varphi \\ \varphi \text{ é válida} & \iff \neg\varphi \text{ é insatisfável} \\ \Gamma \vDash \varphi & \iff \Gamma \vdash \neg(\neg\varphi) \end{aligned}$$

Conforme visto em 1.1, para determinar a consequência lógica em primeira ordem clássica, a insatisfatibilidade é suficiente e necessária. Para essa, basta e são suficientes procedimentos sintáticos de refutação:

$$\Gamma \vDash \varphi \iff \Gamma, \neg\varphi \text{ é insatisfável} .$$

As coisas mudam de figura quando se está considerando semânticas paraconsistentes. Em geral, é possível:

$$\Gamma \vDash_{\mathbf{P}} \neg\varphi \text{ e } \Gamma, \varphi \text{ satisfável} ,$$

ou mesmo:

$$\Gamma \vDash_{\mathbf{P}} \varphi \text{ e } \Gamma, \neg\varphi \text{ satisfável} .$$

Logo, em geral, tem-se apenas:

$$\begin{aligned} \Gamma \vDash_{\mathbf{P}} \varphi & \iff \Gamma, \neg\varphi \text{ insatisfável} \\ \Gamma \vDash_{\mathbf{P}} \neg\varphi & \iff \Gamma, \varphi \text{ insatisfável} \end{aligned}$$

Mas, pela eliminação do corte, conjecturamos que é possível demonstrar:

$$\vdash_{\mathbf{QmbC}} \neg\varphi \iff \varphi \text{ instatisfável.}$$

Apesar de ser possível:

$$\models_{\mathbf{P}} \varphi \text{ com } \neg\varphi \text{ satisfável.}$$

Como demonstrado no Teorema 3.55, a busca pela base de Herbrand não é suficiente para determinar consequência lógica em geral, dado que dois modelos paraconsistentes quaisquer podem coincidir na base de Herbrand de suas linguagens diagrama, mas divergir para fórmulas complexas.

5.2 Um Método de Resolução Restrita

Como visto na Seção 1.2, a resolução faz uso essencial da forma normal conjuntiva. Nas lógicas paraconsistentes, nem sempre é possível estabelecer formas normais para as fórmulas. Outro problema é que o silogismo disjuntivo $[\alpha, \neg\alpha \vee \beta \vdash \beta]$ não pode valer em nenhuma extensão paraconsistente da lógica positiva clássica ou intuicionista, conforme o Teorema 3.19 de [CM02, p.40].

Mas, partindo de fórmulas que já estejam na forma normal conjuntiva, é possível uma espécie de regra de resolução para algumas **LFI**'s.

Teorema 5.1 (Resolução Restrita). *Pode-se definir, para \mathbf{QmbC} e \mathbf{QmCi} , uma regra de resolução restrita, que não elimine totalmente a fórmula resolvente:*

$$(\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma) \wedge \Phi \vdash (\neg \circ \alpha \vee \beta \vee \gamma) \wedge \Phi$$

Demonstração. Pode-se verificar checando todas as valorações paraconsistentes possíveis para as duas lógicas (veja-se a Definição 3.46). \square

5.3 Conclusões

Por meio do teorema de Herbrand demonstrado no capítulo anterior e das características lógicas fundamentais dos programas estendidos definidos (sua monotonicidade e paraconsistência), acreditamos ser possível demonstrar uma conjectura que relaciona as **LFI**'s com várias abordagens à programação lógica paraconsistente:

Conjectura 5.2. *A semântica monotônica definida por Damásio e Pereira em [DP98] (Definição 3.3) para programas estendidos definidos, que fazem uso somente da negação explícita, é correta e completa para a consequência lógica em*

QmbC e **QmCi**, quando restritas ao fragmento em questão. Ou seja, para um programa estendido definido E , um literal L e $\mathfrak{L} \in \{\mathbf{QmbC}, \mathbf{QmCi}\}$:

$$L \in M_E \iff E \vdash_{\mathfrak{L}} L.$$

Parece, então, que seria possível caracterizar a semântica M_E através de *modelos paraconsistentes minimais*, que seguiriam a definição clássica, dada através dos conceitos de base de Herbrand, modelo de Herbrand e operador de consequência imediata (Definições 2.27, 2.35 e 2.30) adaptados às **LFI**'s. Uma diferença, por exemplo, estaria em definir a base de Herbrand paraconsistente de E (BP^E) como o conjunto de todos os literais fechados de sua linguagem. Deveria-se exigir, também, que nos modelos de Herbrand paraconsistentes estivesse sempre presente uma sentença atômica ou sua negação.

Tracemos um paralelo como o caso clássico. No contexto clássico, de um programa lógico P , não é possível deduzir nenhuma literal negada, pois a base de Herbrand de P é um modelo de P contra-modelo para toda negação (veja-se a Seção 2.3):

$$B_P \models P$$

$$B_P \not\models \neg A.$$

Para um programa lógico estendido definido E , esta análise não é possível. Mesmo que sejam interpretadas todas as fórmulas atômicas da base de Herbrand (clássica) da linguagem de E como verdadeiras, pode ser que isto não seja suficiente para verificar todas as suas regras. Isto ocorre porque são permitidas literais negadas na cabeças das regras de programas lógicos estendidos. Dessa maneira, pode haver literais negativos que sejam consequências lógicas de E . Ou seja, se fosse possível a caracterização que está sendo proposta acima, isto seria expresso do seguinte modo:

$$BP_E \models E, \text{ mas:}$$

$$BP_E \models \neg A.$$

A situação análoga é encontrada com o conectivo de consistência. Para E , existe uma estrutura para **QmbC** (ou **QmCi**) \mathfrak{C} que é modelo do programa, mas falsifica todas as fórmulas de consistência. Basta tomar como domínio de interpretação todos os termos fechados, interpretar as funções da maneira feita na Definição 2.29 e atribuir os predicados sempre como válidos, para todos os termos da linguagem. Então, uma bivaloração paraconsistente que atribua o valor de *verdadeiro* a todos literais da linguagem de E (*incluindo os negativos*) satisfaz:

$$\mathfrak{C} \models E$$

$$\mathfrak{C} \neq \circ A .$$

Esta estrutura \mathfrak{C} estaria determinada exatamente por BP^E . Este tipo de análise dá indícios do que seria uma programação lógica paraconsistente monotônica que admitisse fórmulas de consistência, mesmo entre as cabeças de suas regras.

Com esses resultados futuros, estaria refeita em bases *legitimamente paraconsistentes* a teoria dos programas lógicos estendidos definidos. Logo, tais lógicas poderiam ser consideradas *as lógicas* de base da parte monotônica de uma vasta gama de abordagens à programação lógica paraconsistente. Com tal ganho conceitual, haveria melhores condições para se poder analisar com detalhe as semânticas não-monotônicas da negação paraconsistente.

O teorema de Herbrand apresentado nesta dissertação também indica que é possível desenvolver a teoria da resolução de forma a abranger as **LFI**'s como um todo. Desta maneira, estariam lançadas as bases para um ambiente de programação lógica que abrangesse uma ampla variedade de maneiras de se lidar logicamente com inconsistências. Claro que isso estaria condicionado a estender os resultados da completude, da eliminação do corte e o teorema de Herbrand para as demais **LFI**'s.

Paralelamente, seria muito desejável que se realizasse uma análise conceitual sobre o papel da paraconsistência em bases de conhecimento e nas diferentes possibilidades de negações da programação lógica. Tais questões ficam, contudo, condicionadas à pesquisa futura.

Referências Bibliográficas

- [ADP95] José Júlio Alferes, Carlos Viegas Damásio e Luís Moniz Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14:93–147, 1995.
- [AP96] José Júlio Alferes e Luís Moniz Pereira. Reasoning with logic programming. LNAI 1111, 1996.
- [Apt90] Krzysztof R. Apt. Logic programming. *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, pp. 493–574, 1990.
- [AZ06] Arnon Avron e Anna Zamansky. Many-valued non-deterministic semantics for first-order logics of formal (in)consistency. In *Algebraic and Proof-theoretic Aspects of Non-classical Logics*, pp. 1–24, 2006.
- [BF85] J. Barwise e S. Feferman, editores. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer-Verlag, New York, 1985.
- [Boo47] George Boole. *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay and Macmillan, Cambridge, 1847.
- [BS89] H. A. Blair e V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68(2):135–154, 1989.
- [Bur98] S. Burris. *Logic for Mathematics and Computer Science*. Prentice Hall, 1998.
- [Bus98] Samuel R. Buss. An introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, pp. 1–78, Amsterdam, 1998. Elsevier.

- [CCM07] Walter Carnielli, Marcelo E. Coniglio e João Marcos. Logics of Formal Inconsistency. In D. Gabbay e F. Guentner, editores, *Handbook of Philosophical Logic (2nd. edition)*, volume 14, pp. 1–93. Springer, 2007.
- [Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):pp. 345–363, 1936.
- [Cla78] Keith L. Clark. *Negations as Failure*, capítulo Negative Information and Data Bases, pp. 293–322. In Gallaire e Minker [GM78], 1978.
- [CM02] Walter Carnielli e João Marcos. A taxonomy of C-systems. In Walter A. Carnielli, Marcelo E. Coniglio e Itala M. Loffredo D’Ottaviano, editores, *Paraconsistency - the Logical Way to the Inconsistent*, volume 228 of *Lecture Notes in Pure and Applied Mathematics*, pp. 01–94, New York, 2002. Marcel Dekker.
- [Cod69] E. F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *IBM Research Report*, 1969.
- [Col93] Alain Colmerauer. The birth of PROLOG. In *III, CACM Vol.33, No7*, pp. 37–52, 1993.
- [dAP07] Sandra de Amo e Mônica Sakuray Pais. A paraconsistent logic programming approach for querying inconsistent databases. *Int. J. Approx. Reasoning*, 46(2):366–386, 2007.
- [Dav83] Martin Davis. *The Prehistory and Early History of Automated Deduction*, pp. 1–28. Volume 1 of Siekmann e Wrightson [SW83], 1983.
- [dC63] Newton Affonso Carneiro da Costa. *Sistemas Formais Inconsistentes*. Tese de Livre Docência, Universidade Federal do Paraná, 1963.
- [dCBB98] N. C. A da Costa, Jean-Yves Béziau e Otávio Bueno. *Elementos de Teoria Paraconsistente de Conjuntos*, volume 23 of *Coleção CLE*. CLE-UNICAMP, Campinas, 1998.
- [DM47] Augustus De Morgan. *Formal Logic, or, The Calculus of Inference, Necessary and Probable*. Taylor and Walton, Booksellers and Publishers to University College, Upper Gower Street, 28, London, 1847.
- [Doe94] Kees Doets. *From Logic to Logic Programming*. MIT Press, Cambridge, MA, USA, 1994.

- [DP60] Martin Davis e Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
- [DP98] Carlos Viegas Damásio e Luís Moniz Pereira. A survey of paraconsistent semantics for logic programs. *Handbook of Defeasible Reasoning and Uncertainty Management Systems: Volume 2: Reasoning with Actual and Potential Contradictions*, pp. 241–320, 1998.
- [DvH86] Burton Dreben e Jean van Heijenoort. Nota introdutória a [Gö30], 1986. Em [Fef86].
- [Fef86] Solomon Feferman, editor. *Kurt Gödel Collected Works - Volume I - Publications 1929-1936*. Oxford University Press, New York, 1986.
- [Gal85] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row Publishers, Inc., New York, NY, USA, 1985.
- [Gen35] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.
- [Gen07] Paolo Gentilini. Paraconsistent arithmetic with a local consistency operator and global selfreference. *Electron. Notes Theor. Comput. Sci.*, 169:73–86, 2007.
- [Gen10] Paolo Gentilini. Proof-theory and mathematical meaning of paraconsistent c-systems. *Journal of Applied Logic*, 2010. Inédito, aguardando publicação.
- [Gir87] Jean-Yves Girard. *Proof Theory and Logical Complexity*, volume 1. Bibliopolis, 1987.
- [GM78] Hervé Gallaire e Jack Minker, editores. *Logic and Databases*. Plenum Press, New York, N.Y., 1978.
- [GMN78] Hervé Gallaire, Jack Minker e Jean Marie Nicolas. *An Overview and Introduction to Logic and Data Bases*, capítulo Introduction, pp. 3–30. In Gallaire e Minker [GM78], 1978.
- [GTL89] Jean-Yves Girard, Paul Taylor e Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.
- [Gö30] Kurt Gödel. Die Vollständigkeit des Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930. Tradução para o inglês em [Fef86].

- [HA28] David Hilbert e Wilhelm Ackermann. *Grundzüge der Theoretischen Logik*. Julius Springer, 1928.
- [HA50] David Hilbert e Wilhelm Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, 1950. Tradução (publicada em 1938) da segunda edição de [HA28].
- [Hen49] Leon Henkin. The completeness of the first-order functional calculus. *The Journal of Symbolic Logic*, 14(3):pp. 159–166, 1949.
- [Her30] Jackes Herbrand. *Recherches sur la Théorie de la Démonstration*. PhD thesis, Université de Paris, 1930.
- [Hin96] Jaakko Hintikka. *The Principles of Mathematics Revisited*. Cambridge University Press, 1996.
- [Jas48] Stanisław Jaśkowski. Rachunek zdań dla systemów dedukcyjnych sprzecznych. *Studia Societatis Scientiarum Torunensis, Sectio A*, I(5):57–77, 1948. Traduzido para o inglês em [Jas99].
- [Jas99] Stanisław Jaśkowski. A propositional calculus for inconsistent deductive systems. *Logic and Logic Philosophy*, 7:35–56, 1999.
- [Kow74] Robert A. Kowalski. Predicate logic as programming language. In *Proceedings IFIP'74*, pp. 569–574. North-Holland, 1974.
- [Kow79] Robert Kowalski. Algorithm = logic + control. *Commun. ACM*, 22(7):424–436, 1979.
- [Kow88] Robert A. Kowalski. The early years of logic programming. *Commun. ACM*, 31(1):38–43, 1988.
- [Llo87] John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, second edition, 1987.
- [LS91] J. W. Lloyd e J. C. Shepherdson. Partial evaluation in logic programming. *J. Log. Program.*, 11:217–242, August 1991.
- [Lö15] Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915. Traduzido como “On Possibilities in the Calculus of Relatives” em [vH02].
- [Mar05] João Marcos. *Logics of Formal Inconsistency*. Tese de Doutorado, Universidade Estadual de Campinas e Universidade Técnica de Lisboa, 2005.

- [Mck] Richard Mckinley. A sequent calculus demonstration of herbrand's theorem.
- [Men87] Elliot Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, New York, NY, USA, third edition, 1987.
- [Moo85] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75 – 94, 1985.
- [Moo87] R. Moore. *Possible-world semantics for autoepistemic logic*, pp. 137–142. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.
- [Nel59] David Nelson. Negation and separation of concepts in constructive systems. In Arend Heyting, editor, *Constructivity in Mathematics*, Studies in Logic and the Foundations of Mathematics, pp. 208–225, Amsterdam, 1959. North-Holland. Proceeding of the Colloquium held in Amsterdam, NL, 1957.
- [NG78] J. M. Nicolas e H. Gallaire. *Data Base: Theory vs. Interpretation (Nicolas, Gallaire - 1978)*, capítulo Data Bases Viewed Through Formal Logic, pp. 33–54. In Gallaire e Minker [GM78], 1978.
- [Pea93] David Pearce. Answer sets and constructive logic, II: extended logic programs and related nonmonotonic formalisms. In *Proceedings of the Second International Workshop on Logic Programming and Non-monotonic Reasoning*, pp. 457–475, Cambridge, MA, USA, 1993. MIT Press.
- [Pod08] Rodrigo Podiacki. Lógicas da inconsistência formal quantificadas. Dissertação de Mestrado, Universidade Estadual de Campinas, 2008.
- [PR91] Stephen G. Pimentel e William L. Rodi. Belief revision and para-consistency in a logic programming framework. In Wiktor Marek, Anil Nerode, and V. S. Subrahmanian, editores, *Proceedings of the 1st International Workshop on Logic Programming and Non-monotonic Reasoning*, pp. 228–242, Washington, D.C., July 1991. MIT Press.
- [Rei78] Raymond Reiter. *On Closed World Data Bases*, capítulo Data Bases Viewed Through Formal Logic, pp. 55–76. In Gallaire e Minker [GM78], 1978.
- [Rei80] Raymond Reiter. A logic for default theory. *Artificial Intelligence*, 13:81–132, 1980.

- [Rob57] A. Robinson. Proving a theorem (as done by man, logician, or machine). In *Summaries of Talks Presented at the Summer Institute for Symbolic Logic*, 1957. Segunda edição publicada pelo *Institute for Defense Analysis*, 1960. Reeditada em [SW83], pp. 74–76.
- [Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
- [Sak92] Chiaki Sakama. Extended well-founded semantics for paraconsistent logic programs. In *In Fifth Generation Computer Systems*, pp. 592–599, 1992.
- [Sco70] Dana S. Scott. Outline of a mathematical theory of computation. Technical Monograph PRG–2, Oxford University Computing Laboratory, Oxford, England, November 1970.
- [Sho67] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, MA, 1967.
- [SI95] Chiaki Sakama e Katsumi Inoue. Paraconsistent stable semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5:265–285, 1995.
- [Sko20] Thoralf Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Satze nebst einem Theoreme über dichte Mengen. *Videnskapsselskapets Skrifter, I. Matematisk-naturvidenskabeligklasse, no.4.*, 1920. Traduzido como “*Logico-combinatorial investigations in the satisfiability or provability of mathematical propositions: A simplified proof of a theorem by L. Lowenheim and generalizations of the theorem*” em [vH02].
- [Sko28] Thoralf Skolem. Über die matematische Logik. *Norsk matematisk tidsskrift*, 10:125–142, 1928. Traduzido como “*On Mathematical Logic*” em [vH02].
- [Soa96] Robert I. Soare. Computability and recursion. *The Bulletin of Symbolic Logic*, 2(3):284–321, 1996.
- [SW83] J. Siekmann e G. Wrightson, editores. *Automation of Reasoning: Classical Papers on Computational Logic 1957-1966*, volume 1. Springer, Berlin, Heidelberg, 1983.

- [Tak75] Gaisi Takeuti. *Proof Theory*. Number 81 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1975. Second edition, 1987.
- [Tar44] Alfred Tarski. The semantic conception of truth: and the foundations of semantics. *Philosophy and Phenomenological Research*, 4:341–376, 1944.
- [Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.
- [TS96] A. S. Troelstra e H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, New York, NY, USA, 1996.
- [Tur37] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.
- [vEK76] M. H. van Emden e R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:569–574, 1976.
- [vH02] Jean van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Source Books in History of Sciences. Harvard University Press, January 2002.
- [Wag93] Gerd Wagner. Reasoning with inconsistency in extended deductive databases. In *Proceedings of the Second International Workshop on Logic Programming and Non-monotonic Reasoning*, pp. 300–315, Cambridge, MA, USA, 1993. MIT Press.
- [Wag94] Gerd Wagner. *Vivid Logic: Knowledge-Based Reasoning with Two Kinds of Negation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.
- [WSBA09] Claus-Peter Wirth, Jörg Siekmann, Christoph Benzmüller e Serge Autexier. Lectures on Jacques Herbrand as a logician. SEKI Report SR-2009-01 (ISSN 1437-4447), SEKI Publications, DFKI Bremen GmbH, Safe and Secure Cognitive Systems, Cartesium, Enrique Schmidt Str.5, D-28359 Bremen, Germany, 2009.