Programação em Lógica Estendida da Inconsistência Epistêmica

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por João Fernando Lima Alcântara e aprovada pela Banca Examinadora.

Fortaleza, 10 de Fevereiro de 2000.

Marcelino Pequeno (DC-UFC) (Orientador)

Dissertação apresentada ao Mestrado em Ciência da Computação, UFC, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Mestrado em Ciência da Computação
Universidade Federal do Ceará

Programação em Lógica Estendida da Inconsistência Epistêmica

João Fernando Lima Alcântara[1]

10 de Fevereiro de 2000

Banca Examinadora:

- Marcelino Pequeno (DC-UFC) (Orientador)
- Jean-Yves Béziau (DC-PUC)
- Tarcísio Pequeno (DC-UFC)

[1]**Bolsista da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior)**

Considerações Iniciais

"Os sonhos vêm;
Os sonhos vão;
O resto é imperfeito."
Renato Russo

Um tanto contrariando os cânones acadêmicos, não posso furtar-me de apresentar este espaço em que tento desnudar-me do rigor exigido a uma obra científica ao desvelar-me enquanto ser humano. No restante do trabalho, juro beatamente que tentei imprimir um vezo humano em contrapartida ao hermetismo do assunto abordado. Contudo, tenho uma leve percepção que fui infrutífero nessa subversão.

Num trabalho da envergadura de uma Dissertação de Mestrado, muitos dos elementos envolvidos transcendem a esfera do trabalho em si. O seu processo de criação carece de uma enxurrada amazônica de sangue, suor e lágrimas que o leitor mais desavisado talvez não perceba. Não obstante, subliminar a cada parágrafo, roçagando cada equação matemática, no recôndito enervante de cada entrelinha, há o esforço de uma vida inteira eriçada de sonhos e esperanças. Há também o admirável senso de solidariedade e companheirismo presentes em meus colegas que muito contribuíram para a minha formação. Ousaria dizer que durante a realização do Curso de Mestrado, as principais contribuições obtidas não estão encarceradas na frialdade inorgânica desta Dissertação, mas vivas nos corações e mentes das pessoas que compartilharam comigo esses momentos.

Em sua escrita, houve uma intensa simbiose entre autor e obra a ponto de em certos momentos, não se saber quem estava construindo quem. Nesse trabalho, inebriado a cada descoberta, sentia-me, por instantes, como o artesão limando as arestas da tosca matéria bruta. Tal delírio, no entanto, não chegava a ser reconfortante, pois ao contrário da Arte, a Ciência não concede aos seus súditos o doce sabor da perfeição. A mim, não é dado o direito de voltar-se para a minha obra e dizer "Parla". Não! Definitivamente, não posso me sentir como um artista: sou apenas o alferes da cavalaria, um reles apertador de parafusos de uma imensa geringonça, que nem sempre funciona direito, chamada Ciência. Mas como admitir as falhas do meu filho? Como conviver com uma obra já sabida ser imperfeita? Esse retrato de Dorian Gray que não envelhece em detrimento do retratado. Assim, tal qual Tântalo, sou penitenciado ao suplício de buscar o inalcançável.

Feitas essas ponderações, mas ainda a regojizar-me da prerrogativa que o pomposo título de Mestre confere, gostaria que o meu leitor visse nesse trabalho uma singela contribuição para a discussão sobre os caminhos da humanidade. Tem a ver com uma tal de alavanca que teimo em construir...

Agradecimentos

Entrevi, certa vez, que a seção de agradecimentos é uma das mais importantes de um trabalho científico, pois caso seja bem sucedido, poucos se importarão dessa seção; no entanto, caso seja mal sucedido, muitos irão verificar quem foram os infelizes que colaboraram nesse resultado. Para tranqüilizar a todos, devo informar que não compactuo com essa prática tão ardilosa. Também irei tentar não maçar o leitor com uma lista infindável. Assim, em acordo com as diretrizes do Ministério da Saúde, tentarei ser genérico:

Sou eternamente e ternamente agradecido aos meus pais: o seu Chico Manim e a Dona Expedita. Com seu amor e a poesia peculiar às boas pessoas que vivem no interior do Nordeste deste país, proporcionam-me grandes lições de vida;

Não posso deixar de estender esse agradecimento aos meus irmãos, sobrinhos, tios, primos e a toda a boa gente de Alcântaras: *E o que disserem, vocês sempre estiveram esperando por mim.*

Ao meu orientador, prof. Marcelino Pequeno, cuja experiência e rigor técnico tornaram esta obra viável. Seus comentários e sugestões creditaram a este trabalho muito dos méritos que vem recebendo.

A todos que lutam pelo Departamento de Computação da UFC. Em especial ao grupo de IA.

A todos os amigos de Mestrado cuja convivência foi precípua para a realização deste trabalho.

A todos os meus amigos: *Não preciso de modelos, não preciso de heróis; eu tenho os meus amigos*[1].

A todas as organizações populares que lutam contra as injustiças sociais que martirizam o nosso povo.

À cultura popular nordestina;

Às mulheres extraordinárias que povoaram (e povoam) o meu universo.

À tríade.

... e viva às mesas de bar!!!

[1] In Comédia Romântica (Legião Urbana).

Resumo

Na visão tradicional de programação em lógica, a única maneira de representar a informação negativa é através da negação por falha ou negação default. Em muitos casos, isso corresponde a uma séria limitação. Assim, nos últimos anos, vários autores vêm apontando as vantagens de estender programação em lógica com um segundo tipo de negação para explicitamente representar a informação negativa. Os programas resultantes são chamados de programas em lógica estendidos.

Com a introdução da negação explícita, contradições eventualmente podem surgir nesses programas. Das maneiras de lidar com isso, a abordagem paraconsistente, que permite raciocinar significativamente mesmo diante de informações contraditórias, é a que tem sido mais amplamente aceita em programação em lógica. Várias semânticas que seguem essa abordagem foram definidas, sendo considerada a semântica bem fundada estendida paraconsistente (abreviadamente $WFSX_P$) uma das mais intuitivas. Entretanto, essa semântica não está de acordo com o princípio da exceção primeiro, que privilegia a conclusão da exceção de uma regra em detrimento da conclusão da própria regra. Como conseqüência, resultados pouco desejados são obtidos.

Para resolver esse problema, neste trabalho, é introduzido o operador epistêmico "?", definido na Lógica da Inconsistência Epistêmica (LEI), no alfabeto de programas em lógica estendidos. Os programas resultantes são chamados de programas em lógica estendidos da inconsistência epistêmica. Uma semântica é definida para tais programas: semântica bem fundada estendida da inconsistência epistêmica (abreviadamente $WFSX_{EI}$). Alguns exemplos são mostrados em que $WFSX_{EI}$, por estar de acordo com o princípio da exceção primeiro, apresenta soluções mais significativas do que $WFSX_P$.

Além disso, foi definido e implementado um procedimento de derivação que é demonstrado ser correto e completo com relação a $WFSX_{EI}$. Outros importantes resultados tanto de cunho prático quanto teórico foram também demonstrados, evidenciando a naturalidade e intuitividade da semântica proposta.

Dedico esta obra a todas as pessoas que me amam!!!

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Uma das atribuições mais instigantes da Inteligência Artificial (IA) refere-se à construção de máquinas que sejam capazes de manipular o conhecimento apropriadamente. Para tanto, é necessário uma linguagem não ambígua capaz de expressar esse conhecimento e ainda permitir que mecanismos de inferência descrevam o raciocínio desejado.

Na busca desse objetivo, uma das primeiras idéias se deve a Mc Carthy [55], que propôs a utilização de fórmulas lógicas como base para a linguagem de representação do conhecimento. Por possuir uma semântica bem definida e um mecanismo de inferência bem entendido e poderoso, a lógica clássica de primeira ordem serviu inicialmente a esse propósito.

Caracteristicamente, a lógica clássica de primeira ordem é dedutiva, portanto é analítica e conservativa. Isso quer dizer que as conclusões que seguem por raciocínio dedutivo de um conjunto de premissas estão de um certo modo já contidas nesse conjunto. Assim, pode-se concluir que na dedução, não se produz nova informação. Por outro lado, o raciocínio dedutivo é correto, no sentido de que preserva a verdade, isto é, uma vez que as premissas sejam verdadeiras, as conclusões resultantes também serão verdadeiras.

Além disso, inferências dedutivas não são retratáveis quando confrontadas com novas informações. Assim, uma conclusão obtida anteriormente nunca será invalidada com a inclusão de premissas. Por essa razão, o raciocínio dedutivo é monotônico e é adequado para lidar com o raciocínio matemático.

No entanto, em situações que envolvem o raciocínio do senso comum[1], as conclusões são obtidas a partir do conhecimento disponível, que é muitas vezes incompleto. Também é sabido que ele é indutivo, portanto sintético e ampliativo, ou seja, o contéudo das conclusões transcende o conteúdo das premissas, produzindo novas informações. Entretanto, não é garantido que o raciocínio indutivo é correto: a verdade das premissas não conduz à

[1]Neste trabalho, será utilizado o termo raciocínio do senso comum em oposição ao raciocínio matemático.

verdade das conclusões. Uma outra de suas características é a não monotonicidade, logo o acréscimo de novas informações pode conduzir a evidências que invalidam as conclusões obtidas previamente.

A par de tais considerações, percebe-se que o raciocínio dedutivo e monotônico da lógica clássica de primeira ordem não é suficientemente moldável para ser utilizado como ferramenta na formalização e tratamento computacional do raciocínio do senso comum em IA. Para tanto, deve-se buscar formas não monotônicas de raciocínio.

Enquanto isso, uma outra linha de pesquisa, iniciada em [36, 38, 46], conduzia à utilização da lógica como linguagem de programação. O esforço desses trabalhos resultou no desenvolvimento de estratégias simples e eficientes para provas de teoremas tais como a resolução *SLD* aplicada às cláusulas Horn [40]. Nesse período, também foi definido e implementado a primeira linguagem de programação em lógica, o Prolog [19, 46]. A partir de então, passou-se a utilizar o conceito de programação declarativa (em oposição à programação procedural) em Ciência da Computação. Idealmente, na programação declarativa, deve-se estabelecer somente o significado declarativo do programa enquanto que os aspectos procedurais da execução do programa são manipulados automaticamente. Esse princípio foi apresentado por Kowalski em [42] sob a relação $ALGORITMO = LÓGICA + CONTROLE$.

Para alcançar esse ideal, é necessário uma definição precisa da semântica declarativa de tais programas. Entretanto, por admitirem somentes conclusões positivas, as cláusulas Horn possuem semântica monotônica e, assim, são inadequadas para aplicações em IA ou que envolvam o raciocínio do senso comum. Esse obstáculo é suplantado com a introdução do operador *not* para representar a negação por falha ou negação default, segundo o qual é assumido como falso o que não é (finitamente) provado ser verdadeiro. Com o acréscimo do operador *not* ao alfabeto das cláusulas Horn, é definido o conceito de programação normal em lógica.

Claramente, o operador *not* possui um caráter não monotônico. Como tiveram inicialmente desenvolvimentos distintos, ambos os formalismos, raciocínio não monotônico e programação em lógica, foram mutuamente beneficiados. De um lado, pode-se analisar programas em lógica como tipos especiais de teorias não monotônicas, possibilitando a apropriação de muitos resultados nessa área e a conseqüente utilização desse material na busca de semânticas mais significativas para programas em lógica.

Por outro lado, apesar dos programas em lógica constituírem somente uma subclasse das teorias não monotônicas, eles são suficientemente expressivos para formalização de muitos problemas além de prover subsídios para novas formalizações. Como programas em lógica admitem mecanismos computacionais eficientes, eles podem ser utilizados como engenhos de inferência de formalismos não monotônicos.

Desafortunadamente, a representação da informação negativa quando proporcionada somente pela negação default é uma restrição crítica, pois não há maneira eficiente de explicitamente representar a informação negativa. Nesse sentido, vários trabalhos [63, 64, 29, 45, 30, 41, 43, 71, 73, 94, 95] apontaram a necessidade de estender o alfabeto de programas em lógica com um segundo tipo de negação, designado por $\neg$, para representar a negação explícita. Tais programas são chamados de programas em lógica estendidos.

Naturalmente, com a introdução da negação explícita, as semânticas para esses programas devem ser capazes de lidar com contradições. Por seu turno, em situações que envolvem o raciocínio do senso comum ou em aplicações de Inteligência Artificial, uma contradição pode ser resultado da imprecisão do conhecimento. Assim, uma contradição nem sempre é indicativo de erro, mas pode ser reflexo da necessidade de informação mais acurada. Por isso, tem sido crescente a aceitação de semânticas paraconsistentes em programação em lógica [21]. No entanto, muitas dessas semânticas falham por não estarem definidas para todo programa ou por não estabelecerem relação alguma entre as duas formas de negação: default e explícita. Como conseqüência, resultados pouco desejáveis são obtidos.

Nesse cenário, foi proposta a $WFSX_P$ (*Paraconsistent Well-Founded Semantics with eXplicit negation*) [4, 20]. Essa semântica é uma generalização da semântica bem fundada *WFS* [93] e da semântica bem fundada com negação explícita *WFSX* [68, 2], que são respectivamente as semânticas mais importantes para programas normais em lógica e programas em lógica estendidos. Em programas em lógica estendidos não contraditórios, $WFSX_P$ coincide com *WFSX* e em programas normais em lógica, $WFSX_P$ coincide com *WFS*.

$WFSX_P$ herda muitas das boas características de *WFSX* tais como a propriedade de estar unicamente definida para todo programa e a de estar em sintonia com o princípio da coerência ($\neg L \Rightarrow not\ L$). Também, possui a capacidade de detectar contradição e literais dependentes de contradição sem se trivializar. Uma das originalidades dessa semântica é a obediência para o caso paraconsistente das propriedades estruturais apresentadas por Dix em [25, 24]. A obediência a tais propriedades por $WFSX_P$, demonstra a sua superioridade com relação às outras semânticas paraconsistentes encontradas na literatura sobre programação em lógica estendida [20]. No entanto, por interpretar regras com literais defaults no corpo como defaults no sentido de Reiter [82], $WFSX_P$ possui algumas deficiências:

Exemplo 1.1 (Paradoxo do Barbeiro) Seja P o programa em lógica estendido abaixo, que representa o paradoxo do barbeiro:

$$P = shave(b, x) \leftarrow not\ shave(x, x),$$

em que x é uma variável, b é uma constante que representa o barbeiro e *shave* é um predicado que representa o ato de barbear-se.

A única regra desse programa pode ser lida da seguinte maneira: "*se não há nenhuma evidência de que um indivíduo faz a sua barba, então o barbeiro faz a barba dele*". Segundo $WFSX_P$, o literal $shave(b,b)$ possui um valor verdade indefinido, ou seja, nada pode ser dito sobre o ato do barbeiro fazer a sua barba ou não em P.

Agora, suponha que P' seja o programa resultante de P ao adicionar o fato $\neg shave(b,b)$. Em $WFSX_P$, devido ao princípio da coerência, $\neg shave(b,b)$ implica $not\ shave(b,b)$ e com isso conclui-se $shave(b,b)$ em P'. Desse modo, tem-se um modelo contraditório com $\neg shave(b,b)$ e $shave(b,b)$, que é pouco intuitivo.

Além do mais, para qualquer programa P cujos modelos $WFSX_P$ não são contraditórios, $WFSX_P$ é correto com relação a intersecção de todas as extensões de Reiter[2] para a teoria default T correspondente a P [70]. Como resultado, alguns problemas verificados pela lógica default de Reiter persistem em $WFSX_P$:

Exemplo 1.2 Considere o seguinte exemplo apresentado por Morris em [60]:

Animais normalmente não voam;
Animais alados são exceção a essa regra; eles voam;
Pássaro são animais;
Pássaros normalmente têm asas;
Teo[3] é um pássaro.

Usando a lógica default de Reiter, essas declarações podem ser axiomatizadas como segue:

1. $\dfrac{animal(X):\neg voa(X)\wedge\neg alado(X)}{\neg voa(X)}$
2. $alado(X) \Rightarrow voa(X)$
3. $passaro(X) \Rightarrow animal(X)$
4. $\dfrac{passaro(X):alado(X)}{alado(X)}$
5. $passaro(Teo)$

Segundo Reiter, essa teoria possui duas extensões:
$E_1 = \{passaro(Teo), animal(Teo), voa(Teo), alado(Teo)\}$;
$E_2 = \{passaro(Teo), animal(Teo), \neg voa(Teo), \neg alado(Teo)\}$.

[2]Para informações adicionais sobre a lógica default de Reiter, confira a seção A.1 do apêndice A.
[3]No original, Tweet.

Observe que E_2 apresenta o resultado não desejado de que *Teo* é pássaro, mas não tem asas. De acordo com Marcelino Pequeno em [65], isso ocorre porque a semântica da lógica default de Reiter não está de acordo com o princípio da exceção primeiro, que privilegia a conclusão da exceção de uma regra default em detrimento da conclusão da própria regra. No exemplo 1.2, ser alado é uma exceção à regra geral de que animais normalmente não voam. Para estar de acordo com o princípio da exceção primeiro, a conclusão de que *Teo* é alado pela regra 4 deve bloquear a conclusão de que *Teo* não voa pela regra 1. Como isso não é feito na semântica da lógica default de Reiter, a extensão anômala E_2 é obtida. No tocante a programação em lógica estendida, o problema de Morris pode ser codificado da seguinte maneira:

Exemplo 1.3 Seja P o programa em lógica estendido para o exemplo 1.2.

$\neg voa(x) \leftarrow animal(x), not\ voa(x), not\ alado(x)$
$voa(x) \leftarrow alado(x)$
$\neg alado(x) \leftarrow \neg voa(x)$[4]
$animal(x) \leftarrow passaro(x)$
$\neg passaro(x) \leftarrow \neg animal(x)$
$alado(x) \leftarrow passaro(x), not\ \neg alado(x)$
$passaro(Teo)$

Desse programa, tem-se que em $WFSX_P$, $passaro(Teo)$ e $animal(Teo)$ são verdadeiros e $voa(Teo)$ e $alado(Teo)$ são indefinidos, ou seja, nenhuma informação é fornecida sobre a capacidade de *Teo*, que é pássaro, de voar ou de ser alado.

1.1 Solução proposta

Por sua vez, em [66], é apresentado *IDL*[5] (*Inconsistent Default Logic*). Trata-se de uma lógica default que agrupa visões conflitantes numa única extensão. Um default *IDL* é do tipo

$$\frac{\alpha : \beta; \gamma}{\beta?}$$

em que α é o antecedente da regra, β é a sua condição default, γ compõe a parte seminormal, sendo usada para expressar uma condição de exceção e $\beta?$ é a conclusão da regra.

Em *IDL*, parte-se do princípio de que uma conclusão refutável não deve ter o mesmo *status* epistêmico de uma conclusão irrefutável, que é obtida através de dedução. Assim,

[4] Quando possível, para representar a contrapositiva, deve-se explicitamente introduzi-la como uma regra do programa.
[5] Confira seção A.3.

em *IDL*, conclusões refutáveis são sufixadas por um "?" para serem distinguidas das demais. Sob essa argumentação, pode-se ainda acrescentar que uma teoria T em *IDL* é contraditória se e somente se pelo menos um dos seguintes pares de fórmulas for verificado em T:

- **Contradição forte:** $(F, \neg F)$;
- **Contradição fraca:** $(F?, \neg F)$ ou $(F, (\neg F)?)$.

Dessa definição de contradição em teorias *IDL*, é importante destacar que o par de fórmulas $(F?, \neg F?)$ não corresponde a uma contradição. Um outro ponto a ser salientado é que defaults *IDL* podem ser bloqueados tanto pela condição default quanto pela parte seminormal. Pela condição default, é necessário uma contradição forte $\neg\beta$ para impedir a aplicação da regra. No tocante a parte seminormal, uma contradição fraca $(\neg\gamma)?$ é o suficiente.

Retornando ao exemplo de Morris, constata-se como essa definição de default *IDL*, por obedecer ao princípio da exceção primeiro, resolve os problemas verificados na lógica default de Reiter:

Exemplo 1.4 Seja T a teoria correspondente em *IDL* ao problema apresentado no exemplo 1.2.

$$T = \begin{cases} (1) & \dfrac{animal(x) : \neg voa(x) \wedge \neg alado(x)}{\neg voa(x)?} \\ (2) & alado(x) \Rightarrow voa(x) \\ (3) & passaro(x) \Rightarrow animal(x) \\ (4) & \dfrac{passaro(x) : alado(x)}{alado(x)?} \\ (5) & passaro(Teo) \end{cases}$$

De $passaro(Teo)$, obtém-se $animal(Teo)$ e conseqüentemente, da regra (1), conclui-se $\neg voa(Teo)?$. Por *modus tollens* sobre (2), tem-se $\neg alado(Teo?)$. Agora, a regra(4) não é mais bloqueada. Para tanto precisaria de uma contradição forte. Com isso, $alado(Teo)?$ é obtido, sendo o suficiente para bloquear a aplicação da regra(1) por meio da condição de exceção que precisa apenas de uma contradição fraca. Sendo assim, $\neg voa(Teo)?$ e $\neg alado(Teo)$ são descartados e o resultado esperado de que *Teo* é um pássaro alado e voa é obtido.

Em [67], é apresentado LEI[6] (*Logic of Epistemic Inconsistency*), que é uma lógica paraconsistente capaz de extrair conclusões significativas de contradições resultantes do raciocínio sobre o conhecimento incompleto. LEI tem sido utilizado como base monotônica para *IDL*, sendo que a lógica resultante é chamada de *IDL-LEI*.

[6] Confira seção A.2.

Para sobrepujar as deficiências de $WFSX_P$ motivadas pelos exemplos 1.1 e 1.2, é proposto neste trabalho interpretar as regras que possuem literais default no corpo como defaults *IDL-LEI*. Isso requer uma nova interpretação para o operador *not* e a introdução do operador epistêmico "?" no alfabeto de programas em lógica estendidos. Como resultado, tem-se os programas em lógica estendidos da inconsistência epistêmica.

Para atribuir significado a tais programas, é definida uma semântica bem fundada: $WFSX_{EI}$ (*Well-Founded Semântics with eXplicit negation of Epistemic Inconsistency*). Nesta dissertação, $WFSX_{EI}$ é caracterizado através de operadores de pontos fixos em três maneiras distintas, sendo provada a equivalência dessas definições. Para ressaltar a sua importância, são mostrados exemplos clássicos envolvendo raciocínio não monotônico em que $WFSX_{EI}$ proporciona soluções mais significativas do que $WFSX_P$. Como contrapartida procedural à caracterização declarativa de $WFSX_{EI}$, é definido e implementado SLX_{EI} (*Selection rules driven Linear resolution for eXtended logic programs of Epistemic Inconsistency*). Em seguida, é provada a corretude e a completude de SLX_{EI} com relação a $WFSX_{EI}$.

1.2 Principais contribuições

Neste trabalho, significativos resultados foram obtidos tanto de natureza prática quanto teórica. Suas principais contribuições são elencadas abaixo:

- Introdução do operador epistêmico "?" em programação em lógica e interpretação de regras com literais default no corpo como defaults no sentido de *IDL-LEI*, resultando na definição do conceito de programação em lógica estendida da inconsistência epistêmica;
- Caracterização de $WFSX_{EI}$ através de operadores de ponto fixo em três maneiras distintas. É provada a equivalência dessas definições;
- Comparação entre $WFSX_{EI}$ e $WFSX_P$ a partir das respectivas soluções proporcionadas para alguns problemas clássicos de raciocínio não monotônico;
- Demonstração de um resultado de corretude envolvendo $WFSX_{EI}$ e as extensões de *IDL-LEI*;
- Demonstração que o problema de decisão em $WFSX_{EI}$ para programas sem símbolos funcionais possui complexidade polinomial;
- Definição e implementação de um procedimento de derivação, SLX_{EI}, para programas em lógica estendidos da inconsistência epistêmica, que é demonstrado ser correto e completo (teoricamente) com relação a $WFSX_{EI}$.

1.3 Plano deste trabalho

No que tange à estrutura deste trabalho, ela pode ser esboçada da seguinte maneira:

O **capítulo 2** mostra alguns conceitos lógicos elementares. Na seqüência é traçado um histórico sobre alguns procedimentos de prova até a definição da resolução linear. Alguns conceitos referentes aos operadores de pontos fixos são também exibidos.

O **capítulo 3** inicia com a apresentação da lógica de primeira ordem restrita às cláusulas Horn para depois mostrar uma descrição do processo evolutivo da programação em lógica. O capítulo termina com uma discussão sobre a importância do raciocínio paraconsistente em programação em lógica.

No **capítulo 4**, é feita uma análise das principais semânticas paraconsistentes para programas em lógica, ressaltando a elegância e aplicabilidade de $WFSX_P$.

No **capítulo 5**, são formalizadas algumas das principais idéias defendidas nesta dissertação. Ele inicia com a exibição de uma linguagem para programas em lógica da inconsistência epistêmica. Em seguida, são mostradas três definições equivalentes para $WFSX_{EI}$. Também são elencados alguns exemplos clássicos de raciocínio não monotônico em que $WFSX_{EI}$ proporciona soluções mais significativas do que $WFSX_P$. Após isso, é demonstrado um resultado de corretude envolvendo $WFSX_{EI}$ e as extensões de *IDL-LEI*. Por fim, é demonstrado o caráter polinomial do problema de decisão em $WFSX_{EI}$ para programas sem símbolos funcionais.

O **capítulo 6** é voltado para a definição do procedimento de derivação SLX_{EI} para programas sem variáveis, que é demonstrado ser correto e completo com relação a $WFSX_{EI}$.

No **capítulo 7**, são exibidas as conclusões deste trabalho bem como futuras frentes de pesquisas que podem ser erigidas com as idéias aqui esmiuçadas.

Parte deste trabalho foi publicado em

- [1] João F. L. Alcântara e Marcelino Pequeno. $WFSX_{EI}$ - Uma semântica bem fundada estendida da inconsistência epistêmica. *Anais do II Encontro Nacional de Inteligência Artificial (ENIA).* (Díbio Borges e Ariadne Carvalho, eds), p.255 -271, Edições EntreLugar, 1999.

Capítulo 2

Aspectos Elementares sobre Automação do Raciocínio

Um dos desafios da Inteligência Artificial está na construção de ferramentas "inteligentes" que sejam capazes de manusear o conhecimento adequadamente. Para alcançar esse objetivo, comumente é preciso uma grande quantidade de dados além de mecanismos eficientes de representação do conhecimento e do raciocínio. Nesse cenário, aflora a importância dos provadores mecânicos de teoremas.

A idéia precursora para a construção de tais engenhos remonta aos trabalhos de Leibniz(1646-1716) sobre procedimentos de decisão geral para provar teoremas. Essa idéia for retomada por Peano na virada do século XX e pela escola de Hilbert nos anos vinte desse século. Em 1930, Herbrand propôs um importante teorema que facilita enormemente a prova mecânica. Mesmo assim o seu método tomava uma grande quantidade de tempo para ser executado manualmente. Com o advento dos computadores digitais, o interesse pela prova automática foi revigorado. Em 1960, Gilmore implementou o procedimento de Herbrand em um computador digital, sendo aperfeiçoado no mesmo ano por Davis e Putnam.

A principal contribuição em prova automática de teoremas foi proporcionada por Robinson em 1965 através do princípio da resolução. Esse princípio trata-se de uma única regra de inferência altamente eficiente e facilmente implementável. Desde então, muitos refinamentos vêm sendo feitos. Um de especial interesse é a resolução linear. Tal refinamento é um dos mecanismos elementares de inferência da programação em lógica, que será abordada no próximo capítulo.

Para uma melhor compreensão da importância desses métodos mecânicos de prova, alguns conceitos elementares de lógica de primeira ordem são necessários. Esse assunto é relatado na seção 2.1, que inicia com a apresentação da sintaxe e da semântica da lógica de primeira ordem. Em seguida, é encetada uma discussão sobre o teorema de Herbrand e

alguns resultados importantes são mostrados. Ainda nessa seção, é introduzido o conceito de forma padrão de Skolem. Na seção 2.2, são definidas as noções fundamentais sobre teoria da prova. Na seqüência, são apresentadas a teoria da resolução, o método da resolução linear e procedimentos de refutação linear. Este capítulo termina com alguns conceitos envolvendo pontos fixos e operadores monotônicos (seção 2.3).

2.1 Conceitos fundamentais de lógica de primeira ordem

No século XVII, Leibniz proporcionou uma base matemática para o raciocínio lógico. Em seus trabalhos, ele idealizou uma linguagem universal em que uma declaração pudesse ser representada diretamente sem ambigüidades, termos vagos ou figuras de linguagem como é característico na linguagem natural. Assim, através de operações mecânicas envolvendo manipulações de símbolos seria possível verificar a validade de uma sentença da lógica. No entanto, naquele período, o raciocínio lógico baseava-se na análise das sentenças da lógica sob seus aspectos gramaticais, principalmente em termos de sujeito e predicado. Tais sistemas demonstraram ser ineficazes para lidar até mesmo com deduções matemáticas elementares.

Para contornar essa situação, Frege formalizou um meio de analisar as sentenças com base em conectivos e quantificadores. A teoria lógica fundamentada nessa análise é denominada de lógica de primeira ordem, que é completamente adequada para o raciocínio matemático.

A lógica de primeira ordem possui grande importância para pesquisas em Inteligência Artificial, servindo tanto como linguagem para formalizar o raciocínio quanto uma fonte de técnicas e de ferramentas analíticas que proporcionam fundamentos matemáticos e conceituais utilizados no desenvolvimento de sistemas de Inteligência Artificial.

Dois aspectos devem ser levados em consideração na lógica de primeira ordem: sintaxe e semântica. Na sintaxe, são definidas as fórmulas bem formadas que podem ser obtidas a partir de uma linguagem. Nela também são especificados os meios pelos quais essa linguagem possa ser manipulada por mecanismos automáticos de prova. Por sua vez, na semântica, são definidas técnicas precisas de atribuir significado para fórmulas bem formadas e para os símbolos que elas contêm.

2.1.1 Linguagem

Definição 2.1 *Na lógica de primeira ordem, um alfabeto $\mathcal{A}$ é constituído de*

1. *Infinitos símbolos variáveis*

2. *Símbolos para operações lógicas:* $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \exists, \forall$*;*

3. *Sinais de pontuação: "(" "," ")";*

4. *Símbolos predicados;*

5. *Símbolos funcionais;*

6. *Símbolos constantes.*

Pressupõe-se que as categorias (1)-(6) são disjuntas entre si, sendo que as três primeiras são fixas para todo alfabeto. Em contrapartida, as categorias (4)-(6) são constituídas por símbolos não lógicos e podem variar de alfabeto para alfabeto.

Os conectivos $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$ representam respectivamente a negação, disjunção, conjunção, implicação e a equivalência. Por sua vez, $\exists$ representa o quantificador existencial ao passo que $\forall$ representa o quantificador universal. No transcorrer deste trabalho serão adotadas algumas convenções notacionais: varíaveis serão usualmente denotadas pelas letras u, v, w, x, y e z (possivelmente subescritas); constantes, pelas letras a, b e c (possivelmente subescritas); símbolos funcionais com aridade > 0, pelas letras f, g e h (possivelmente subescritas); símbolos predicados de aridade ≥ 0, pelas letras $p, q,$ e r (possivelmente subescritas). Em algumas situações, por motivo de clareza, essa convenção não será aplicada. Nesses casos, eventuais ambigüidades serão dirimidas pelo contexto.

Definição 2.2 *Um termo é definido indutivamente como segue:*

- *Um símbolo constante é um termo;*
- *Uma variável é um termo;*
- *Se f é um símbolo funcional e $t_1 \ldots t_n$ são termos, então $f(t_1 \ldots t_n)$ também é um termo.*

Definição 2.3 *Fórmulas bem formadas (ou abreviadamente fórmulas) são definidas indutivamente como segue:*

- *Se p é um símbolo predicado enário e $t_1, \ldots, t_n$ são termos, então $p(t_1 \ldots t_n)$ é uma fórmula, também chamada de fórmula atômica ou átomo;*
- *Se F e G são fórmulas, então $(\neg F)$, $(F \vee G)$, $(F \wedge G)$, $(F \Rightarrow G)$ e $(F \Leftrightarrow G)$ também são fórmulas;*
- *Se F é uma fórmula e x uma variável, então $(\exists x)F$ e $(\forall x)F$ são fórmulas.*

Quando isto for conveniente, alguns parênteses serão omitidos de acordo com a convenção utilizada em [15]. Uma linguagem de primeira ordem pode então ser definida como segue:

Definição 2.4 *A linguagem de primeira ordem $\mathcal{L}$ dada por um alfabeto $\mathcal{A}$ é o conjunto de todas as fórmulas obtidas a partir de símbolos de $\mathcal{A}$.*

Na próxima definição é estabelecido o escopo de um quantificador bem como a diferença entre uma variável livre e uma limitada:

Definição 2.5 *O escopo de $\forall x$ (resp. $\exists x$) na fórmula $\forall xF$ (resp. $\exists xF$) é F. Uma ocorrência de uma varíavel é limitada se e somente se a ocorrência está dentro do escopo do quantificador que emprega essa variável ou a ocorrência faz parte do próprio quantificador. Caso contrário, a ocorrência dessa variável é livre.*

Definição 2.6 *Um literal é um átomo ou a negação de um átomo, que são respectivamente chamados de literal positivo e literal negativo.*

Algumas tipos de fórmulas de primeira ordem recebem nomes especiais. Uma delas é a forma normal conjuntiva prenex:

Definição 2.7 *Uma fórmula da lógica de primeira ordem está na forma normal conjuntiva prenex se ela é do tipo*

$$(Q_1x_1)\ldots(Q_kx_k)(L_{11}\vee\ldots\vee L_{1m_1})\wedge\ldots\wedge(L_{n1}\vee\ldots\vee L_{nm_j}),$$

em que cada (Q_i) com $1\leq i\leq k$, é $(\forall)$ ou $(\exists)$ e cada L_{ij} é um literal. $(Q_1x_1)\ldots(Q_nx_n)$ é chamado de prefixo e $(L_{11}\vee\ldots\vee L_{1m_1})\wedge\ldots\wedge(L_{n1}\vee\ldots\vee L_{nm_j})$, de matriz da fórmula.

Definição 2.8 *Uma cláusula é uma fórmula do tipo*

$$\forall x_1\ldots\forall x_m(L_1\vee\ldots\vee L_n),$$

em que cada L_i é um literal e $x_1,\ldots x_m$ são todas as variáveis que ocorrem em $L_1\vee\ldots\vee L_n$

Quando conveniente, uma cláusula C será representada simplesmente por uma disjunção de literais, ficando implícito que toda variável de C está dentro do escopo de algum quantificador universal de C. Uma cláusula que não contém literal é chamada de vazia e é denotada pelo símbolo $\square$. Um conjunto finito S de cláusulas é considerado como uma conjunção de todas as cláusulas em S, em que toda variável em S é assumida estar governada por um quantificador universal. Desse modo, uma fórmula na forma normal conjuntiva sem quantificadores existenciais pode ser vista como um conjunto de cláusulas.

Exemplo 2.9 $(\forall x)((p(f(x))\vee q(f(x),x,g(x)))\wedge(r(x,x)\vee\neg p(g(x),f(x))))$ é uma fórmula que está na forma normal conjuntiva. Alternativamente, ela pode ser representada por um conjunto de cláusulas:

$$\{p(f(x)\vee q(f(x),x,g(x)),r(x,x)\vee\neg p(g(x),f(x))\}.$$

2.1.2 Semântica

Para uma análise puramente matemática das noções lógicas de verdade e falsidade de uma fórmula, é preciso determinar o significado de cada um de seus símbolos. Os quantificadores, conectivos e sinais de pontuação possuem significados fixos; em contrapartida, os símbolos constantes, predicados e funcionais possuem significados que podem variar. Esse significado é especificado por uma interpretação cuja definição é apresentada abaixo:

Definição 2.10 (Interpretação) *Uma interpretação I de uma linguagem de primeira ordem consiste de um conjunto não vazio $\mathcal{D}$, chamado de domínio de I e de uma atribuição tal que*

- *Para cada símbolo constante c, é atribuído um elemento $c_I \in D$;*
- *Para cada símbolo funcional f, é atribuído uma mapeamento f_I de D^n em D, em que $D^n = \{(x_1,\ldots,x_n) \mid x_1 \in D,\ldots x_n \in D\}$;*
- *Para cada símbolo predicado p, é atribuído um mapeamento p_I de D^n em $\{0,1\}$, que respectivamente designa os valores lógicos falso e verdadeiro.*

Uma *valoração* com respeito a um domínio D é uma seqüência infinita $v_1, v_2, \ldots$ de elementos de D. Uma valoração de uma interpretação I é uma valoração com respeito ao domínio de I. Além do mais, diz-se que v_i é o *valor* da variável x_i com respeito a valoração v. Se v e w são valorações com respeito ao mesmo domínio D, então $v \equiv_k w$ se $v_i = w_i$ para todo $i \neq k$. Todas as valorações de todas as interpretações para as fórmulas de uma linguagem $\mathcal{L}$ são agrupadas por uma função de valoração V:

Definição 2.11 *Sejam I uma interpretação com domínio D e v é uma valoração de I.*

- *Seja ν um termo, então $V(\nu, I, v) \in D$ é uma função tal que:*
  - *Se ν é um símbolo constante c, então $V(\nu, I, v) = c_I$;*
  - *Se ν é a variável x_j, então $V(\nu, I, v) = v_j$;*
  - *Se $\nu = f(x_1,\ldots,x_n)$ e $V(x_i, I, v) = t_i \in D$ para $i = 1,\ldots,n$, então*

$$V(\nu, I, v) = f_I(t_1,\ldots,t_n).$$

- *Seja γ uma fórmula, então $V(\gamma, I, v) \in \{1, 0\}$ é uma função tal que:*
  - *Se $\gamma = p(x_1, \ldots, x_n)$ é um átomo e $V(x_i, I, v) = t_i \in D$ para $i = 1, \ldots, n$, então*

$$V(\gamma, I, v) = p_I(t_1, \ldots, t_n).$$

  - *Se $\gamma = \neg\alpha$, então*

$$V(\gamma, I, v) = \begin{cases} 0 & \text{se } V(\alpha, I, v) = 1 \\ 1 & \text{caso contrário} \end{cases}$$

  - *Se $\gamma = \alpha \Rightarrow \beta$, então*

$$V(\gamma, I, v) = \begin{cases} 0 & \text{se } V(\alpha, I, v) = 1 \text{ e } V(\beta, I, v) = 0 \\ 1 & \text{caso contrário} \end{cases}$$

  - *Se $\gamma = \alpha \vee \beta$, então*

$$V(\gamma, I, v) = \begin{cases} 0 & \text{se } V(\alpha, I, v) = 0 \text{ e } V(\beta, I, v) = 0 \\ 1 & \text{caso contrário} \end{cases}$$

  - *Se $\gamma = \alpha \wedge \beta$, então*

$$V(\gamma, I, v) = \begin{cases} 1 & \text{se } V(\alpha, I, v) = 1 \text{ e } V(\beta, I, v) = 1 \\ 0 & \text{caso contrário} \end{cases}$$

  - *Se $\gamma = \alpha \Leftrightarrow \beta$, então*

$$V(\gamma, I, v) = \begin{cases} 1 & \text{se } V(\alpha, I, v) = V(\beta, I, v) \\ 0 & \text{caso contrário} \end{cases}$$

  - *Se $\gamma = (\exists x_j)\alpha$, então $V(\gamma, I, v) = 1$ se há uma valoração w de D tal que $w \equiv_j v$ e $V(\gamma, I, w) = 1$; caso contrário $V(\gamma, I, v) = 0$;*
  - *Se $\gamma = (\forall x_j)\alpha$, então $V(\gamma, I, v) = 0$ se há uma valoração w de D tal que $w \equiv_j v$ e $V(\gamma, I, w) = 0$; caso contrário $V(\gamma, I, v) = 1$;*

Dessa definição, emerge que a função de valoração V depende do valor de v_i somente quando a variável x_i possui uma ocorrência livre no termo ν ou na fórmula *gamma*. Com isso, sendo γ uma sentença, $V(\gamma, I, v)$ não depende de v e pode ser abreviada para $V(\gamma, I)$.

Definição 2.12 (Modelo) *Sejam Γ um conjunto de sentenças da lógica de primeira ordem $\mathcal{L}$ e I uma interpretação de $\mathcal{L}$. I é um modelo para Γ sss $V(\gamma, I) = 1$ para qualquer $\gamma \in \Gamma$.*

Feitas essas ponderações, pode-se apresentar as seguintes relações lógicas:

Definição 2.13 *Seja* Γ *um conjunto de sentenças da lógica de primeira ordem* $\mathcal{L}$.

Γ *é satisfatível sss* $\mathcal{L}$ *tem uma interpretação que é um modelo para* Γ;

Γ *é válido sss toda interpretação de* $\mathcal{L}$ *é um modelo para* Γ;

Γ *é insatisfatível sss não há interpretação de* $\mathcal{L}$ *que seja um modelo para* Γ.

Agora, já se está em condições de definir a importante noção de conseqüência lógica:

Definição 2.14 (Conseqüência lógica) *Uma sentença* γ *é uma conseqüência lógica de um conjunto de sentenças* Γ, *denotado por* $\Gamma \models \gamma$ *sss qualquer modelo para* Γ *é também um modelo para* γ.

Em tais casos, também se diz que Γ veicula γ ou que γ segue de Γ. Para indicar que γ não segue de Γ, é utilizado $\Gamma \not\models \gamma$. As sentenças pertencentes a Γ são chamadas de premissas ou hipóteses e γ é chamado de conclusão. Se $\Gamma = \emptyset$, escreve-se apenas $\models \gamma$, ou seja, I satisfaz γ para qualquer interpretação I. Quando isso ocorre, γ é chamada de *validade*.

Alternativamente, pode-se verificar se uma fórmula é conseqüência lógica de um conjunto de fórmulas através da proposição abaixo cuja prova pode ser encontrada em [50]:

Proposição 2.15 *Sejam* Γ *um conjunto de sentenças e* γ *uma sentença de uma linguagem de primeira ordem* $\mathcal{L}$. γ *é uma conseqüência lógica de* Γ *sss* $\Gamma \cup \{\neg\gamma\}$ *é insatisfatível.*

2.1.3 Interpretações de Herbrand

Por definição, um conjunto de sentenças Γ é insatisfatível se e somente se não há nenhuma interpretação de $\mathcal{L}$ que seja um modelo de Γ. Todavia, em geral, há um número infinito de domínios na lógica de primeira ordem e conseqüentemente, há um número infinito de interpretações. Em trabalhos independentes, Church [16] e Turing [90] demonstraram que não existe um algoritmo para determinar se uma fórmula arbitrária ou um conjunto arbitrário de fórmulas é válido ou insatisfatível. Desse modo, seria muito útil se houvesse um domínio especial tal que Γ fosse insatisfatível se e somente se Γ fosse falso sob todas as interpretações com esse domínio. Caso Γ seja constituído somente por cláusulas, esse domínio existe e é chamado de universo de Herbrand.

Na seqüência, uma *expressão* quer dizer um termo, um conjunto de termos, um átomo, um conjunto de átomos, um literal, um conjunto de literais, uma cláusula ou um conjunto de cláusulas. Uma expressão é *básica*[1] se e somente se nenhuma variável ocorre nessa expressão.

[1] Do inglês ground.

Definição 2.16 (Universo de Herbrand) *Seja* Γ *um conjunto de cláusulas. O universo de Herbrand* $\mathcal{H}_U$ *para uma linguagem de primeira ordem* $\mathcal{L}$ *referente a* Γ *é constituído de todos os termos básicos que podem ser construídos a partir das constantes e símbolos funcionais que ocorrem em* Γ*. Se não há nenhuma constante em* Γ*, é escolhida uma constante arbitrária para que os termos básicos possam ser construídos.*

Definição 2.17 (Base de Herbrand) *Seja* Γ *um conjunto de cláusulas. A base de Herbrand* $\mathcal{H}_B$ *para uma linguagem de primeira ordem* $\mathcal{L}$ *referente a* Γ *é constituída de todos os átomos básicos que podem ser obtidos de* Γ *usando termos de* $\mathcal{H}_U$ *de* Γ*.*

Exemplo 2.18 Seja $\Gamma = \{p(x), q(f(y)) \vee r(y)\}$ um conjunto de cláusulas. Então $\mathcal{H}_U = \{a, f(a), f(f(a)), \ldots\}$ é o universo de Herbrand para $\mathcal{L}$ e $\{p(a), p(f(a)), p(f(f(a))), \ldots, q(a), q(f(a)), q(f(f(a))), \ldots, r(a), r(f(a)), r(f(f(a))), \ldots\}$ é a base de Herbrand $\mathcal{H}_B$ para $\mathcal{L}$, em que a é uma constante arbitrária.

Utilizando o universo de Herbrand como domínio de uma linguagem de primeira ordem $\mathcal{L}$, um tipo especial de interpretação e modelo pode ser definido:

Definição 2.19 (Interpretação de Herbrand) *Seja* $\mathcal{L}$ *uma linguagem de primeira ordem.* $I_{\mathcal{H}}$ *é uma interpretação de Herbrand se está da acordo com as seguintes condições:*

- *O domínio de* $I_{\mathcal{H}}$ *é o universo de Herbrand* $\mathcal{H}_U$ *para* $\mathcal{L}$*;*
- *Constantes em* $\mathcal{L}$ *são atribuídas a elas próprias em* $\mathcal{H}_U$*;*
- *Se* f *é um símbolo funcional enário em* $\mathcal{L}$*, então é atribuído a* f *um mapeamento* f_{I_H} *de* $\mathcal{H}_U^n$ *em* $\mathcal{H}_U$*, que mapeia a seqüência* $t_1, \ldots, t_n$ *de termos básicos no termo básico* $f(t_1, \ldots, t_n)$*;*
- *Se* p *é um símbolo predicado enário de* $\mathcal{L}$*, então é atribuído a* p *um mapeamento* p_{I_H} *de* $\mathcal{H}_U^n$ *em* $\{1, 0\}$*.*

Definição 2.20 (Modelo de Herbrand) *Sejam* $\mathcal{L}$ *uma linguagem de primeira ordem e* Γ *um conjunto de cláusulas. Um modelo de Herbrand para* Γ *é uma interpretação de Herbrand para* $\mathcal{L}$ *que é um modelo para* Γ*.*

Observe que uma interpretação de Herbrand não é somente uma interpretação com domínio restrito ao $\mathcal{H}_U$. Em adição, as constantes e os símbolos funcionais têm interpretações fixas: símbolos constantes são atribuídos a si mesmos e símbolos funcionais são interpretados como construtores de termos sobre $\mathcal{H}_B$. Com isso, toda interpretação de Herbrand para um conjunto de cláusulas Γ possui o mesmo domínio e a mesma valoração

para constantes e símbolos funcionais. As eventuais diferenças entre essas interpretações são restritas, portanto, à valoração dos símbolos predicados. Assim, uma interpretação de Herbrand $I_{\mathcal{H}}$ pode ser alternativamente definida da seguinte maneira:

$$I_{\mathcal{H}} = \{a_1, \ldots, a_n, \ldots\}$$

em que a_j é um átomo (A_j) ou a negação de um átomo ($\neg A_j$), em que $A_j \in \mathcal{H}_B$ para $j = 1, 2, \ldots$. O significado de $I_{\mathcal{H}}$ é que se a_j é A_j, então A_j é verdadeiro em $I_{\mathcal{H}}$; caso contrário, A_j é falso em $I_{\mathcal{H}}$.

Exemplo 2.21 Seja $\Gamma = \{p(x) \vee q(x), r(f(y))\}$. O universo de Herbrand de Γ é $\mathcal{H}_U = \{a, f(a), f(f(a)), \ldots\}$ e a base de Herbrand $\mathcal{H}_B$ é igual a

$$\mathcal{H}_B = \{p(a), q(a), r(a), p(f(a)), q(f(a)), r(f(a)), \ldots\}$$

Algumas interpretações de Herbrand podem então ser exibidas:

$$\begin{aligned} I_1 &= \{p(a), q(a), r(a), p(f(a)), q(f(a)), r(f(a)) \ldots\} \\ I_2 &= \{\neg p(a), \neg q(a), \neg r(a), \neg p(f(a)), \neg q(f(a)), \neg r(f(a)) \ldots\} \\ I_3 &= \{p(a), q(a), \neg r(a), p(f(a)), q(f(a)), \neg r(f(a)) \ldots\} \end{aligned}$$

As interpretações de Herbrand possuem duas propriedades fundamentais com conseqüências importantes. A primeira delas indica que é necessário considerar somente as interpretações de Herbrand para constatar se um conjunto de cláusulas é insatisfatível:

Teorema 2.22 *[15] Um conjunto de cláusulas Γ é insatisfatível se e somente se Γ é insatisfatível sob todas as interpretações de Herbrand para Γ.*

A segunda propriedade garante que é possível reduzir o problema de testar a insatisfatibilidade de um conjunto de cláusulas Γ ao problema de testar a existência de um conjunto finito e insatisfatível de fórmulas da lógica sentencial gerado de forma sistemática a partir de Γ.

Teorema 2.23 (Teorema de Herbrand) *[39] Um conjunto Γ de cláusulas é insatisfatível sss há um conjunto insatisfatível finito Γ' de instâncias básicas (sem variáveis) de cláusulas de Γ.*

Exemplo 2.24 Seja $\Gamma = \{p(x), \neg p(f(a))\}$. Esse conjunto possui uma instância insatisfatível: $p(f(a)), \neg p(f(a))$. Pelo teorema de Herbrand, Γ é também insatisfatível.

Doravante, ao se referir a interpretações ou modelos, está-se referindo respectivamente a interpretações ou modelos de Herbrand. Por simplicidade, $I_{\mathcal{H}}$ será indistintamente denotada por I.

2.1.4 Forma padrão de Skolem

Convém salientar que os teoremas 2.22 e 2.23 são somente aplicáveis a cláusulas. No entanto, qualquer fórmula pode ser transformada em um conjunto de cláusulas. Esse processo é executado em duas etapas: inicialmente, uma fórmula F é transformada em uma fórmula G na forma normal conjuntiva prenex. Como pode ser facilmente verificado, isso sempre é possível e além disso, F e G são equivalentes. Na segunda etapa, os eventuais quantificadores existenciais de uma fórmula na forma conjuntiva normal prenex são eliminados através do método da skolemização, que foi introduzido por Davis e Putnam em [23].

Definição 2.25 *Seja γ uma fórmula na forma normal prenex $(Q_1x_1)\ldots(Q_nx_n)\beta$, em que β está na forma normal conjuntiva. Seja Q_j um quantificador existencial, com $1 \leq j \leq n$, ocorrendo em γ.*

- *Se não há quantificador universal que apareça antes de Q_j em γ, toda ocorrência de x_j em β é substituída por uma constante c_j que não aparece em γ e Q_j é retirado de γ.*
- *Senão, se a seqüência $Q_{s_1}\ldots Q_{s_m}$ é constituída de quantificadores universais que aparecem antes de Q_j, tal que $1 \leq s_1 < \ldots < s_m < j$, toda ocorrência de x_j em β é substituída por $f_j(x_{s_1}\ldots x_{s_m})$, em que f_j é um símbolo funcional que não aparece em γ, e Q_j é retirado de γ.*

Esse processo, chamado de skolemização, continua até não haver mais quantificadores existenciais em γ. A última fórmula obtida está na forma padrão de Skolem (ou simplesmente forma padrão).

Exemplo 2.26 Seja γ a fórmula

$$(\exists x)(\forall y)(\exists z)(\forall v)(\exists w)P(x,y,z,v,w).$$

Como $\exists x$ não é precedido por nenhum quantificador universal, a variável x é substituída por uma constante c e $\exists x$ é eliminado de γ; $\exists z$, por sua vez, é precedida por $\forall y$, portanto, a variável z é trocada por uma função $f(y)$ e $\exists z$ é eliminado de γ; finalmente, $\exists w$ é precedido por $(\forall y)$ e por $(\forall v)$, assim, a variável w é trocada por $g(y,v)$. A fórmula resultante está na forma padrão e é mostrada abaixo:

$$(\forall y)(\forall v)P(c,y,f(y),v,g(y,v)).$$

Exemplo 2.27 Seja $\gamma = (\exists x)p(x)$. A sua forma padrão pode ser escrita como $p(a)$. Além disso seja I uma interpretação tal que

Domínio $D = \{1, 2\}$

$I(a) = 1$

$I(p(1)) = 0$ e $I(p(2)) = 1$

Disso resulta que γ é verdadeiro em I, mas $p(a)$ é falso, logo γ não é equivalente a $p(a)$. Felizmente, a skolemização preserva a propriedade da inconsistência:

Teorema 2.28 *[39] Seja Γ um conjunto de cláusulas que representa uma fórmula F na forma padrão. Então F é insatisfatível se e somente se Γ é insatisfatível.*

Como qualquer variável numa fórmula padrão F está no escopo de algum quantificador universal, F pode ser visto como um conjunto Γ de cláusulas. Assim, pelo teorema 2.28 e pelo teorema de Herbrand, o problema de demonstrar que F é insatisfatível restringe-se a encontrar um conjunto insatisfatível finito Γ' de instâncias básicas de cláusulas de Γ.

2.2 Teoria da prova

Muitas tentativas foram feitas visando estabelecer métodos capazes de averiguar a relação de conseqüência lógica de um modo puramente mecânico. Para tanto, é preciso constituir uma demonstração de que uma conclusão segue das premissas. Essa demonstração é especificada por uma teoria.

Definição 2.29 (Teoria) *Na lógica de primeira ordem, uma teoria $\mathcal{T}$ é constituída de*

- *Um alfabeto $\mathcal{A}$;*
- *Uma linguagem de primeira ordem $\mathcal{L}$ sobre $\mathcal{A}$;*
- *Um subconjunto de fórmulas da linguagem $\mathcal{L}$, chamadas de axiomas;*
- *Um conjunto finito $R_1, \ldots, R_n$ de relações entre fórmulas chamadas de regras de inferência. Cada R_i tem a forma genérica $\frac{\mathcal{F}_1 \ldots \mathcal{F}_n}{\mathcal{F}}$, em que $\mathcal{F}_1 \ldots \mathcal{F}_n$ e $\mathcal{F}$ são representações para fórmulas arbitrárias. Uma aplicação da regra é possível se $\mathcal{F}_1, \ldots, \mathcal{F}_n$ são dadas ou são obtidas de aplicações prévias de alguma regra de inferência da teoria.*

Definição 2.30 (Prova) *Sejam γ uma fórmula, Γ um conjunto de fórmulas e $\mathcal{T}$ uma teoria da lógica da primeira ordem. Uma prova ou dedução de γ a partir de Γ em $\mathcal{T}$ é uma seqüência finita $\gamma_1, \ldots, \gamma_n$ de fórmulas tal que $\gamma_n = \gamma$ e para cada i, γ_i é um axioma de $\mathcal{T}$, é um elemento de Γ ou é uma conseqüência direta de algumas das fórmulas precedentes da prova em virtude da aplicação de uma das regras de inferência.*

Numa prova, é estabelecida uma nova relação, denotada por $\vdash$, que busca capturar a relação $\models$ de um modo puramente mecânico.

Definição 2.31 *Uma fórmula γ é derivada ou é uma conseqüência de um conjunto de fórmulas Γ numa teoria $\mathcal{T}$ ($\Gamma \vdash_{\mathcal{T}} \gamma$) sss há uma prova de γ a partir de Γ em $\mathcal{T}$. Os elementos de Γ e a fórmula γ são respectivamente chamados de premissas e conclusão da prova. Em particular, se $\Gamma = \emptyset$, γ é dito ser um teorema de $\mathcal{T}$ ($\vdash_{\mathcal{T}} \gamma$).*

Quando isto não causar ambigüidades, $\Gamma \vdash_{\mathcal{T}} \gamma$, será simplesmente denotado por $\Gamma \vdash \gamma$. Às noções de satisfatibilidade e de insatisfatibilidade definidas em $\models$, existem respectivamente as de consistência e inconsistência definidas em $\vdash$.

Definição 2.32 *Um conjunto de sentenças Γ é inconsistente sss $\Gamma \vdash \gamma \wedge \neg\gamma$ para alguma fórmula γ pertencente a linguagem de Γ; caso contrário, Γ é consistente.*

Quando é encontrada uma dedução de $\gamma \wedge \neg\gamma$ a partir de Γ, diz-se que existe uma *refutação* em Γ. Para simplificar o processo de dedução numa teoria, algumas restrições são impostas sobre a aplicação das regras de inferência e sobre a seqüência de sentenças pertencentes a uma dedução. Nesse sentido, é definida a noção de *método de dedução* como consistindo de um par $\mathcal{M}\langle\mathcal{T}, \mathcal{D}\rangle$, em que $\mathcal{T}$ é uma teoria e $\mathcal{D}$ é um conjunto de deduções em $\mathcal{T}$, chamada de deduções admissíveis em $\mathcal{M}$.

Por sua vez, dado um método de dedução $\mathcal{M}$ e um conjunto de sentenças Γ, um *espaço de busca* de Γ é o conjunto de todas as deduções a partir de Γ admissíveis em $\mathcal{M}$. Um método de dedução $\mathcal{M}$ é um *refinamento* de um método de dedução $\mathcal{M}'$ se e somente se

- As teorias de ambos métodos admitem a mesma classe de linguagens;
- Para todo conjunto Γ de sentenças, o espaço de busca de Γ em $\mathcal{M}$ está contido no espaço de busca de Γ em $\mathcal{M}'$;
- Existe pelo menos um conjunto Γ de sentenças tal que o espaço de busca de Γ em $\mathcal{M}$ não é igual ao espaço de busca de Γ em $\mathcal{M}'$.

Um *procedimento de dedução* baseado num método $\mathcal{M}$ é um algoritmo que recebe como entrada um conjunto Γ de sentenças e uma sentença γ da linguagem de $\mathcal{T}$ e gera sistematicamente todas as deduções no espaço de busca de Γ em $\mathcal{M}$ até encontrar uma dedução de γ a partir de Γ. Dependendo do tipo de procedimento de dedução, uma teoria pode ser classificada quanto a decidibilidade e a tratabilidade.

Definição 2.33 *Uma teoria $\mathcal{T}$ é decidível sss para qualquer fórmula γ pertencente a linguagem de $\mathcal{T}$, existe um procedimento que determine em tempo finito a existência de*

uma prova de γ em $\mathcal{T}$; caso contrário, $\mathcal{T}$ é indecidível. Por sua vez, $\mathcal{T}$ é tratável se existe um procedimento de prova definido por um algoritmo de complexidade polinomial; caso contrário, $\mathcal{T}$ é intratável.

Um procedimento de dedução para determinar $\Gamma \vdash \gamma$ é o meio mecânico para demonstrar $\Gamma \models \gamma$. Uma teoria é *correta* se $(\Gamma \vdash \gamma \Rightarrow \Gamma \models \gamma)$ e é completa se $\Gamma \models \gamma \Rightarrow \Gamma \vdash \gamma$ para qualquer fórmula γ da linguagem de $\mathcal{T}$.

Além disso, como pode ser verificado em [15, 22], $\Gamma \models \gamma$ é equivalente às seguintes declarações:

1. $\Gamma \Rightarrow \gamma$ é válida;

2. $\Gamma \wedge \neg\gamma$ é insatisfatível.

Assim, um procedimento de dedução pode ser de *validação* se for baseado na condição 1 ou de *refutação* se for baseado na condição 2. Um método $\mathcal{M}$ é *refutacionalmente correto* se e somente se, para todo conjunto de sentenças Γ, se existe uma refutação no espaço de busca de Γ em $\mathcal{M}$, então Γ é insatisfatível; $\mathcal{M}$ é *refutacionalmente completo* se e somente se, para todo conjunto de sentenças Γ se Γ é insatisfatível, então existe uma refutação no espaço de busca de Γ em $\mathcal{M}$. Observe que se $\mathcal{T}$ é correto, então $\mathcal{M}$ é refutacionalmente correto; contudo se $\mathcal{T}$ é completo, nem sempre $\mathcal{M}$ é refutacionalmente completo, pois uma dedução em $\mathcal{T}$ pode não ser admissível em $\mathcal{M}$. Seguindo essa linha de raciocínio, se $\mathcal{M}$ é um refinamento de $\mathcal{M}'$ e $\mathcal{M}'$ é refutacionalmente correto, então $\mathcal{M}$ também é refutacionalmente correto. Em contrapartida, $\mathcal{M}'$ pode ser refutacionalmente completo e $\mathcal{M}$ ser refutacionalmente incompleto.

2.2.1 Teoria da resolução

A primeira tentativa de encontrar um procedimento de prova geral foi feita por Leibniz com o cálculo raciocinador, que deveria solucionar problemas arbitrários através de uma computação puramente mecânica. Posteriormente, esse trabalho foi revisto por Peano na virada do século e pela escola de Hilbert nos anos 20. No entanto, em trabalhos independentes, Church [16] e Turing [90] demonstraram que na lógica de primeira ordem não há, em termos gerais, procedimentos de prova para averiguar a validade/insatisfatibilidade de fórmulas. Isso não ocasionou, todavia, na rejeição por inteiro dessa idéia e alguns procedimentos de prova que verificam a validade de fórmulas foram desenvolvidos. Para fórmulas válidas, esses procedimentos sempre param e encontram a resposta correta. No entanto, para fórmulas inválidas, pode ocorrer de nenhuma resposta ser apresentada em tempo finito. Face aos resultados de Church e Turing, esse é o melhor desempenho que se pode esperar de um procedimento de prova geral para a lógica de primeira ordem.

Os resultados obtidos por Herbrand permitiram reduzir o problema de determinar a insatisfatibilidade de uma fórmula ao problema de determinar a insatisfatibilidade para um conjunto finito de instâncias básicas de um conjunto de cláusulas Γ. Baseado nisso, Herbrand elaborou um procedimento de refutação que gera conjuntos de instâncias básicas de cláusulas de Γ até que um desses conjuntos seja insatisfatível. Pelo teorema de Herbrand, é garantido que sendo Γ insatisfatível, esse procedimento sempre encontra uma solução em um número finito de tentativas. Entretanto, se Γ é satisfatível e contém símbolos funcionais, esse procedimento não pára de executar. Isso ocorre porque essa teoria aplicada a cláusulas arbitrárias é indecidível, portanto não se pode esperar que um procedimento de prova sempre termine.

Em [31], Gilmore foi um dos primeiros a implementar a idéia acima de Herbrand. No entanto, o seu método de testar a insatisfatibilidade das fórmulas é bastante ineficiente. Em [23], Davis e Putnam implementaram algumas melhorias no método de Gilmore, porém a geração de instâncias básicas na maioria das vezes cresce exponencialmente, inviabilizando inclusive a prova de cláusulas simples num computador.

Substituição e unificação

Para dirimir o problema acima, Robinson introduziu o princípio da resolução [83]. Esse princípio é aplicado a um conjunto de cláusulas Γ para testar a sua insatisfatibilidade. Isso é alcançado ao ser demonstrada que a cláusula vazia $\Box$ segue de Γ. Antes de definir o princípio da resolução, será apresentado um método que permite encontrar um par de literais complementares $(A, \neg A)$ pertencentes a cláusulas de Γ. Em se tratando de literais básicos, essa tarefa é elementar; porém para cláusulas contendo variáveis, o processo torna-se mais complexo.

Exemplo 2.34 Sejam $\gamma_1 = p(x) \vee r(x)$ e $\gamma_2 = \neg p(f(x)) \vee q(x)$. Não há nenhum literal em γ_1 complementar a γ_2. Entretanto, substituindo em γ_1, a varíavel x por $f(a)$ e em γ_2, a variável x pelo termo a, obtém-se o par de literais complementares $p(f(a)), \neg p(f(a))$. Uma substituição é definida da seguinte maneira:

Definição 2.35 *Uma substituição θ é um conjunto finito da forma $\{v_1/t_1, \ldots, v_n/t_n\}$, em que todo v_i é uma variável e todo t_i é um termo diferente de v_i e as variáveis $v_1 \ldots v_n$ são distintas entre si. Quando $t_1 \ldots, t_n$ são termos básicos, a substituição é chamada de substituição básica. A substituição dada pelo conjunto vazio recebe o nome de substituição identidade e é denotada por ε.*

Na seqüência, uma *expressão* é um termo, um literal, uma conjunção ou uma disjunção de literais. Uma *expressão simples* é um termo ou um átomo.

Definição 2.36 *Sejam E e F expressões. E e F são variantes se existe uma substituição θ e σ tal que $E = F\theta$ e $F = E\sigma$. E é chamado de variante de F e F é chamado de variante de E.*

Definição 2.37 *Uma substituição θ é uma renomeação de variáveis ou simplesmente renomeação se e somente se cada elemento x/y de θ for tal que y é uma variável e não existirem dois elementos x/y e z/w em θ tais que $x \neq z$ e $y = w$.*

Definição 2.38 *Sejam $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ uma substituição e E uma expressão. Assim, $E\theta$ é uma expressão obtida de E trocando simultaneamente cada ocorrência da variável v_i, $1 \leq i \leq n$, em E por um termo t_i. $E\theta$ é chamada de uma instância de E. Se $E\theta$ é básico, então $E\theta$ é chamado de uma instância básica de E.*

Exemplo 2.39 Sejam $\theta = \{x/a, y/f(b), z/c\}$ e $E = \{p(x, y, z)\}$. Então $E\theta = P(a, f(b), c)$.

Definição 2.40 *Sejam $\theta = \{u_1/s_1, \ldots, u_m/s_n\}$ e $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ substituições. Então a composição $\theta\sigma$ de θ e σ é a substituição obtida do conjunto*

$$\{u_1/s_1\sigma, \ldots, u_m/s_n\sigma, v_1/t_1, \ldots, v_n/t_n\}\}$$

retirando todo $u_i/s_i\sigma$ para o qual $u_i = s_i\sigma$ e retirando todo v_j/t_j para o qual $v_j \in \{u_1, \ldots, u_m\}$.

Exemplo 2.41 Sejam $\theta = \{x/y, z/f(x)\}$ e $\sigma = \{x/b, y/x, z/a\}$. Então, tem-se que $\theta\sigma = \{z/f(b), y/x\}$.

Uma substituição especial que torna todos os elementos de um conjunto sintaticamente idênticos é chamada de unificador. Tal substituição é utilizada para reduzir o espaço de busca por um par de literais complementares e foi resgatada por Robinson em [83] a partir de um trabalho de Herbrand [39].

Definição 2.42 *Uma substituição θ é chamada de unificador para um conjunto de expressões $\{E_1, \ldots, E_k\}$ se e somente se $E_1\theta = E_2\theta = \ldots = E_k\theta$. O conjunto $\{E_1, \ldots, E_k\}$ é unificável se há um unificador para ele.*

Definição 2.43 *Um unificador σ para um conjunto de expressões $\{E_1, \ldots, E_k\}$ é um unificador mais geral (mgu)[2] se e somente se para cada unificador θ para esse conjunto, há uma substituição λ tal que $\theta = \sigma\lambda$.*

[2] *Do inglês Most General Unifier.*

Exemplo 2.44 O conjunto $\{p(f(x), z), p(y, a)\}$ é unificável desde que a substituição $\theta = \{y/f(a), x/a, z/a\}$ é um unificador para esse conjunto. O unificador mais geral é $\sigma = \{y/f(x), z/a\}$ (Observe que $\sigma = \theta\{x/a\}$).

Para encontrar o *mgu*, é apresentado o algoritmo da unificação, que tem como entrada de dados um conjunto Γ de expressões simples e como saída o *mgu* dessas expressões se o conjunto for unificável; caso contrário, é relatado que esse conjunto não é unificável. Nesse algoritmo, inicialmente, é localizada a posição do símbolo mais à esquerda tal que nem todos os elementos de Γ sejam sintaticamente idênticos. Em seguida, uma tentativa é feita de unificar todas as subexpressões de Γ que começam na posição desse símbolo. Se essa tentativa é bem sucedida, o processo continua com a expressão resultante da substituição; senão Γ não é unificável e o algoritmo pára. Se o final das expressões de Γ é alcançado, o procedimento pára e a composição de todas as substituições feitas em Γ é o *mgu* de Γ.

Definição 2.45 *O conjunto da discórdia*[3] *de um conjunto não vazio* Γ *de expressões simples é obtido localizando a posição do símbolo mais à esquerda que nem todas as expressões em* Γ *tenham exatamente esse mesmo símbolo. Em seguida, de cada expressão de* Γ *é extraída a subexpressão que começa com o símbolo ocupando aquela posição. O conjunto dessas respectivas subexpressões é o conjunto da discórdia.*

Exemplo 2.46 Seja $\Gamma = \{p(x, f(y, z)), p(x, a), p(x, g(h(k(x))))\}$, então a primeira posição do símbolo que nem todos as expressões de Γ são exatamente as mesmas é a quinta. Desse modo, o conjunto da discórdia de Γ é igual a $\{f(y, z), a, g(h(k(x)))\}$.

O algoritmo da unificação é apresentado abaixo. Neste algoritmo, Γ denota um conjunto de expressões simples:

Algoritmo da Unificação

Passo 1 - Faça $k = 0$ e $\sigma_k = \varepsilon$.

Passo 2 - Se $\Gamma\sigma_k$ é um conjunto unitário, pare; σ_k é o *mgu* para Γ. Caso contrário, encontre o conjunto da discórdia D_k de $\Gamma\sigma_k$.

Passo 3 - Se há elementos v e t em D_k tal que v é uma variável que não ocorre em t, faça $\sigma_{k+1} = \sigma_k\{v/t\}$, incremente k e vá ao passo 2. Caso contrário, pare; Γ não é unificável.

O algoritmo da unificação como foi apresentado é não determinístico, pois pode haver várias escolhas para v_k e t_k no passo 3. No entanto, é garantido que esse algoritmo sempre termina já que Γ contém finitamente muitas variáveis e em cada aplicação do passo 3, uma delas é eliminada.

[3] *Do inglês disagreement set.*

Exemplo 2.47 [50] Seja $\Gamma = \{p(a, x, h(g(z))), p(z, h(y), h(y))\}$.

$$\begin{aligned}
\sigma_0 &= \varepsilon \\
D_0 &= \{a, z\}, \sigma_1\{z/a\} \text{ e } \Gamma\sigma_1 = \{p(a, x, h(g(a)), p(a, h(y), h(y))\} \\
D_1 &= \{x, h(y)\}, \sigma_2 = \{z/a, x/h(y)\} \text{ e } \Gamma\sigma_2 = \{p(a, h(y), h(g(a))), p(a, h(y), h(y))\} \\
D_2 &= \{y, g(a)\}, \sigma_3 = \{z/a, x/h(g(a)), y/g(a)\} \text{ e } \Gamma\sigma_3 = \{p(a, h(g(a)), h(g(a)))\}
\end{aligned}$$

Assim, Γ é unificável e σ_3 é o seu *mgu*.

Princípio da resolução

Tendo introduzido o algoritmo da unificação, pode-se agora considerar o princípio da resolução, apresentado por Robinson em [83], que é definido como uma regra de inferência que gera uma nova cláusula a partir de duas cláusulas.

Definição 2.48 *Se dois ou mais literais de uma cláusula C têm um unificador mais geral σ, então $C\sigma$ é chamado de fator de C.*

Exemplo 2.49 Seja $C = \{p(x) \vee p(f(y)) \vee \neg q(x)\}$. Os literais $p(x)$ e $p(f(y))$ possuem o *mgu* $\sigma = \{x/f(y)\}$. Conseqüentemente, $C\sigma = \{p(f(y)) \vee \neg q(f(y))\}$ é um fator de C.

Na definição a seguir, γ_1 e γ_2 são duas cláusulas. Se elas possuem variáveis em comum, essas variáveis são renomeadas de modo a não mais isso acontecer. Em um abuso notacional, os literais resultantes dessa renomeação também serão denotados por γ_1 e γ_2.

Definição 2.50 *Sejam $\gamma_1 = L_1 \vee \ldots \vee L_n$ e $\gamma_2 = M_1 \vee \ldots \vee M_m$ duas cláusulas (chamadas de cláusulas pai) sem variáveis em comum, em que todo L_i, $1 \leq i \leq n$ e todo M_j, $1 \leq j \leq m$, são literais. Sejam L_r, $1 \leq r \leq n$ e M_s, $1 \leq s \leq m$, dois literais em γ_1 e γ_2 respectivamente. Se L_r e $\neg M_s$ possuem um unificador mais geral θ, então a cláusula*

$$(L_1 \vee \ldots \vee L_{r-1} \vee L_{r+1} \vee \ldots \vee L_n \vee M_1 \vee \ldots \vee M_{s-1} \vee M_{s+1} \vee M_m)\theta$$

é um resolvente binário de γ_1 e γ_2. Nesse caso, diz-se também que γ_1 resolve-se com γ_2 (e vice-versa). Os literais L_r e M_s são chamados de literais resolvidos.

A renomeação das cláusulas assegura que o fato de γ_1 e γ_2 possuírem variáveis em comum não irá impedir a derivação de novas cláusulas. Por exemplo, seja $\gamma_1 = p(x)$ e $\gamma_2 = p(f(x))$. Observe que não há resolventes binários de γ_1 e γ_2, pois o conjunto $\Gamma = p(x), p(f(x))$ não é unificável. Entretanto, $\square$ é um resolvente binário de $\gamma_1 = p(x)$ e $\gamma_2\theta = p(f(y))$, em que $\theta = (x/y)$ é uma renomeação em γ_2.

Definição 2.51 *Um resolvente das cláusulas pai γ_1 e γ_1 é um dos seguintes resolventes binários:*

1. *Um resolvente binário de γ_1 e γ_2;*
2. *Um resolvente binário de γ_1 e um fator de γ_2;*
3. *Um resolvente binário de um fator de γ_1 e γ_2;*
4. *Um resolvente binário de um fator de γ_1 e de um fator de γ_2.*

O *principio da resolução* ou simplesmente resolução é uma regra de inferência que gera resolventes de um conjunto de cláusulas. Essa regra é consideravelmente mais eficiente do que os procedimentos de refutação baseados no teorema de Herbrand tais como os apresentados por Gilmore e por Davis e Putnam. Devido a sua simplicidade e eficiência, o princípio da resolução exerce um papel de destaque, sendo que algumas de suas características como a unificação são comuns a todos os procedimentos de prova modernos.

Exemplo 2.52 Sejam $\gamma_1 = p(x) \vee p(f(y)) \vee r(g(y))$ e $\gamma_2 = \neg p(f(g(a))) \vee q(x)$. Como x aparece em γ_1 e γ_2, x é renomeado em γ_2. Seja $\gamma_2 = \neg p(f(g(a))) \vee q(z)$. Um fator de γ_1 é $\gamma_1' = p(f(y)) \vee r(g(y))$. Um resolvente binário de γ_1' e γ_2 é $r(g(g(a))) \vee q(z)$. Portanto, $r(g(g(a))) \vee q(z)$ é um resolvente de γ_1 e γ_2.

De posse dessas noções, uma teoria da resolução $\mathcal{T}_R$ pode ser enunciada da seguinte forma:

Definição 2.53 (Teoria da resolução) *Uma teoria da resolução $\mathcal{T}_R$ consiste de um alfabeto de primeira ordem $\mathcal{A}$, uma linguagem constituída de cláusulas sobre $\mathcal{A}$, nenhum axioma e como única regra de inferência o princípio da resolução.*

A noção de dedução em $\mathcal{T}_R$ é formalizada seguindo a noção de dedução apresentada na seção 2.2:

Definição 2.54 *Sejam Γ um conjunto de cláusulas e γ uma cláusula. Uma dedução de γ a partir de Γ em $\mathcal{T}_R$ é uma seqüência $S = \{S_1, \ldots, S_n\}$ de cláusulas tal que*

1. *$S_n = \gamma$;*
2. *Para todo i, $1 \leq i \leq n$, $S_i \in \Gamma$ ou S_i é um resolvente de S_j e S_k para algum $j, k < i$.*

Definição 2.55 *Uma refutação a partir de Γ em $\mathcal{T}_R$ é uma dedução de $\square$ a partir de Γ.*

Exemplo 2.56 Seja $\Gamma = \{p(x) \vee q(x), \neg p(x) \vee q(f(y)), p(x) \vee \neg q(f(x)), \neg p(x) \vee \neg q(x)\}$ um conjunto de cláusulas. A seqüência abaixo é uma refutação a partir dessas cláusulas em $\mathcal{T}_R$:

(1)	$\neg p(x) \vee \neg q(x)$	(Cláusula de Γ)
(2)	$\neg p(z) \vee q(f(y))$	(Cláusula renomeada de Γ)
(3)	$\neg p(z) \vee \neg p(f(y))$	(Resolução de (1) e (2))
(4)	$p(x) \vee q(x)$	(Cláusula de Γ)
(5)	$q(f(y))$	(Resolução de (3) e (4))
(6)	$p(x) \vee \neg q(f(x))$	(Cláusula de Γ)
(7)	$p(x)$	(Resolução de (5) e (6))
(8)	$\neg p(u) \vee \neg q(u)$	(Cláusula renomeada de Γ)
(9)	$\neg q(u)$	(Resolução de (7) e (8))
(10)	$\square$	(Resolução de (5) e (9))

Os próximos resultados garantem a correção e a completude da teoria da resolução:

Teorema 2.57 (Correção da resolução) *[83]Para todo conjunto Γ de cláusulas, se existe uma refutação a partir de Γ em $\mathcal{T}_R$, então Γ é insatisfatível.*

Teorema 2.58 (Completude da resolução) *[83]Para todo conjunto Γ de cláusulas, se Γ é insatisfatível, então existe uma refutação a partir de Γ em $\mathcal{T}_R$.*

2.2.2 Método da resolução linear

Apesar de sua eficiência, no princípio da resolução, o espaço de busca pode crescer exponecialmente em alguns casos, tornando-se um sério obstáculo para muitas aplicações práticas. Essa explosão combinatorial é provocada pela liberdade de escolha de cláusulas, literais e fatores no princípio da resolução. Para evitar isso, foram desenvolvidos meios de limitar o espaço de busca. Um desses refinamentos é o método da resolução linear, que foi independentemente proposto por Loveland [52] e Luckham [53] em 1970.

Resolução linear permite somente deduções em que cada cláusula é obtida resolvendo-se sempre a cláusula imediatamente anterior com uma cláusula de entrada ou uma cláusula deduzida anteriormente. Isso estabelece um seqüenciamento linear de resoluções em vez das várias ramificações admissíveis no princípio da resolução sem restrições. Nos conceitos a seguir, é assumido que todas as deduções estão na teoria da resolução $\mathcal{T}_R$.

Definição 2.59 (Resolução linear) *Sejam Γ um conjunto de cláusulas e C_0 uma cláusula de Γ. Uma dedução linear de C_n de Γ com cláusula inicial C_0 é obtido do modo mostrado na figura 2.1 em que:*

1. Para $i = 0, 1, \ldots, n-1$, C_{i+1} é um resolvente de C_i (chamado de *cláusula central*) e B_i (chamada de cláusula lateral);

2. Por sua vez, cada B_i está em Γ ou está em C_j, para algum $j < i$.

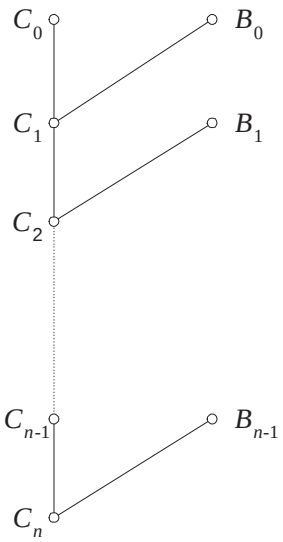

Figura 2.1: Resolução Linear

Definição 2.60 *Uma refutação linear de C_0 de Γ é uma dedução linear de $\square$ a partir de Γ.*

Exemplo 2.61 Seja $\Gamma = \{\neg p(x) \vee q(x), p(x) \vee q(x), p(x) \vee \neg q(x), \neg p(x) \vee \neg q(x)\}$. Por simplicidade, neste capítulo, serão utilizados símbolos proposicionais para representar literais quando isso não causar ambigüidades. Assim, $p(x), q(x)$ serão tratados respectivamente por p e q. Escolhendo $\neg p \vee q$ como cláusula inicial, a figura 2.2 representa uma refutação linear da cláusula $\neg p \vee q$ de Γ.

Uma escolha errônea da cláusula inicial C_0 de um conjunto de cláusulas Γ pode conduzir à incompletude, ou seja, Γ é inconsistente, mas $\square$ não é obtido de C_0 e Γ.

Exemplo 2.62 Seja $\Gamma = \{p(x), \neg p(y) \vee r(y), \neg r(u), q(a)\}$. Γ é insatisfatível, mas selecionando $q(a)$ como cláusula inicial, a única dedução linear de Γ e $q(a)$ é o próprio $q(a)$. Desse modo, não existe uma refutação linear de Γ com cláusula inicial $q(a)$. Entretanto, se quaisquer das demais cláusulas de Γ forem escolhidas, $\square$ será deduzido.

Para estabelecer um método de resolução linear completo, as cláusulas iniciais devem ser restritas às relevantes:

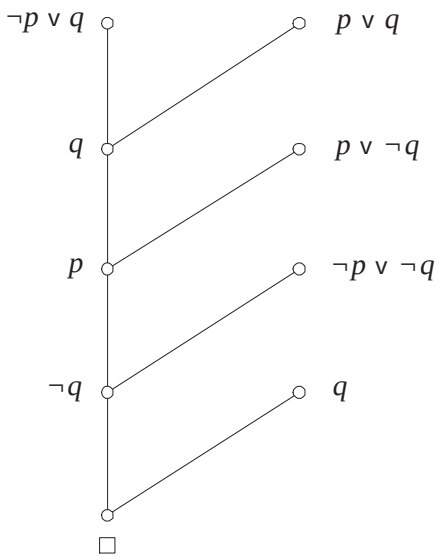

Figura 2.2: Exemplo de Refutação Linear

Definição 2.63 *Seja* Γ *um conjunto de cláusulas. Uma cláusula* $C \in \Gamma$ *é chamada de relevante (como cláusula inicial de uma dedução linear) se há um subconjunto* $\Gamma' \subseteq \Gamma$ *tal que* Γ' *é satisfatível, mas* $\Gamma' \cup \{C\}$ *é insatisfatível.*

No exemplo 2.62, excetuando $q(a)$, todas as demais cláusulas são relevantes.

Por ser um refinamento de resolução, esse método herda a sua característica de ser refutacionalmente correto. O teorema abaixo mostra que o método da resolução linear é também refutacionalmente completo:

Definição 2.64 (Completude da resolução linear) *Sejam* Γ *um conjunto insatisfatível de cláusulas e* C *uma cláusula relevante em* Γ*. Então existe uma refutação linear de* Γ *com cláusula inicial* C*.*

2.2.3 Procedimentos de refutação linear

Um procedimento de refutação linear pode ser definido como um algoritmo para construir um conjunto de árvores especiais chamadas de árvores de refutação [84, 44].

Definição 2.65 (Árvore de refutação) *Sejam* Γ *um conjunto de cláusulas e* C_0 *uma cláusula de* Γ*. Uma árvore de refutação* $\mathcal{A}_R$ *para* Γ *com cláusula inicial* C_0 *é uma árvore que satisfaz as seguintes condições:*

1. C_0 *é o nó raiz de* $\mathcal{A}_R$*;*

2. *Sejam C um nó da árvore e $\mathcal{N}$ o conjunto de seus nós ancestrais em $\mathcal{A}_R$. Cada resolvente C' de C (Cláusula pai) e $\mathcal{N} \cup \Gamma$ (Cláusulas laterais) é um filho de C;*

3. *Todos os filhos de C são resolventes de C e $\mathcal{N} \cup \Gamma$;*

4. *Nós que são cláusulas vazias não têm filhos.*

Por convenção, na representação gráfica de uma árvore de refutação, as cláusulas laterais utilizadas estão dispostas ao lado da aresta correspondente.

Definição 2.66 *A floresta de refutação para um conjunto de cláusulas Γ é o conjunto de todas as árvores de refutação para Γ tendo como cláusula inicial alguma cláusula de Γ.*

Cada ramo de uma árvore de refutação para um conjunto de cláusulas Γ com cláusula inicial C_0 pode ser visto como uma dedução linear de uma folha C_n de Γ a partir de C_0. Disso resulta as seguintes definições:

Definição 2.67 *Sejam Γ um conjunto de cláusulas, C_0 uma cláusula em Γ e $\mathcal{A}_R$ uma árvore de refutação para Γ com cláusula inicial C_0.*

- *Um folha de $\mathcal{A}_\mathcal{R}$ é um nó de sucesso de $\mathcal{A}_\mathcal{R}$ se e somente se é rotulada pela cláusula vazia. Caso contrário, essa folha é um nó de fracasso;*

- *Um ramo de sucesso de $\mathcal{A}_\mathcal{R}$ é um ramo finito de $\mathcal{A}_\mathcal{R}$ terminado com um nó de sucesso. Por outro lado, esse ramo finito é chamado de ramo de fracasso se ele termina com um nó de fracasso.*

A correspondência entre ramos de uma árvore de refutação e deduções pode ser explorada para se obter o seguinte resultado:

Teorema 2.68 *Um conjunto Γ de cláusulas é insatisfatível se e somente se existe uma árvore de refutação $\mathcal{A}_\mathcal{R}$ na floresta de refutação para Γ tal que $\mathcal{A}_\mathcal{R}$ tem um ramo de sucesso.*

Desse resultado, conclui-se que a floresta de refutação para um conjunto de cláusulas Γ é uma representação compacta e organizada do espaço de busca de Γ no método da resolução linear. Então, um procedimento de refutação linear se reduz a um algoritmo que recebe como entrada um conjunto Γ de cláusulas e progressivamente constrói a floresta de refutação para Γ em busca de um nó de sucesso. Há duas estratégias fundamentais para a expansão das árvores de uma floresta: expansão em amplitude[4] e expansão em profundidade[5].

[4]Do inglês, breadth-first method.
[5]Do inglês, depth-first method.

Na expansão em amplitude, todos os resolventes $R_1, \ldots, R_m$ de uma cláusula central C_i são gerados. Se algum deles for a cláusula vazia, o procedimento pára e uma prova é encontrada. Caso contrário, para cada R_i, todos os seus resolventes são gerados e o processo continua até a cláusula vazia ser obtida ou não houver mais cláusulas laterais que possam ser resolvidas com algum R_i.

Desse modo, seja Γ um conjunto de cláusulas e C_0 uma cláusula de Γ selecionada para ser a cláusula inicial. A estratégia da expansão em amplitude é descrita como segue abaixo:

A Estratégia da Expansão em Amplitude

Passo 1 - CLIST = $\{C_0\}$;

Passo 2 - Se CLIST está vazia, o algoritmo termina sem uma prova. Caso contrário, continue;

Passo 3 - Seja C a primeira cláusula em CLIST. Retire C de CLIST;

Passo 4 - Encontre todas as cláusulas em Γ que podem ser cláusulas laterais de C. Se não existe tais cláusulas laterais, vá ao passo 2. Caso contrário, resolva C com todas essas cláusulas laterais. Sejam $R_1, \ldots, R_m$ esses resolventes;

Passo 5 - Se algum R_q é uma cláusula vazia, $1 \leq q \leq m$, o algoritmo termina com uma prova; caso contrário, continue;

Passo 6 - Coloque $R_1, \ldots, R_m$ (numa ordem arbitrária) ao final de CLIST e vá ao passo 2.

Exemplo 2.69 Seja $\Gamma = \{\neg p \vee q, p \vee q, p \vee \neg q, \neg p \vee \neg q\}$. Escolhendo $\neg p \vee q$ como cláusula inicial e aplicando a estratégia da expansão em amplitude, obtém-se uma árvore de refutação, que é parcialmente mostrada na figura 2.3.

Na figura 2.3, são mostrados dois ramos que conduzem a uma refutação. Ambos esses ramos são possíveis refutações de $\neg p \vee q$ de Γ, sendo que um deles foi mostrado na figura 2.2 do exemplo 2.61.

Na estratégia da expansão em amplitude, muitas cláusulas irrelevantes e redundantes podem ser geradas antes da cláusula vazia ser encontrada, acarretando em um custo computacional elevado.

No tocante à estratégia da expansão em profundidade, a inserção dos nós na árvore é feita de cima para baixo ao contrário da expansão em amplitude que é feita da esquerda para a direita. A expansão em profundidade é mais fácil de implementar do que a anterior, mas possui alguns problemas relacionados à inserção de nós em caminhos errados, ou seja, caminhos que não conduzem à solução.

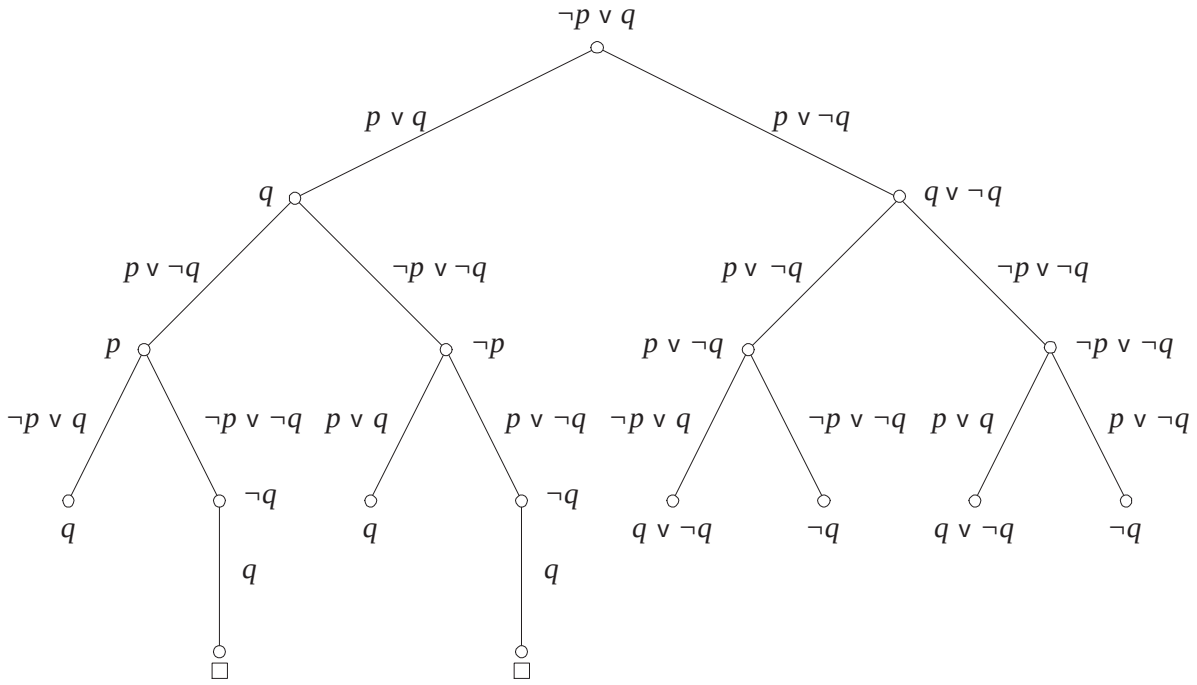

Figura 2.3: Estratégia da Expansão em Amplitude

A Estratégia da Expansão em Profundidade

Passo 1 - CLIST = $\{C_0\}$;

Passo 2 - Se CLIST está vazia, o algoritmo termina sem uma prova. Caso contrário, continue;

Passo 3 - Seja C a primeira cláusula em CLIST. Retire C de CLIST;

Passo 4 - Encontre todas as cláusulas em Γ que podem ser cláusulas laterais de C. Se não existe tais cláusulas laterais, vá ao passo 2. Caso contrário, resolva C com todas essas cláusulas laterais. Sejam $R_1, \ldots, R_m$ esses resolventes;

Passo 5 - Se algum R_q é uma cláusula vazia, $1 \leq q \leq m$, o algoritmo termina com uma prova; caso contrário continue;

Passo 6 - Coloque $R_1, \ldots, R_m$ (em uma ordem arbitrária) ao início de CLIST e vá ao passo 2.

Exemplo 2.70 Seja $\Gamma = \{\neg p \vee q, p \vee q, p \vee \neg q, \neg p \vee \neg q\}$. Escolhendo $\neg p \vee q$ como cláusula inicial e aplicando a estratégia da expansão em profundidade, obtém-se uma árvore de refutação, que é parcialmente mostrada na figura 2.4.

Na expansão em profundidade, nem sempre é garantido encontrar uma refutação caso ela exista. No entanto, essa estratégia é em geral mais simples do que a expansão em

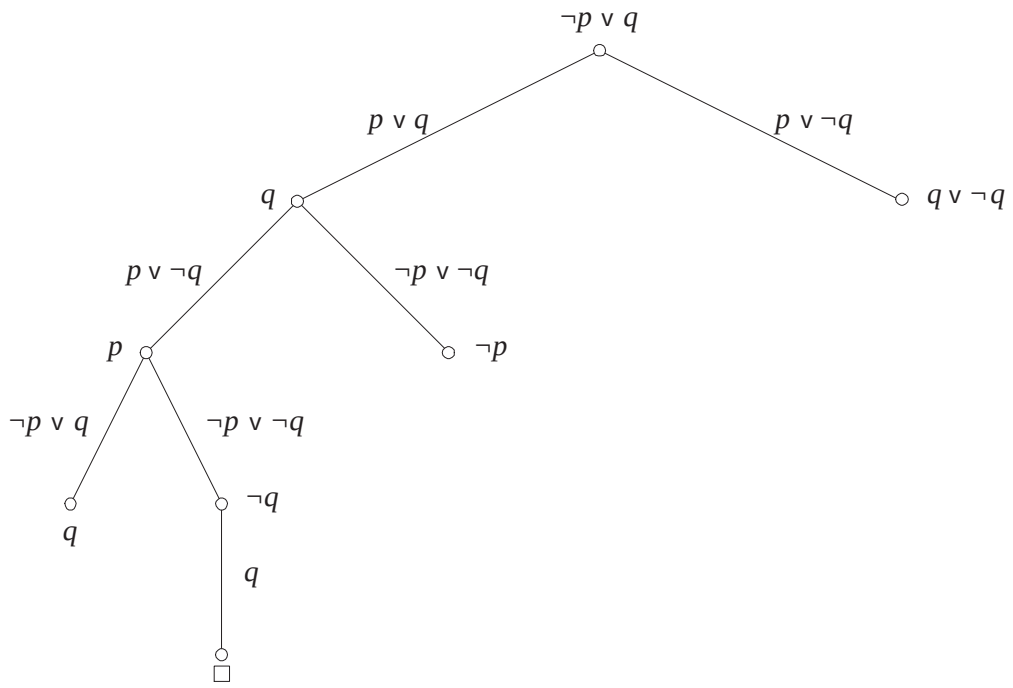

Figura 2.4: Estratégia da Expansão em Profundidade

amplitude. Por causa disso, é bastante utilizada em provadores automáticos de teorema como pode ser conferido no próximo capítulo.

2.3 Pontos fixos

Esta seção fornece algumas noções elementares envolvendo mapeamentos monotônicos e pontos fixos que serão muito utilizadas no transcorrer deste trabalho. As definições apresentadas aqui seguem muito proximamente de [50].

Definição 2.71 *Seja C um conjunto. Uma relação $\mathcal{R}$ sobre C é qualquer subconjunto de $C \times C$.*

Será utilizada a notação infixa $x\mathcal{R}y$ para designar que $(x, y) \in R$.

Definição 2.72 *Uma relação $\mathcal{R}$ sobre um conjunto C é uma ordem parcial se as seguintes condições são verificadas:*

- **Reflexividade:** *$x\mathcal{R}x$, para todo $x \in C$;*
- **Simetria:** *$x\mathcal{R}y$ e $y\mathcal{R}x$ implica $x = y$, para todo $x, y \in C$;*
- **Transitividade:** *$x\mathcal{R}y$ e $y\mathcal{R}z$ implica $x\mathcal{R}z$ para todo $x, y, z \in C$.*

Adotando convenção padrão, será utilizado o símbolo $\leq$ para denotar uma relação parcial.

Definição 2.73 *Seja C um conjunto com uma ordem parcial $\leq$. Então $a \in C$ é um limite superior de um subconjunto X de C se $x \leq a$, para todo $x \leq X$. Similarmente, $b \in C$ é um limite inferior de X se $b \leq x$ para todo $x \in X$.*

Definição 2.74 *Seja C um conjunto com uma ordem parcial $\leq$. Então $a \in C$ é o menor limite superior de um subconjunto X de C se a é um limite superior de X e para todo limite superior a' de X, $a \leq a'$. Similarmente, $b \in C$ é o limite inferior de um subconjunto X de C se b é o maior limite inferior de X e para todo limite inferior b' de X, $b' \leq b$.*

Se o menor limite superior de X existe, ele é único e é denotado por $lub(X)$[6]. Similarmente, se o maior limite inferior de X existe, ele é único e é denotado por $glb(X)$[7].

Definição 2.75 *Um conjunto ordenado parcialmente L é um reticulado completo se e somente se $lub(X)$ e $glb(X)$ existem para todo subconjunto X de L.*

Nesta seção, será utilizado o símbolo $\top$ para designar o *elemento superior* $lub(L)$ e o símbolo $\bot$ para designar o *elemento inferior* $glb(L)$ do reticulado completo.

Definição 2.76 *Sejam L um reticulado completo e $T : L \to L$ um mapeamento. T é monotônico se toda vez que $x \leq y$, $T(x) \leq T(y)$.*

Definição 2.77 *Sejam L um reticulado completo e $X \subseteq L$. X é um conjunto direto se todo subconjunto finito de X tem um limite superior em X.*

Definição 2.78 *Sejam L um reticulado completo e $T : L \to L$ um mapeamento. T é finitário se $T(lub(X)) \subseteq lub(T(X)))$ para todo subconjunto direto X de L. T é chamado de contínuo se ele é monotônico e finitário. Uma definição equivalente de continuidade é T é contínuo se $T(lub(X)) = lub(T(X))$ para todo subconjunto direto X de L.*

Definição 2.79 *Sejam L um reticulado completo e $T : L \to L$ um mapeamento. $a \in L$ é o menor ponto fixo de T se é um ponto fixo, isto é, $T(a) = a$ e para todo ponto fixo b de T, $a \leq b$. De um modo similar é definido o maior ponto fixo.*

O próximo resultado é uma generalização do teorema do ponto fixo de Knaster-Tarski [88].

[6]Do inglês, least upper bound.
[7]Do inglês, greatest upper bound.

Proposição 2.80 *Sejam L um reticulado completo e $T : L \to L$ monotônico. Então, T possui um menor ponto fixo, $lfp(T)$[8] e um maior ponto fixo, $gfp(T)$[9]. Além do mais, $lfp(T) = glb\{x : T(x) = x\} = glb\{x : T(x) \leq x\}$ e $gfp(T) = lub\{x : T(x) = x\} = lub\{x : x \leq T(x)\}$.*

Na definição abaixo, é apresentada as potências ordinais de T.

Definição 2.81 *Sejam L um reticulado completo e $T : L \to L$ monotônico. Então*

$$\begin{aligned} T^{\uparrow 0} &= \bot \\ T^{\uparrow i+1} &= T(T^{\uparrow i}) \\ T^{\uparrow \delta} &= lub\{T^{\uparrow i} : i < \delta\} \text{ se } \delta \text{ é um limite ordinal.} \end{aligned}$$

$$\begin{aligned} T^{\downarrow 0} &= \top \\ T^{\downarrow i+1} &= T(T^{\downarrow i}) \\ T^{\downarrow \delta} &= glb\{T^{\downarrow i} : i < \delta\} \text{ se } \delta \text{ é um limite ordinal.} \end{aligned}$$

O próximo resultado caracteriza $lfp(T)$ e $gfp(T)$ em termos das potências ordinais de T. A prova dessa proposição pode ser verificada em [50].

Proposição 2.82 *Sejam L um reticulado completo e $T : L \to L$ um mapeamento monotônico. Então, para qualquer ordinal i, $T^{\uparrow i} \leq lfp(T)$ e $T^{\downarrow i} \geq gfp(T)$. Além do mais, existem ordinais j_1 e j_2 tal que $k_1 \geq j_1$ implica $T^{\uparrow k_1} = lfp(T)$ e $k_2 \geq j_2$ implica $T^{\downarrow k_2} = gfp(T)$.*

O menor i tal que $T^{\uparrow i} = lfp(T)$ é chamado de *fecho ordinal* de T. O próximo resultado mostra que sob a hipótese mais forte de que T é contínuo, o fecho ordinal de T é $\leq \omega$[10]. A prova dessa proposição também pode ser encontrada em [50].

Proposição 2.83 *Sejam L um reticulado completo e $T : L \to L$ contínuo. Então $lfp(T) = T^{\uparrow \omega}$.*

[8] *Do inglês, least fixpoint.*

[9] *Do inglês, greatest fixpoint.*

[10] Neste trabalho, ao se referir a limites ordinais, ω será utilizado para designar o primeiro ordinal infinito.

Capítulo 3

Fundamentos de Programação em Lógica

Neste capítulo, é apresentada um relato sobre programação em lógica que inicia com a lógica restrita à linguagem Horn e finda com uma discussão sobre a importância de semânticas paraconsistentes em programação em lógica estendida. A linguagem Horn é mostrada na seção 3.1, em que são descritas algumas de suas características, ressaltando a simplicidade do método da resolução linear para tais cláusulas. A seção 3.2 refere-se à utilização da lógica como uma linguagem de programação, sendo apresentadas a sua sintaxe, a sua semântica declarativa e a sua semântica operacional. Na seção 3.3, é introduzida a noção de programação normal em lógica com a apresentação dos conceitos de interpretação e modelo para tais programas. Também são descritas as duas principais semânticas declarativas para programas normais em lógica: a semântica dos modelos estáveis e a semântica bem fundada. Na seqüência, é feito um rápido comentário sobre as semânticas operacionais em programação normal em lógica. O capítulo termina com a apresentação da programação em lógica estendida. Nesse sentido é discutida a importância do operador de negação explícita, "$\neg$", em programação em lógica. Então, define-se a noção de linguagem para programas em lógica estendida e, por fim, é entabulada uma discussão sobre a conveniência de semânticas paraconsistentes para tais programas.

3.1 A Linguagem Horn

Como discorrido no capítulo anterior, para qualquer conjunto Γ_1 de sentenças da lógica de primeira ordem, existe um conjunto Γ_2 de sentenças na forma clausal tal que Γ_1 é insatisfatível se e somente se Γ_2 é também insatisfatível. Com isso, as questões pertinentes à satisfatibilidade ou insatisfatibilidade podem ser transladadas para as sentenças na forma clausal. Por sua vez, pelo teorema de Herbrand, é garantido que um conjunto Γ de

cláusulas é insatisfatível se e somente se Γ não possui modelos de Herbrand. Com isso, para determinar $\Gamma \models \gamma$, em que Γ é um conhunto de cláusulas e γ é uma cláusula, basta demonstrar que todo modelo de Herbrand para Γ é um modelo de Herbrand para γ.

No entanto, teorias definidas sobre a linguagem clausal continuam intratáveis. Felizmente, esse problema pode ser contornado restringindo as cláusulas a conterem no máximo um literal positivo. Tais cláusulas são chamadas de cláusulas Horn [57]. Ao restringir a linguagem da lógica de primeira ordem às cláusulas Horn, obtém-se importantes propriedades tanto do ponto de vista semântico quanto sintático.

Sob o aspecto semântico, sabe-se que para qualquer conjunto satisfatível de cláusulas Horn, é garantida a existência de um menor modelo de Herbrand. Isso se deve porque a intersecção dos modelos Herbrand para um conjunto satisfatível Γ de cláusulas Horn continua sendo um modelo para Γ. Assim, numa teoria satisfatível, a intersecção de todos os modelos de Herbrand para Γ é única e não vazia, sendo chamada de *menor modelo de Herbrand*, denotado por $\mathcal{M}_\Gamma$.

Em [91], é demonstrado que o menor modelo de Herbrand para um conjunto Γ de cláusulas Horn é precisamente o conjunto de literais básicos que são conseqüência lógica de Γ. Desse modo, seja Γ um conjunto de cláusulas Horn e γ uma cláusula Horn, $\Gamma \models \gamma$ se e somente se $\gamma \in \mathcal{M}_\Gamma$ ou ainda, $\Gamma \cup \{\neg\gamma\}$ é insatisfatível se e somente se $\gamma \in \mathcal{M}_\Gamma$. Por isso, costuma-se considerar o modelo $\mathcal{M}_\Gamma$ como o significado pretendido para Γ. Observe que para cláusulas arbitrárias, a propriedade da intersecção de modelos ainda ser um modelo nem sempre é verificada:

Exemplo 3.1 Seja $\Gamma = p(a) \vee p(b)$. Γ possui dois modelos de Herbrand: $M_1 = p(a)$ e $M_2 = p(b)$. A intersecção desse dois modelos é igual ao conjunto vazio, que não é um modelo para Γ.

Por outro lado, sob o aspecto sintático, a construção de um cálculo de resolução para cláusulas Horn também se torna bastante simplificada. Para provar a insatisfatibilidade de um conjunto de cláusulas Γ, como de praxe, parte-se de uma cláusula inicial e prossegue-se até encontrar $\square$ ou não ser mais possível aplicar a regra de resolução. No entanto, em cada aplicação dessa regra, precisa-se somente manter um registro da cláusula central atual, pois as cláusulas laterais são constituídas somentes por elementos de Γ[1]. Com isso, uma estrutura de dados bem mais simples é suficiente para armazenar os dados utilizados durante uma prova.

Definição 3.2 (Resolução linear para cláusulas Horn) *Sejam Γ um conjunto de cláusulas Horn e C_0 uma cláusula em Γ. Uma dedução linear de C_n de Γ com cláusula inicial C_0 é obtida da seguinte maneira:*

[1]Para conjuntos de cláusulas arbitrárias, as cláusulas laterais também poderiam conter as cláusulas centrais previamente deduzidas.

- *C_{i+1} é um resolvente da cláusula central C_i e da cláusula lateral B_i, em que B_i é uma cláusula de* Γ.

Uma *refutação linear* de Γ é uma dedução linear de $\square$ de Γ.

Exemplo 3.3 Seja $\Gamma = \{p(f(x)) \vee \neg p(x), p(a), \neg p(f(f(a)))\}$. Uma refutação linear para Γ a partir da cláusula inicial $\neg p(f(f(a)))$ é mostrada na figura 3.1:

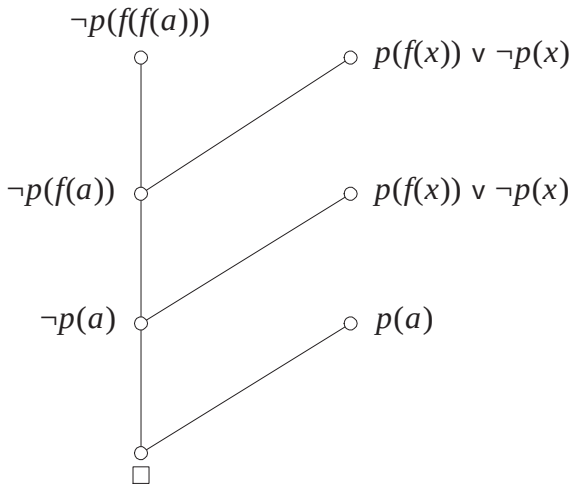

Figura 3.1: Resolução linear para cláusulas Horn

A corretude do método da resolução linear para as cláusulas Horn segue da corretude desse método para cláusulas arbitrárias. Em contrapartida, a sua completude é apresentada pelo teorema abaixo cuja prova pode ser verificada em [49].

Teorema 3.4 (Completude da resolução linear para cláusulas Horn) *Seja Γ um conjunto finito e insatisfatível de cláusulas Horn. Então para toda cláusula C relevante em Γ, existe um refutação linear de Γ com cláusula inicial C. Além disso, existe pelo menos uma cláusula relevante em Γ.*

Apesar de ser menos expressiva do que a linguagem da lógica de primeira ordem, a linguagem Horn permite a obtenção de soluções satisfatórias em muitas aplicações, sendo que a insatisfatibilidade de uma teoria $\mathcal{T}$ para o caso proposicional é um problema de complexidade linear. Por sua vez, na lógica de primeira ordem, tal problema é NP-completo. Por esses motivos, a linguagem Horn tem sido profusamente estudada como uma ferramenta eficiente para a construção de provadores automáticos de teoremas.

3.2 A Lógica como linguagem de programação

Em [46], é mostrado que a lógica também pode ser utilizada como um formalismo computacional ao qual um conjunto finito de sentenças representa um programa. Tais programas são chamados de programas em lógica. Essa noção foi um dos resultados obtidos a partir do grande impulso tomado pelas pesquisas envolvendo provas automáticas de teoremas com a definição da regra de resolução.

Numa visão mais ampla, um programa em lógica é um conjunto P de sentenças que representa os axiomas de uma teoria. Definir um programa em lógica significa, portanto, obter uma descrição axiomática de um determinado problema ou situação. Em oposição, os programas convencionais podem ser vistos como a descrição dos procedimentos para obter a solução de um problema. Por essa razão, diz-se que programas em lógica possuem uma interpretação declarativa enquanto que os programas convencionais, uma interpretação procedural. Além disso, programas em lógica também possuem uma interpretação procedural já que suas sentenças podem ser consideradas como definições de procedimentos [46].

Desse modo, do ponto de vista da lógica, a programação em lógica possibilita uma interpretação procedural das sentenças enquanto que do ponto de vista da programação, a programação em lógica possibilita uma intepretação declarativa dos componentes do programa.

Uma chamada a um programa em lógica P representa uma indagação se uma sentença S é conseqüência lógica de P. Uma execução de P a partir de S significa pesquisar uma dedução de S a partir de P numa teoria de prova adequada. Sob essa ótica, programação em lógica se aproxima bastante da área de prova automática de teoremas. No entanto, programação em lógica não se restringe a verificar se uma sentença é conseqüência lógica de P. A noção de resposta é ampliada visando permitir a computação da informação efetivamente extraída de P.

Com a introdução de uma interpretação declarativa em programação, um dos objetivos dos pesquisadores era o de separar a parte relacionada à representação do problema, da parte operacional, que soluciona o problema. Essa idéia foi defendida por Kowalski [42] sob a argumentação de que um algoritmo consiste de dois componentes disjuntos: lógica e controle. Assim, numa linguagem de programação puramente declarativa, o programador especificaria o componente lógico do algoritmo e o sistema de programação em lógica ficaria encarregado de todos os aspectos operacionais. Infelizmente, na prática, isso não pode ser alcançado e o projetista de um programa em lógica deve preocupar-se com questões procedimentais como a ordem das cláusulas no programa, a posição do operador *cut* numa regra e mesmo com a representação da negação default.

3.2.1 A sintaxe da programação em lógica

Restringindo as sentenças da linguagem de um programa em lógica às cláusulas Horn, Colmerauer [19] e Kowalski [46] definiram e implementaram a primeira linguagem de programação em lógica declarativa, o *PROLOG* (*PROgramming LOGic*). Neste trabalho, os programas construídos dessa linguagem serão referidos como programas em lógica definidos[2].

Um *alfabeto* $\mathcal{A}$ para um programa em lógica definido é constituído das seguintes classes disjuntas de símbolos:

- **constantes**;
- **símbolos funcionais**;
- **símbolos relacionais**;
- **variáveis**;
- **parentêses**: "(" e ")";
- **implicação revertida**: $\leftarrow$;
- **vírgula**: ",".

Os três últimos itens são comuns a todo alfabeto enquanto que os três primeiros podem variar de alfabeto para alfabeto. Como de praxe, as variáveis são assumidas serem infinitas e fixas enquanto que o conjunto de símbolos relacionais, funcionais e de constantes é infinito ou finito contável. Em particular, podem ser vazios. Por praticidade, assume-se normalmente que os conjuntos de constantes, símbolos funcionais e relacionais de um alfabeto $\mathcal{A}$ são determinados pelas constantes, símbolos funcionais e símbolos relacionais que explicitamente aparecem num programa.

As definições de termo e de átomo seguem similar as da lógica de primeira ordem. A partir de um alfabeto $\mathcal{A}$ de um programa em lógica definido, pode-se constituir três tipos de fórmulas (cláusulas Horn), que recebem uma nova representação para tornar a sua leitura computacional mais clara.

- Uma cláusula Horn $A \vee \neg B_1 \vee \ldots \vee \neg B_n$, em que $n \geq 0$, é representada em programação em lógica definida por $A \leftarrow B_1, \ldots, B_n$, e recebe o nome de *regra*. Em particular, se $n = 0$, essa regra é chamada de *fato* e o símbolo $\leftarrow$ é omitido.

[2]Do inglês, definite logic programs.

- $\neg B_1 \vee \ldots \vee \neg B_n$, $n \geq 1$, é uma cláusula Horn que não contém nenhum átomo. Em programação em lógica, é representada por $\leftarrow B_1, \ldots, B_n$ e recebe o nome de *meta*[3] ou cláusula objetivo.

- $\square$ é uma cláusula (Horn) vazia, recebendo em programação em lógica essa mesma representação.

A linguagem de um programa em lógica definido P é então determinada pelo conjunto de cláusulas Horn que podem ser formadas a partir do alfabeto para P.

Definição 3.5 (Programa em lógica definido) *Um programa em lógica definido sobre uma linguagem $\mathcal{L}$ é um conjunto finito de regras do tipo $A \leftarrow B_1, ..., B_n$, em que $n \geq 0$ e $A, B_1, ..., B_n$ são átomos de $\mathcal{L}$. O átomo A é chamado de cabeça da regra e $B_1, \ldots, B_n$ é uma conjunção de literais, que constitui o corpo da regra.*

Um termo, um átomo ou uma regra são chamados de básicos se eles não contêm variáveis. O universo de Herbrand $\mathcal{U}_P$ (resp. base de Herbrand $\mathcal{B}_P$) de um programa P é o conjunto de todos os termos básicos (resp. átomos básicos) de um alfabeto $\mathcal{A}$ determinado por um programa P.

A versão básica de um programa em lógica P é o conjunto (possivelmente infinito) de regras básicas obtidas de P substituindo cada uma das variáveis de P por elementos do seu universo de Herbrand.

Exemplo 3.6 Seja P o programa em lógica definido

$p(a)$
$p(b)$
$p(x) \leftarrow p(f(x, y))$
$q(y) \leftarrow p(y), p(x)$

Além dos símbolos fixos, o alfabeto de P é constituído pelas constantes $\{a, b\}$, pelos símbolos relacionais $\{p, q\}$; e pelo símbolo funcional f. Nesse programa, os fatos $p(a)$ e $p(b)$ são básicos. $\mathcal{U}_P$ é igual a $\{a, b, f(a), f(b), f(f(a)), f(f(b)), \ldots\}$ e $\mathcal{B}_P$ é igual a $\{p(a), p(b), q(a), q(b), p(f(a), p(f(b)), q(f(a)), q(f(b)), p(f(f(a))), p(f(f(b))), q(f(f(a))), q(f(f(b)))\ldots\}$.

3.2.2 A semântica da programação em lógica

Como programas em lógica possuem uma interpretação declarativa e uma procedural, dois tipos de semântica foram definidos: a semântica declarativa e a semântica operacional.

[3]Do inglês, goal.

- **Semântica Declarativa** - O significado de um programa P é determinado pelas conseqüências lógicas de P. Nessa semântica, detalhes sobre a execução do programa são irrelevantes, permitindo que o programador defina os resultados esperados sem se preocupar com as instruções para executar a sua computação.
- **Semântica Operacional** - O significado de um programa P é determinado pelo comportamento de uma máquina abstrata, o interpretador do programa. Nessa semântica, é descrito como a busca por uma prova ocorre a partir das regras de P.

Em [91], é demonstrado que a semântica declarativa é um caso especial da semântica da teoria dos modelos e a semântica operacional está incluída na sintaxe referente à teoria da prova. Desse modo, a equivalência entre a semântica declarativa e a operacional torna-se um caso especial do teorema da completude de Gödel [34, 35].

A semântica declarativa

Desde que um programa em lógica definido é um conjunto de cláusulas, o seu significado pode ser determinado restringindo o enfoque às interpretações de Herbrand. No restante deste trabalho, serão utilizadas simplesmente as denominações intepretação e modelo para se referirem respectivamente à interpretação e modelo de Herbrand.

Como pode ser verificado na seção 3.1, um conjunto Γ de cláusulas Horn sempre possui um menor modelo que corresponde ao significado pretendido para Γ. Assim, analisando um programa em lógica definido P como um conjunto de cláusulas Horn, o seu significado pode ser obtido através do menor modelo para P. Essa forma de definir uma semântica é chamada de declarativa. De posse desses elementos, seguem as definições abaixo:

Definição 3.7 (Interpretação) *Uma interpretação I para um programa em lógica definido P é qualquer subconjunto da base de Herbrand $\mathcal{B}_P$ de P.*

Alternativamente, uma intepretação pode ser definida da seguinte maneira:

Proposição 3.8 *Qualquer interpretação I pode ser equivalentemente vista com uma função $I : \mathcal{B}_P \to V$, em que $V = \{0, 1\}$, definida por*

$$I(A) = \begin{cases} 1 & se\ A \in I \\ 0 & se\ A \notin I \end{cases}$$

Relembrando que os elementos 1 e 0 de V denotam respectivamente os valores lógicos verdadeiro e falso. Como usual, modelos são definidos tomando como base uma função de valoração verdade:

Definição 3.9 (Valoração verdade) *Sejam I uma interpretação, C um conjunto de fórmulas da linguagem e $V = \{0, 1\}$. A valoração verdade $\hat{I}$ correspondente a I é uma função $\hat{I} : C \leftarrow V$ definida recursivamente como segue:*

- *Se A é um átomo básico, então $\hat{I} = I(A)$;*
- *Se $\mathcal{R}$ é uma fórmula do tipo $\leftarrow A_1, \ldots, A_n$, em que $n \geq 1$ e cada A_i, com $1 \leq i \leq n$, é um átomo, então*

$$\hat{I}(\mathcal{R}) = min(\hat{I}(A_1), \ldots, \hat{I}(A_n));$$

- *Se A é um átomo e S é da forma $A_1, \ldots, A_n$, em que $n \geq 1$ e cada A_i, com $1 \leq i \leq n$, é um átomo, então*

$$\hat{I}(A \leftarrow S) = \begin{cases} 1 & \text{se } \hat{I}(S) \leq \hat{I}(A) \\ 0 & \text{nos demais casos} \end{cases}$$

- $\hat{I}(\square) = 0$.

Definição 3.10 (Modelos) *Uma interpretação I é um modelo de um programa em lógica definido P sss para toda instância básica de uma regra $A \leftarrow B_1, \ldots, B_n$ de P com $n \geq 0$, tem-se que $\hat{I}(A \leftarrow B_1, \ldots, B_n) = 1$.*

Todo programa em lógica definido P possui pelo menos a base de Herbrand $\mathcal{B}_P$ como modelo, logo o conjunto de todos os modelos de Herbrand para P é sempre não vazio. Como as cláusulas Horn possuem a propriedade da intersecção dos modelos, a intersecção de todos os modelos de Herbrand para P corresponde ao menor modelo de Herbrand $\mathcal{M}_P$.

Definição 3.11 (Conseqüência Lógica) *Um átomo A básico é conseqüência lógica de um programa em lógica definido P, denotado por $P \models A$ sss qualquer modelo para P é também um modelo para A.*

A relação de conseqüência lógica, então, estabelece todos os átomos A básicos que são verdadeiros em todos os modelos de um programa P. Disso resulta que $P \models A$ se e somente se $A \in \mathcal{M}_P$. Portanto o problema de determinar que átomos são conseqüência lógica de P, reduz-se a determinar os elementos de $\mathcal{M}_P$. Com esse propósito, é definido em [91] o operador conseqüência imediata T_P.

Definição 3.12 (Operador T_p) *Seja P a versão básica de um programa em lógica. O operador conseqüência imediata T_P de uma interpretação I de P é definido como*

$$T_P(I) = \{H \mid \exists \mathbf{B}(H \leftarrow \mathbf{B} \in P, I \models \mathbf{B}\}).$$

Em particular, se A é um fato de P, então toda instância básica de A está em $T_P(I)$ para todo I.

Exemplo 3.13 Considere o programa em lógica definido abaixo:

$$P = \begin{cases} p(a) \leftarrow q(s), q(w) \\ q(w) \leftarrow r(s) \\ q(w) \leftarrow q(s) \\ q(s) \end{cases}$$

Seja $I = \{p(a), q(s), q(w), r(s)\}$. Aplicando o operador T_P ao conjunto I, obtém-se $T_P(I) = \{p(a), q(w), q(s)\}$.

Como observado em [91], é possível caracterizar os modelos de Herbrand de P a partir do operador T_P:

Teorema 3.14 (Caracterização de T_P) *[91] Uma interpretação de Herbrand I é um modelo de P sss $T_P(I) \subseteq I$.*

Como pode ser facilmente verificado, a coleção de todas as interpretações de Herbrand constitui um reticulado completo. Baseado nisso, algumas propriedades como a da monotonicidade e da continuidade podem ser exploradas:

Teorema 3.15 *[91] O operador T_P é contínuo sob a relação de ordem $\subseteq$ de inclusão de conjuntos.*

Como o operador T_P é contínuo, ele possui um menor ponto fixo que pode ser obtido da seguinte maneira:

Definição 3.16 *Sejam P um programa em lógica definido e $T_P^{\uparrow\omega}$ o menor ponto fixo do operador T_P, que pode ser calculado da seguinte maneira:*

$$\begin{aligned} T_P^{\uparrow 0} &= \emptyset \\ T_P^{\uparrow (i+1)} &= T_P(T_P^{\uparrow i}) \\ T_P^{\uparrow\omega} &= \bigcup_{i=0}^{\infty} T_P^{\uparrow i} \end{aligned}$$

O menor ponto fixo é atingido quando $T_P^{\uparrow k} = T_P(T_P^{\uparrow k}) = T_P^{\uparrow (k+1)} = T_P^{\uparrow\omega}$. Em se tratando de um operador contínuo, T_P sempre possui um menor ponto fixo $T_P^{\uparrow k}$, com $k \in I\!N$. Do teorema 3.14, resulta que $T_P^{\uparrow\omega}$ é o menor modelo para P, ou seja, $T_P^{\uparrow\omega} = \mathcal{M}_P$. Com isso, se A é um átomo, então $P \models A$ sss $A \in T_P^{\uparrow\omega}$; portanto o problema de determinar se um literal é verdadeiro num programa em lógica definido P resume-se a saber se esse literal pertence a $T_P^{\uparrow\omega}$.

Exemplo 3.17 Seja P o programa em lógica do exemplo 3.13:

$p(a) \leftarrow q(s), q(w)$
$q(w) \leftarrow r(s)$
$q(w) \leftarrow q(s)$
$q(s)$

O menor modelo para P pode ser obtido aplicando a definição 3.16:

$T_P^{\uparrow 0} = \emptyset$
$T_P^{\uparrow 1} = \{q(s)\}$
$T_P^{\uparrow 2} = \{q(s), q(w)\}$
$T_P^{\uparrow 3} = \{q(s), q(w), p(a)\}$
$T_P^{\uparrow 4} = \{q(s), q(w), p(a)\}$

Desse modo, $T_P^{\uparrow \omega} = T_P^{\uparrow 4} = \{p(a), q(s), q(w)\}$. Das considerações acima, sabe-se que $\{p(a), q(s), q(w)\}$ é o menor modelo para P.

As principais relações envolvendo o menor modelo $\mathcal{M}(P)$ e operadores de ponto fixo apresentadas nessa seção estão sumarizadas no teorema abaixo:

Teorema 3.18 *[7] O menor modelo de Herbrand $\mathcal{M}_P$ satisfaz as seguintes propriedades:*

1. *$\mathcal{M}(P)$ é o menor ponto fixo de T_P;*
2. *$\mathcal{M}(P) = T_P^{\uparrow \omega}$;*
3. *$\mathcal{M}(P) = \{A \mid A$ é uma instância básica e $P \models A\}$.*

A semântica operacional

A interpretação procedural de um programa em lógica P descreve como ocorre a busca por uma derivação de $A_1, \ldots, A_n$ a partir de P, em que cada A_i, $1 \leq i \leq n$, é um átomo. Em outras palavras, a interpretação procedural descreve a relação $P \vdash A_1, \ldots, A_n$. A seqüência de átomos $A_1, \ldots, A_n$ é chamada de *consulta* e pode ser intuitivamente interpretada como uma fórmula $\exists x_1, \ldots, \exists x_n (A_1 \wedge \ldots \wedge A_n)$, em que $x_1, \ldots, x_n$ são todas as variáveis que ocorrem em $A_1, \ldots, A_n$.

Devido a sua simplicidade e eficiência, em programação em lógica, normalmente é utilizado o sistema de resolução como teoria formal. Nesse sistema, uma prova de $\exists x_1, \ldots, \exists x_n (A_1 \wedge \ldots \wedge A_n)$ a partir de P é obtida ao encontrar uma refutação de $P \cup \{\neg(\exists x_1, \ldots, \exists x_n (A_1 \wedge \ldots \wedge A_n))\}$. Entretanto, $\neg(\exists x_1, \ldots, \exists x_n (A_1 \wedge \ldots \wedge A_n))$ é equivalente a cláusula Horn $\forall x_1, \ldots, \forall x_n (\neg A_1 \vee \ldots \vee \neg A_n)$, que, por sua vez, pode ser denotada pela meta ou cláusula objetivo $\leftarrow A_1, \ldots, A_n$. Portanto, a relação de prova $P \vdash A_1, \ldots, A_n$

pode ser obtida ao encontrar uma refutação a partir de $P \cup \{\leftarrow A_1, \ldots, A_n\}$. Às vezes, é utilizada a denominação *meta definida* para reforçar a idéia de que uma certa meta é constituída somente por átomos.

Na interpretação procedural, a computação é iniciada a partir da meta $\leftarrow A_1, \ldots, A_n$. Como o leitor pode verificar, uma derivação por resolução dessa cláusula a partir de um programa P equivale a encontrar uma regra de P cuja cabeça é unificável com algum elemento da meta. O processo continua a partir da cláusula resultante até a cláusula vazia ser gerada ou não houver mais nenhuma regra unificável em P.

Dada uma meta, eventualmente várias regras podem ser unificáveis, sendo que cada uma dessas regras unificáveis deriva uma nova cláusula objetivo. Um procedimento de prova que seqüencia a geração de derivações na busca por uma refutação comporta-se como um interpretador procedural para o programa. Tal interpretação é a que define a semântica operacional.

Existem vários procedimentos de refutação baseados na regra de resolução que são refinamentos da versão original definida por Robinson [83]. Um deles, chamado de *SLD* (*Selection rule driven Linear resolution for Definite clauses*), é de especial interesse para a programação em lógica e foi apresentado por Kowalski em [46]. Basicamente, *SLD* é um método de resolução linear que utiliza uma regra de seleção em programas em lógica definidos.

Definição 3.19 (Regra de seleção) *Uma regra de seleção é uma função de um conjunto de metas a um conjunto de átomos tal que o valor da função para uma meta M é um dos átomos pertencentes a M, que é chamado de átomo selecionado de M.*

Definição 3.20 *Sejam M uma meta* $\leftarrow A_1, \ldots, A_m, \ldots, A_k$ *e C, uma cláusula do tipo* $A \leftarrow B_1, \ldots, B_q$. *Então* M' *é derivada de M e C com mgu* θ *se as seguintes condições são obedecidas:*

1. A_m *é um átomo selecionado pela regra de seleção;*
2. θ *é um mgu de* A_m *e A;*
3. M' *é a meta* $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots A_k)\theta$.

M' é então chamado de resolvente de M e C.

Definição 3.21 (Derivação SLD) *Sejam P um programa em lógica definido e M uma meta definida. Uma derivação SLD de* $P \cup \{M\}$ *consiste de uma seqüência (finita ou infinita)* $M_0 = M, M_1, \ldots$ *de metas; uma seqüência* $C_1, C_2, \ldots$ *de variantes de cláusulas de P e de uma seqüência* $\theta_1, \theta_2, \ldots$ *de* mgu_s *tal que cada* M_{i+1} *é derivado de* M_i *e* C_{i+1} *usando* θ_{i+1}. *Uma meta* M_i *é dita estar num nível i da derivação SLD.*

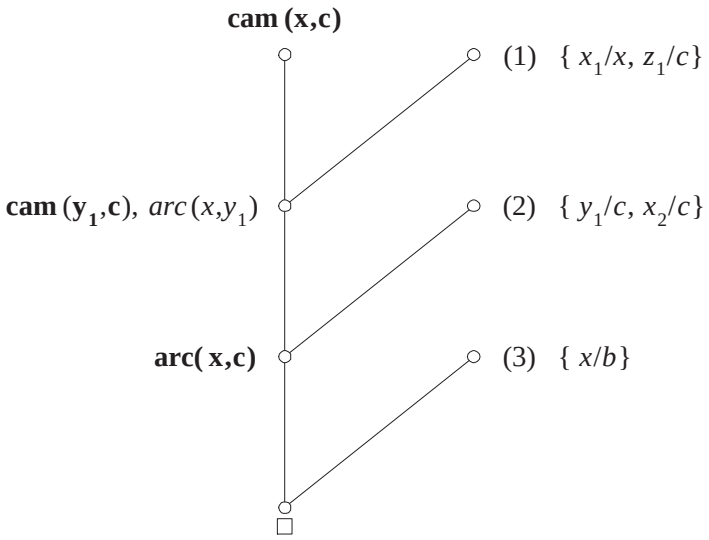

Figura 3.2: Refutação *SLD*

Cada C_i é uma variante da cláusula do programa correspondente tal que C_i não deve ter variáveis que já ocorreram nas derivações até M_{i-1}. Isso é necessário para permitir a unificação de cláusulas que não podem ser unificados por conta de terem variáveis comuns. Por exemplo, $\leftarrow p(x)$ e $p(f(x))$ não são unificáveis, mas renomeando $p(x)$ como $p(y)$, obtém-se que $y/f(x)$ é o *mgu* dessas duas cláusulas. Cada uma das variantes acima das cláusulas $C_1, C_2, \ldots$ do programa é chamada de *cláusula de entrada* da derivação.

Definição 3.22 *Uma refutação SLD de $P \cup \{M\}$ é uma derivação finita de $P \cup \{M\}$ que tem a cláusula vazia $\square$ como a última cláusula na derivação.*

Exemplo 3.23 Sejam P o programa em lógica

(1) $cam(x, z) \leftarrow cam(y, z), arc(x, y)$
(2) $cam(x, x)$
(3) $arc(b, c)$

e M a meta $\leftarrow cam(x, c)$. Uma refutação *SLD* de $P \cup \{M\}$ é mostrada na figura 3.2, em que para cada derivação realizada, os átomos selecionados estão em negrito e as substituições executadas estão dispostas junto com as respectivas regras de P.

Uma derivação *SLD* pode ser *finita* ou *infinita*. Uma derivação *SLD* finita pode ser *bem sucedida* ou *falha*. Uma derivação bem sucedida de $P \cup \{M\}$ é exatamente uma refutação *SLD* de $P \cup \{M\}$. Por outro lado, uma derivação *SLD* falha é uma que termina com a meta não vazia, pois o átomo selecionado não unifica com a cabeça de nenhuma regra de P.

3.2.3 Corretude e completude da resolução SLD

Antes de apresentar alguns resultados envolvendo a corretude e a completude da resolução *SLD*, são definidos os seguites conceitos:

Definição 3.24 *O conjunto sucesso de um programa P é constituído por todo $A \in \mathcal{B}_P$ tal que $P \cup \{A\}$ tem uma refutação SLD.*

Definição 3.25 *Sejam P um programa em lógica definido e M uma meta definida. Uma resposta computada*[4] *θ para $P \cup \{M\}$ é a substituição obtida restringindo a composição $\theta_1 \ldots \theta_n$ às variáveis de M, em que $\theta_1 \ldots \theta_n$ é a seqüência de mgus utilizados numa refutação SLD de $P \cup \{M\}$.*

Exemplo 3.26 Seja P o programa em lógica

$cam(x, z) \leftarrow cam(y, z), arc(x, y)$
$cam(x, x)$
$arc(b, c)$

e $M = \leftarrow cam(x, c)$ a meta apresentados no exemplo 3.23. Como o leitor pode verificar $\{x/b\}$ e $\{x/c\}$ são duas respostas computadas para $P \cup \{M\}$.

A resposta computada deve ser considerada como o resultado computado pelo interpretador na busca por uma refutação de M. Com isso, ela proporciona os valores para as variáveis de M que torna M verdadeiro em relação a uma derivação *SLD* bem sucedida de M. De posse dessas noções, seguem os resultados abaixo:

Teorema 3.27 (Corretude da resolução SLD) *[18] Sejam P um programa em lógica definido e M uma meta definida. Se existe uma derivação bem sucedida SLD de $P \cup \{M\}$ com resposta computada θ, então $P \models Q\theta$.*

Relembrando que $P \models Q\theta$ é equivalente a verificar se $Q\theta \in \mathcal{M}_P$. Com relação ao inverso do teorema acima, não é exatamente possível prová-lo porque respostas computadas são sempre mais gerais do que as conseqüências lógicas de um programa P [50]. No entanto, pode-se provar que toda conseqüência lógica de P é uma instância de uma resposta computada. Baseado nisso, é definido o teorema da completude. Esse resultado ainda estabelece que uma derivação bem sucedida existe independentemente da regra de seleção escolhida.

[4] *Do inglês, computed answer.*

Teorema 3.28 (Completude da resolução SLD) *[18] Sejam P um programa em lógica definido, M uma meta definida e θ uma substituição básica. Se $P \models M\theta$, então para toda regra de seleção $\mathcal{R}$, existe uma derivação bem sucedida de $P \cup \{M\}$ via $\mathcal{R}$ com resposta computada ν tal que $M\nu$ é mais geral do que $M\theta$.*

Ainda relacionado à completude da resolução *SLD*, Apt e van Emden em [8] demonstraram que o conjunto sucesso de um programa em lógica P é igual a $\mathcal{M}_P$.

3.2.4 Procedimento de refutação *SLD*

Na busca por uma refutação *SLD*, vários caminhos precisam ser testados até a cláusula vazia ser obtida ou não haver mais caminhos possíveis. O *espaço de busca SLD* representa o conjunto de todos esses caminhos possíveis e é normalmente definido como um tipo de árvore chamado de árvore *SLD*. Como a cláusula vazia pode ser obtida independentemente da regra de seleção, tal regra pode ser escolhida de antemão, reduzindo sensivelmente o espaço de busca.

Definição 3.29 (Árvore SLD) *Sejam P um programa em lógica definido e M uma meta definida. Uma árvore SLD para $P \cup \{M\}$ é uma árvore satisfazendo as seguintes condições:*

1. *Cada nó da árvore é uma meta definida possivelmente vazia;*
2. *O nó raiz é M;*
3. *Seja $\leftarrow A_1, \ldots, A_m, \ldots, A_k$, com $k \geq 1$, um nó na árvore e suponha que A_m é o átomo selecionado. Então para cada cláusula de entrada $A \leftarrow B_1, \ldots, B_q$ que seja uma variante de alguma cláusula de P tal que A_m e A são unificáveis com mgu θ, o nó tem um filho*

$$\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta;$$

4. *Nós que são cláusulas vazias não têm filhos.*

Cada ramo numa árvore *SLD* corresponde a uma derivação de $P \cup \{M\}$. Assim, nessa árvore, podem ser encontrados ramos bem sucedidos, falhos e infinitos, que correspondem respectivamente às derivações bem sucedidas, falhas e infinitas.

Uma *regra de busca* é uma estratégia para encontrar ramos bem sucedidos numa árvore. Junto com uma regra de seleção, uma regra de busca especifica um *procedimento de refutação*. Como mostrado no capítulo anterior, as duas principais regras de busca são a de busca em amplitude e a de busca em profundidade. Na estratégia de busca em profundidade, nem sempre é garantido encontrar um ramo bem sucedido numa árvore

SLD. Para isso é necessário uma regra de busca em amplitude. No entanto, a regra de busca em amplitude demanda um alto custo computacional. Por esse motivo, a maioria dos procedimentos de refutação utilizam a busca em profundidade.

Para sistemas que adotam a busca em profundidade, a regra de busca se reduz a um mecanismo de ordenação, ou seja, uma regra que especifica a ordem em que as cláusulas do programa vão ser utilizadas. Por serem mais simples, normalmente utiliza-se como regra de ordenação a ordem em que as cláusulas estão dispostas num programa. No restante deste trabalho, ao se referir a procedimentos de prova, é assumido que eles utilizam uma regra de seleção que sempre escolhe o literal mais à esquerda e a regra de busca em profundidade ao qual as cláusulas são selecionadas na ordem que elas aparecem no programa.

Por convenção, na representação gráfica de uma árvore de refutação, os elementos escolhidos pela regra de seleção estarão em negrito e as cláusulas do programa bem como as substituições efetuadas num passo da derivação estarão dispostas ao lado da aresta correspondente. Além disso, as cláusulas de entrada num nível i são obtidas das cláusulas do programa adicionando um subscrito i a todas as variáveis que forem utilizadas anteriormente na derivação.

Exemplo 3.30 Considere o seguinte programa em lógica P:

(1) $cam(x, z) \leftarrow cam(y, z), arc(x, y)$
(2) $cam(x, x)$
(3) $arc(b, c)$

Seja $M = \leftarrow cam(x, c)$ uma meta. Uma árvore *SLD* para $P \cup \{M\}$ é mostrada na figura 3.3.

3.3 Programação normal em lógica

Na programação em lógica definida, a informação negativa nunca é deduzida, ou seja, dado um átomo A pertencente a $\mathcal{B}_P$, não se pode provar que $\neg A$ é conseqüência lógica de um programa definido P. Para isso, seria preciso provar que $P \cup \{A\}$ é insatisfatível. No entanto, $P \cup \{A\}$ é sempre satisfatível, pois tem pelo menos a base de Herbrand $\mathcal{B}_P$ como modelo.

Por admitirem somente conclusões positivas, semânticas para programas em lógica definidos são monotônicas. Isso quer dizer que as conclusões obtidas previamente nunca são questionadas com o acréscimo de informações. Portanto, o número de conclusões derivadas jamais diminui.

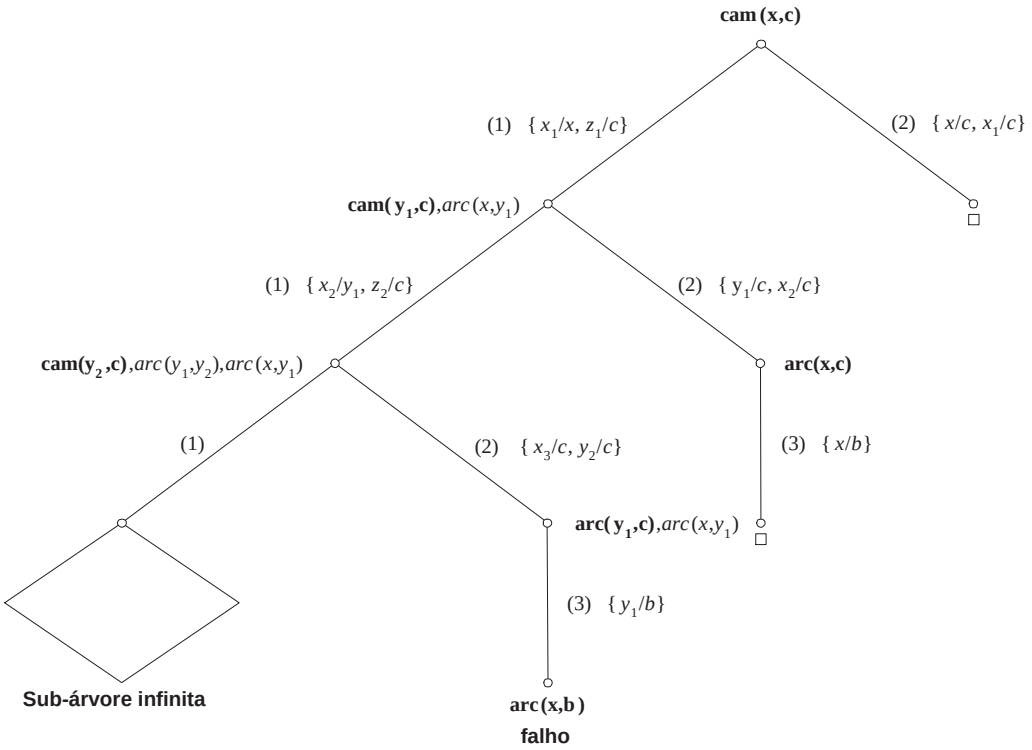

Figura 3.3: Árvore *SLD*

O raciocínio monotônico é adequado para situações em que se tem um total conhecimento do mundo como por exemplo o mundo matemático. Em oposição, para situações que envolvem o raciocínio do senso comum e em muitas aplicacões de Inteligência Artificial, é necessário utilizar formas de raciocínio não monotônico. Visando formalizar esse tipo de raciocínio, varias definições foram apresentadas tais como a lógica default [82], circunscrição [56] e a lógica auto-epistêmica [58, 59].

Para introduzir o raciocínio não monotônico em programação em lógica e permitir que informação negativa seja deduzida, foi apresentada a *regra da negação como falha*. Segundo essa regra, nada pode ser concluído falso, exceto assumindo falso o que não é finitamente provado ser verdadeiro.

Uma vez que literais negativos podem ser derivados de um programa positivo, é natural estender a sintaxe desses programas para permitir literais negativos no corpo das regras. Programas com essa característica são chamados de *normais*.

Definição 3.31 (Programa normal em lógica) *Um programa normal em lógica é um conjunto de regras da forma*

$$A \leftarrow B_1, \ldots B_m, not\ C_1, \ldots, not\ C_n, \qquad (m, n \geq 0)$$

em que A, cada B_i e cada C_j[5] *são átomos. Os literais do tipo not C são chamados de default. Como de praxe, se o corpo da regra é vazio, o símbolo $\leftarrow$ é omitido e tais regras são chamadas de fatos.*

As definições de termos, átomos, universo de Herbrand e base de Herbrand seguem análogas às apresentadas para programas em lógica definidos. Por sua vez, um *literal* é um átomo A ou a sua negação default *not* A. A *linguagem* $\mathcal{L}$ de um programa normal em lógica P é constituída por todas as cláusulas que podem ser formadas a partir do alfabeto $\mathcal{A}$ sobre P.

3.3.1 Interpretações e modelos de programas normais

As definições de interpretação, valor verdade e de modelo seguem muito próximas às das respectivas definições para programa em lógica definida. No entanto, por conta da introdução do operador *not*, algumas novas características são introduzidas.

Definição 3.32 (Interpretação bivalorada) *Uma interpretação I bivalorada para um programa normal em lógica P é qualquer subconjunto da base de Herbrand $\mathcal{B}_P$ de P.*

Claramente, qualquer interpretação bivalorada I pode ser equivalentemente vista como o conjunto

$$T \cup not\ F^6,$$

em que T é o conjunto dos átomos que são verdadeiros em I e $F = \mathcal{B}_P - T$ é o conjunto de átomos que são falsos em I.

Em [76], é argumentado que uma interpretação de um programa P pode ser considerada como mundos possíveis representando estados possíveis de nosso conhecimento sobre o significado de P. Desde que o conhecimento normalmente é incompleto, é necessária uma interpretação em que alguns átomos nem sejam verdadeiros nem sejam falsos. Para tanto, é utilizada uma interpretação trivalorada.

Definição 3.33 (Interpretação trivalorada) *Uma interpretação I trivalorada de um programa P é o conjunto*

$$T \cup not\ F,$$

em que T e F são subconjuntos de $\mathcal{B}_P$. O conjunto T contém todos os átomos básicos que são verdadeiros em I e o conjunto F, todos os átomos básicos que são falsos em I; o valor verdade dos átomos básicos restantes de $\mathcal{B}_P$, que não pertencem a T nem a F, é indefinido.

[5] *Naturalmente,* $1 \leq i \leq m$ *e* $1 \leq j \leq n$.

[6] Em que *not* $\{a_1, \ldots, a_n\}$ significa $\{not\ a_1, \ldots, not\ a_n\}$.

A interpretação bivalorada representa um caso especial da interpretação trivalorada acrescida da restrição $\mathcal{B}_P = T \cup F$. Alternativamente, uma interpretação pode ser vista sob o formato de função:

Proposição 3.34 *Qualquer interpretação $I = T \cup not\ F$ pode ser equivalentemente vista como uma função $I : \mathcal{B}_H \to V$ em que $V = \{0, \frac{1}{2}, 1\}$, definido como*

$$I(A) = \begin{cases} 0 & se\ not\ A \in I \\ 1 & se\ A \in I \\ \frac{1}{2} & Nos\ demais\ casos \end{cases}$$

Como de praxe, $1, \frac{1}{2}$ e 0 denotam respectivamente os valores verdadeiro, indefinido e falso. Obviamente, para interpretações bivaloradas não existe átomo A básico tal que $I(A) = \frac{1}{2}$.

Definição 3.35 (Valoração verdade) *Sejam I uma interpretação e C o conjunto de todas as fórmulas da linguagem, a valoração verdade $\hat{I}$ correspondente a I é uma função $\hat{I} : C \to V$ definida recursivamente como segue:*

- *Se A é um átomo básico, então $\hat{I}(A) = I(A)$;*
- *Se A é um átomo básico, então $\hat{I}(not\ A) = 1 - \hat{I}(A)$;*
- *Se $\mathcal{R}$ é uma fórmula do tipo $\leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n$, em que $m, n \geq 0$ e cada A_i e cada B_j, com $1 \leq i \leq m$ e $1 \leq j \leq n$, é um literal objetivo, então*

$$\hat{I}(\mathcal{R}) = min(\hat{I}(A_1), \ldots, \hat{I}(A_m), \hat{I}(not\ B_1), \ldots, \hat{I}(not\ B_n));$$

- *Se A é um átomo e S é da forma $A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n$, em que $m, n \geq 0$ e cada A_i e cada B_j, com $1 \leq i \leq m$ e $1 \leq j \leq n$, é um literal objetivo, então*

$$\hat{I}(A \leftarrow S) = \begin{cases} 1 & se\ \hat{I}(S) \leq \hat{I}(L) \\ 0 & nos\ demais\ casos \end{cases}$$

- $\hat{I}(\Box) = 0$.

Definição 3.36 (Modelo) *Um interpretação I bivalorada (resp. trivalorada) é um modelo bivalorado (resp. trivalorado) de um programa P sss para toda instância básica de uma regra $A \leftarrow B_1, \ldots, B_n$ de P com $n \geq 0$, tem-se que $\hat{I}(A \leftarrow B_1, \ldots, B_n) = 1$.*

Em estudos comparativos de interpretações, serão bastantes utilizadas dois tipos de ordenações entre interpretações: o ordenamento clássico ou padrão da verdade e o ordenamento de Fitting.

Definição 3.37 (Ordenamento clássico ou padrão da verdade[7]) *Sejam* $I_1 = T_1 \cup not\ F_1$ *e* $I_2 = T_2 \cup not\ F_2$ *duas interpretacões com relação à linguagem de um dado programa em lógica.* $I_1 \preceq_T I_2$ *sss* $T_1 \subseteq T_2$ *e* $F_2 \subseteq F_1$.

Sejam $\mathcal{I}$ *um conjunto de interpretações e* $\mathcal{J} = \{J_s \mid s \in S\}$, *um subconjunto de* $\mathcal{I}$ *com* $J_s = \langle T_s, F_s \rangle$. *Uma interpretação* $I \in \mathcal{I}$ *é chamada de minimal em* $\mathcal{I}$ *se não existe intepretação* $I' \in \mathcal{I}$ *tal que* $I' \preceq_T I$ *e* $I \neq I'$. *O menor limite superior de* $\mathcal{J}$ *com respeito a* $\preceq_T$ *é obtido como segue:*

$$\bigsqcup\nolimits_{T_{s\in S}} J_s = \bigcup_{s\in S} T_s \cup not \bigcap_{s\in S} F_s.$$

Definição 3.38 (Ordenamento de Fitting) *Sejam* $I_1 = T_1 \cup not\ F_1$ *e* $I_2 = T_2 \cup not\ F_2$ *duas interpretacões com relação a linguagem de um dado programa em lógica.* $I_1 \preceq_F I_2$ *sss* $T_1 \subseteq T_2$ *e* $F_1 \subseteq F_2$. *Como* $I_1 \preceq_F I_2$ *sss* $I_1 \subseteq I_2$, *às vezes, será utilizada a segunda notação quando isso não causar confusão.*

Sejam $\mathcal{I}$ *um conjunto de interpretações e* $\mathcal{J} = \{J_s \mid s \in S\}$, *um subconjunto de* $\mathcal{I}$ *com* $J_s = \langle T_s, F_s \rangle$. *Uma interpretação* $I \in \mathcal{I}$ *é chamada de F-minimal em* $\mathcal{I}$ *se não existe intepretação* $I' \in \mathcal{I}$ *tal que* $I' \preceq_F I$ *e* $I \neq I'$. *O menor limite superior de* $\mathcal{J}$ *com respeito a* $\preceq_F$ *é obtido como segue:*

$$\bigsqcup\nolimits_{F_{s\in S}} J_s = \bigcup_{s\in S} T_s \cup not \bigcup_{s\in S} F_s.$$

Observe que a menor interpretação sob o ordenamento clássico é $I = \langle \emptyset, \mathcal{B}_P \rangle$ e a menor interpretação sob o ordenamento de Fitting é $I = \langle \emptyset, \emptyset \rangle$. Se $I_1 \preceq_T I_2$, então I_2 possui mais literais verdadeiros que I_1. Por sua vez, se $I_1 \preceq_F I_2$, o grau de indefinição em I_1 é mais alto do que em I_2. Por essa razão, costuma-se dizer que o ordenamento clássico está relacionado à quantidade de verdade enquanto que o ordenamento de Fitting está relacionado à quantidade de informação.

3.3.2 Semânticas declarativas para programas normais

Em contrapartida a esse aspecto procedural da negação como falha, várias tentativas foram feitas para lhe dar uma semântica declarativa. A primeira a obter uma aceitação da comunidade de programação em lógica foi a semântica de completação de Clark[8] [17]. Essa

[8] Do inglês, completion semantics of Clark.

semântica é matematicamente elegante e é baseada na idéia de que no discurso comum, freqüentemente tende-se a usar declarações com "se" quando a intenção do locutor é dizer "se e somente se". Intuitivamente, na completação de um programa normal, isso pode ser visto como substituir a implicação reversa $\leftarrow$ pela relação de equivalência $\equiv$. A semântica da completação de Clark é então determinada pelos modelos bivalorados do programa completado.

No entanto, a semântica de Clark tem alguns sérios problemas relacionados à introdução de tautologias indesejáveis e contradições. Esse último problema faz com que nem todo programa normal segundo essa semântica tenha significado. Os dois próximos exemplos ilustram respectivamente essas situações:

Exemplo 3.39 Seja P o programa em lógica

$$a \leftarrow a.$$

O programa completado é $a \equiv a$, ou seja, um significado arbitrário é atribuído ao programa.

Exemplo 3.40 Considere o programa em lógica

$$a \leftarrow not\ a.$$

O programa completado é $a \equiv not\ a$, ou seja, o programa é insatisfatível.

Em [27], é mostrado que esse último problema pode ser suscintamente eliminado ao se considerar modelos trivalorados em vez de modelos bivalorados. Com esse objetivo, é definida uma semântica que atribui um único modelo para cada programa, chamada de *semântica de Fitting* [27]. Posteriormente, em [47] é apresentada a sua versão recursivamente enumerável.

No exemplo 3.40, o programa $P = \{a \leftarrow not\ a\}$ tem o menor modelo trivalorado $M = \langle \emptyset, \emptyset \rangle$. Assim, em M o átomo a é indefinido e portanto a regra $a \leftarrow not\ a$ é verdadeira. Todavia pela semântica de Fitting, no exemplo 3.39, o menor modelo também é igual a $M = \langle \emptyset, \emptyset \rangle$, ou seja, o átomo a possui o valor indefinido, contrariando o resultado esperado de que a seja falso. Afinal de contas, não há diferença alguma entre incluir ou não essa regra num programa. Esse problema está relacionado à impossibilidade de representar o fecho transitivo usando completação [48]:

Para resolver tais problemas, algumas semânticas foram definidas. A princípio, essas semânticas não eram aplicáveis a todo programa normal em lógica, mas somente a restrições sintáticas desses programas. Assim, somente programas que estivessem de acordo com essas restrições possuíam semântica. As duas principais semânticas para programação normal em lógica surgiram somente no final dos anos 80 e início dos anos 90, nomeadamente, a semântica dos modelos estáveis e a semântica bem fundada.

Semântica dos modelos estáveis

Em [28], é introduzida a semântica dos modelos estáveis por meio de uma definição baseada em pontos fixos. Essa semântica generaliza resultados obtidos das semânticas referidas acima para uma classe mais ampla de programas. Para defini-la, inicialmente é apresentado o operador de Gelfond-Lifschitz [28]:

Definição 3.41 (Operador de Gelfond-Lifschitz) *Sejam P um programa normal em lógica e I uma interpretação bivalorada. A transformação GL de P modulo I é o programa $\frac{P}{I}$ obtido de P executando as seguintes operações:*

- *Remova de P todas as regras que contenham um literal default not A tal que $A \in I$;*
- *Remova das regras restantes todos os literais default.*

Desde que $\frac{P}{I}$ é um programa definido, ele possui um único menor modelo J. Seja $\Gamma(I) = J$.

Em [28], é demonstrado que todo ponto fixo de Γ para um programa P é um modelo de P.

Definição 3.42 (Semântica dos modelos estáveis) *[28] Uma interpretação bivalorada I de um programa normal em lógica P é chamada de modelo estável de P sss $\Gamma(I) = I$. Um átomo A de P é verdadeiro sobre a semântica dos modelos estáveis sss A pertence a todos os modelos estáveis de P.*

Intuitivamente, os modelos estáveis de um programa normal em lógica P representam os conjuntos possíveis de conclusões que um agente racional pode ter sobre a base de informação expressa pelas regras de P. Se I é o conjunto de átomos básicos que o agente acredita ser verdadeiro, então qualquer regra que tem um literal *not* A com $A \in I$ no corpo não é utilizada, logo pode ser removida. Ademais, qualquer literal *not* A com $A \notin I$ é trivial, logo pode ser retirado do corpo de uma regra. Isso conduz ao programa definido $\frac{P}{I}$. Se I é exatamente o conjunto de átomos que logicamente segue de $\frac{P}{I}$, então o conjunto de crenças I é "racional".

A semântica dos modelos estáveis atribui falso para o átomo a no exemplo 3.39 e não está definida no exemplo 3.40. Além disso, em alguns casos mais de um modelo estável é obtido de um programa em lógica. Por definição, um literal é verdadeiro nessa semântica se ele pertence a todos os modelos estáveis e é falso nos demais casos.

Exemplo 3.43 Considere o programa em lógica abaixo:

$c \leftarrow a \qquad a \leftarrow not\ b$

$c \leftarrow b \qquad b \leftarrow not\ a$

Esse programa possui dois modelos estáveis $\{c, a\}$ e $\{c, b\}$. Desse modo, pela semântica dos modelos estáveis, somente o átomo c é verdadeiro e os átomos a e b são falsos.

Semântica bem fundada

Mesmo possuindo uma larga vantagem se comparada com seus predecessores, a semântica dos modelos estáveis ainda continua tendo alguns sérios problemas:

- Não está definida para todo programa. Um desses programas é $P = \{a \leftarrow not\ a\}$;
- Em alguns programas a semântica bem fundada não conduz ao resultado esperado. Por exemplo, seja P o programa normal em lógica abaixo:

$$a \leftarrow not\ b$$
$$b \leftarrow not\ a$$
$$c \leftarrow not\ a$$
$$c \leftarrow not\ c$$

  Seu único modelo estável é $\{b, c\}$. Assim, b e c são conseqüências da semântica dos modelos estáveis de P. No entanto, se c é adicionado a P, a semântica de P muda e b não é mais derivado;
- De um modo geral, não é obtido um único modelo para cada programa e o procedimento para computar os modelos estáveis é NP-completo;
- Por ser definida sobre uma interpretação bivalorada, a semântica dos modelos estáveis carece de expressividade.

Para sobrepujar tais problemas, foi proposto em [93] a *semântica bem fundada* (*WFS*)[9]. Essa semântica está unicamente definida para cada programa. Em se tratando de programas básicos finitos, a complexidade para verificar se um dado literal pertence ao modelo bem fundado é polinomial [93].

Além disso, por sua naturalidade, várias descrições equivalentes de semânticas bem fundadas foram propostas [93, 80, 81, 92, 12, 76]. Por causa de sua relação muito forte com a semântica dos modelos estáveis, nesta seção é apresentada a definição baseada em pontos fixos iterados introduzida em [76]. Ainda em [76], é demonstrada que a semântica bem fundada pode ser também vista como uma generalização da semântica dos modelos estáveis para interpretações trivaloradas. Para tanto, é mostrado que a definição original de modelos bem fundados [93] coincide com o menor modelo estável trivalorado sob a ordem de Fitting.

Para definir um modelo estável parcial, em [76], é feita uma expansão da linguagem de programas normal em lógica, acrescentando o átomo $\mathbf{u}$ que possui valor indefinido para toda interpretação. Assim, é assumido que $\hat{I}(\mathbf{u}) = \hat{I}(not\ \mathbf{u}) = \frac{1}{2}$.

[9] Do inglês, *Well-Founded Semantics*.

Um programa não negativo é aquele constituído por regras cujas premissas são átomos ou **u**. É provado em [76] que todo programa não negativo possui um menor modelo trivalorado. Com isso, tem-se a seguinte generalização do operador de Gelfond-Lifschitz:

Definição 3.44 (Operador Γ) *[76] Sejam P um programa normal em lógica e I uma interpretação trivalorada. A transformação GL de P modulo I é o programa $\frac{P}{I}$ obtido de P executando as operações abaixo:*

- *Remova de P todas as regras que contenham um literal default not A tal que $I(A) = 1$;*
- *Substitua nas regras restantes de P aqueles literais default not A tal que $I(A) = \frac{1}{2}$ por* **u***;*
- *Remova das regras restantes todos os literais default.*

Claramente, o programa resultante é não negativo, portanto ele tem um menor modelo trivalorado J. Seja $\Gamma(I) = J$. Os pontos fixos de Γ são definidos como modelos estáveis parciais trivalorados de um programa P.

Definição 3.45 (Semântica bem fundada) *[76] Uma interpretação trivalorada I de um programa normal em lógica P é chamada de modelo estável parcial trivalorado de P sss $\Gamma(I) = I$. O modelo bem fundado $\mathcal{M}_P$ de P é determinado pelo menor modelo estável parcial de P sob a ordem de Fitting*[10].

É garantido em [76] que todo programa possui pelo menos um modelo estável parcial trivalorado, logo é garantido que a semântica bem fundada está unicamente definida para todo programa.

Assim como a semântica dos modelos estáveis, a *WFS* atribui falso para o literal a no exemplo 3.39. No entanto, no exemplo 3.40, *WFS* atribui o valor indefinido para a ao passo que a semântica dos modelos estáveis não está definida para esse programa. Quanto ao exemplo 3.43, a *WFS* atribui o valor indefinido para os literais a, b e c. Esse resultado difere do apresentado pela semântica bem fundada, que possui dois modelos estáveis: $\{c, a\}$ e $\{c, b\}$.

3.3.3 Semânticas operacionais para programação normal em lógica

A regra da negação como falha possui uma característica bem procedural: um átomo A é falso num programa P se não houver nenhuma prova finita de A a partir de P. Essa

[10] *Para uma definição construtiva de WFS, o leitor pode recorrer a [69, 92].*

regra, foi implementada e adicionada à *SLD*, resultando na *SLDNF* (*Selection rule to Linear resolution for Definite programs with Negation as Failure*) [50]. A *SLDNF* é regra de derivação utilizada pelo *PROLOG*.

Na *SLDNF*, quando um literal positivo é selecionado, utiliza-se essencialmente a resolução *SLD* para derivar uma nova meta; todavia, quando um literal negativo *not* A é selecionando, uma tentativa é feita para construir uma árvore finitamente falha[11] tendo $\leftarrow A$ como raiz. Se tal árvore é encontrada, então *not* A é bem sucedido e é retirado da meta ao qual ele pertence. Em seguida uma nova meta (se houver) é selecionada e o processo continua. Por outro lado, se uma refutação *SLDNF* é encontrada para $\leftarrow A$, então *not* A falha.

É importante ressaltar que somente a derivação a partir de literais positivos pode gerar novas metas. Em se tratanto de literais negativos, é realizada uma avaliação para verificar se eles são bem sucedidos ou falhos. Conseqüentemente, a regra da negação como falha é simplesmente um teste. Em [50] é mostrado que *SLDNF* é correta e completa com relação à semântica completada de Clark aplicada a restrições de programas normais em lógica.

Na busca de semânticas operacionais mais expressivas, foi desenvolvida um procedimento de derivação para modelos bem fundados (derivação *WFM*) [72]. Esse procedimento é demonstrado ser correto e completo com relação a *WFM*. Ainda em [72], é mostrado um procedimento de derivação correto e completo com relação a semântica dos modelos estáveis trivalorados.

3.4 Programação em lógica estendida

Como mostrado na seção anterior, o operador negação default permite introduzir não somente informação negativa em programação em lógica, mas também formas não monotônicas de raciocínio. Na negação default, os literais são assumidos falsos quando não houver evidências para crer que eles sejam verdadeiros. Em muitos casos, isso é adequado para representar a informação negativa. Um exemplo clássico é o do banco de dados que contém as conexões de vôos existentes numa cidade. As conexões que estão explicitadas no banco de dados são as que existem realmente para essa cidade enquanto que a ausência de uma conexão no banco de dados indica implicitamente que ela não existe.

3.4.1 Importância do operador ¬

Apesar da importância do operador *not* no discurso natural e no raciocínio do senso comum, nem sempre é adequado representar a informação negativa de um modo implícito.

[11] Árvore constituída por finitos ramos falhos.

Assim, em vários trabalhos [63, 64, 29, 45, 30, 41, 43, 71, 73, 94, 95], tem sido argüído sobre as vantagens de estender a linguagem de programação em lógica para um segundo tipo de negação, denotado por $\neg$, que represente a informação negativa explícita. Limitando-se à negação default, a representação de alguns problemas configura-se pouco natural, pois não captura toda as conexões existentes entre um literal e sua negação.

Exemplo 3.46 Seja P o programa em lógica:

$não\_voa(x) \leftarrow pingüim(x)$
$voa(x) \leftarrow pássaro(x)$
$pássaro(x) \leftarrow pingüim(x)$
$pingüim(teobaldo)$

Claramente, está-se diante de uma contradição, pois pode-se concluir tanto que *teobaldo* voa quanto que não voa. No entanto, não há nenhuma relação entre essas conclusões em programação normal em lógica. Esse problema torna-se crítico quando a informação negativa for utilizada para representar exceções. A primeira regra de P pode ser vista como uma exceção à regra geral de que pássaros voam. Para poder caracterizá-la adequadamente, é necessário estabelecer uma conexão entre os literais *voa* e *não_voa*. Isso pode ser feito com a introdução do operador negação explícita $\neg$.

Também a diferença entre $not\ L$ e $\neg L$ na programação em lógica é essencial. Em muitas circunstâncias do dia-a-dia, não se pode assumir a negação de um literal L simplesmente quando não existe evidências para acreditar na verdade de L. Às vezes, é preciso certificar-se de sua negação:

"*Um ônibus pode cruzar uma ferrovia quando não houver um trem aproximando-se*".

Representando essa declaração como $cruzar \leftarrow not\ trem$ não estaria adequado, pois essa regra permite que o ônibus cruze a ferrovia mesmo que não haja informação sobre a ausência ou presença de um trem. Em algumas situações do mundo real, cruzar uma ferrovia nessas condições seria bastante arriscado, pois um trem pode estar vindo, mas não ser notado por conta de um denso nevoeiro por exemplo. Isso não ocorreria se a declaração acima fosse representada por $cruzar \leftarrow \neg trem$. Nesse caso, o ônibus somente cruzaria a ferrovia quando o motorista tiver certeza de que nenhum trem está aproximando-se.

Além disso, o uso combinado da negação explícita com a negação default em programação em lógica promove uma representação mais direta da informação, permitindo capturar uma ampla variedade de formas de raciocínio lógico. Assim a declaração

"*Se o motorista não tem certeza que o trem não está aproximando-se, então ele deveria esperar*" pode ser naturalmente representada por $esperar \leftarrow not\ \neg trem$.

Devido a simetria entre informação positiva e negativa em programação em lógica com negação explícita, determinadas informações ficam mais fáceis de serem representadas usando o seu complementar. Para clarificar essa idéia, observe o seguinte exemplo:

Exemplo 3.47 *[29]* Considere a descrição de um grafo a partir do predicado $arc(x, y)$, que expressa a existência no grafo de um arco do vértice x até o vértice y. Seja V o conjunto de vértices do grafo. Agora, suponha que se queira determinar quais vértices são terminais. A representação dessa informação torna-se bem mais fácil utilizando a negação explícita combinada com a negação default para estabelecer o complemento do conjunto de vértices de V que são não terminais.

$\neg terminal(x) \leftarrow arco(x, y)$
$terminal(x) \leftarrow not\ \neg terminal(x)$

Com a introdução da negação explícita em programação em lógica, uma classe bem mais profusa de formalismos de raciocínio não monotônico pode ser caracterizada. Em programação normal em lógica, até mesmo algumas simples regras default normais e seminormais não são passíveis de representação. Isso ocorre porque em programação normal em lógica, não é possível representar regras default com conclusões negativas ou com justificativas positivas como pode ser verificado nos seguintes casos:

$$\frac{a\ :\ b}{b} \qquad \frac{\neg a\ :\ \neg b}{c} \qquad \frac{a\ :\ \neg b}{\neg c}$$

No entanto, com a introdução da negação explícita, as regras default acima podem ser respectivamente caracterizadas como

$$b \leftarrow a, not\ \neg b \qquad c \leftarrow \neg a, not\ b \qquad \neg c \leftarrow a, not\ b$$

O operador $\neg$ junto com o operador *not* têm sido fundamentais para interligar a programação em lógica e raciocínio não monotônico. Como tiveram inicialmente desenvolvimentos independentes um do outro, essa interação tem fornecido novas abordagens para ambos os lados. Dessa forma, muitos formalismos em lógicas não monotônicas são mais fáceis de serem compreendidos quando representados em programação em lógica; por sua vez, a não monotonicidade proporciona novos paradigmas para a programação em lógica, que com isso, torna-se mais robusta e mais próxima do raciocínio do senso comum.

Espera-se que com essa motivação, tenha-se suficentemente ressaltado sobre as vantagens da inclusão da negação explícita em programação em lógica. Na tabela 3.1, é mostrado um sumário de tais vantagens:

Possibilita lidar com contradições
Representação adequada de exceções
Permite representar a informação negativa explícita
Uso combinado dos operadores *not* e $\neg$ aumenta poder de expressividade
Permite representar uma informação usando o seu complementar
Promove uma maior interligação com o raciocínio não monotônico

Tabela 3.1: Importância do operador $\neg$ em programação em lógica

3.4.2 Linguagem e semânticas de programas em lógica estendidos

O alfabeto $\mathcal{A}$ de um programa em lógica estendido segue da definição de alfabeto para programas normais em lógica acrescidos da negação explícita $\neg$ em sua classe de símbolos fixos. As definições de átomo e termo sobre $\mathcal{A}$ são análogas às apresentadas para programas em lógica definidos. Um literal pode ser objetivo ou default. Por sua vez, um literal objetivo é um átomo A ou sua negação explícita $\neg A$ enquanto um literal default é do tipo *not* L em que L é um literal objetivo. O símbolo $\neg$ é também usado para designar literais complementares. Assim $\neg\neg A = A$.

Definição 3.48 (Programa em lógica estendido) *Um programa em lógica estendido é um conjunto finito de regras da forma*

$$A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \qquad (m, n \geq 0),$$

em que A e cada B_i e C_j, com $0 \leq i \leq m$ e $0 \leq j \leq n$, são literais objetivos.

A linguagem $\mathcal{L}$ de um programa em lógica estendido P é constituída de todos as cláusulas que podem ser formadas a partir do alfabeto $\mathcal{A}$ sobre P. O universo de Herbrand estendido (resp. base de Herbrand estendida) de $\mathcal{A}$ a partir de um programa P é o conjunto de todos os termos básicos (resp. literais objetivos básicos) que podem ser construídos utilizando os símbolos de $\mathcal{A}$). Quando isto não causar ambigúidades, o universo de Herbrand estendido (resp. a base de Herbrand estendida) de $\mathcal{A}$ será simplesmente referido como o universo de Herbrand (resp. a base de Herbrand) e será denotado por $\mathcal{U}_P$ (resp $\mathcal{B}_P$). Uma interpretação é definida como para programas normais em lógica, porém usando a base de Herbrand estendida.

Dado que o conhecimento sobre um literal nem sempre é completo em programação em lógica, a fórmula $A \vee \neg A$ não é uma tautologia. Portanto não é aplicável o princípio do terceiro excluído e por conseqüência não vale o princípio da contraposição $((A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A))$. Por essa razão o termo negação clássica deve ser preterido, já que o

significado esperado de $\neg L$ é que L é explicitamente falso, ou seja, L é conhecido ser falso. Assim, semânticas que interpretam o operador $\neg$ como a negação clássica não capturam o significado esperado de um programa. Esse é o caso da semântica bem fundada com negação clássica [75].

A primeira semântica apresentada para programas em lógica estendidos foi a semântica dos conjuntos resposta [29]. Essa semântica é uma generalização da semântica dos modelos estáveis para a linguagem de programas estendidos. Basicamente, um conjunto resposta de um programa estendido P é um modelo estável do programa normal obtido de P substituindo cada literal objetivo da forma $\neg L$ por um átomo da forma $\neg\_L$.

Várias outras semânticas foram propostas para programas em lógica estendidos [45, 78, 26, 75]. Segundo Alferes [2], nenhuma delas captura o significado de programas em lógica estendidos. Para contrapor isso, é proposta em [68, 6], a *WFSX* (*Well-Founded Semantics for eXtended programs*). Essa semântica é uma generalização de *WFS* para programas em lógica estendidos. *WFSX* foi a primeira semântica baseada em modelos bem fundados a relacionar as duas formas de negação, explícita e default, através do princípio da coerência. Segundo esse princípio, um literal que é declaradamente considerado falso, deve ser assumido falso por default. De um modo sintático, tem-se $\neg L \Rightarrow not\ L$. Com isso, alguns resultados pouco intuitivos verificados em outras semânticas bem fundadas não são mais obtidos em *WFSX*.

Exemplo 3.49 [68] Seja P o programa em lógica

$a \leftarrow not\ b$

$b \leftarrow not\ a$

$\neg b$

De acordo com a *WFSX*, P tem como modelo

$$\{a, \neg b, not\ \neg a, not\ b\}.$$

Tais resultados não são obtidos por outras semânticas baseadas na *WFS* porque elas interpretam de forma independente os literais positivos e os seus complementos explícitos.

3.4.3 Paraconsistência na programação em lógica

Com a introdução do operador $\neg$, novas relações surgiram na programação em lógica. Em particular, como num programa em lógica podem coexistir um literal L e sua negação explícita $\neg L$, contradições podem ocorrer. Em [97] é observado que há três maneiras de uma semântica lidar com contradições:

- **Aproximação explosiva**: se o programa é contraditório, então qualquer literal pode ser derivado do programa;

- **Aproximação da revisão de crença**: se o programa é contraditório, é feita uma revisão para tentar recuperar a consistência;
- **Aproximação paraconsistente**: é possível extrair conclusões significativas mesmo estando diante de uma base de conhecimento contraditória.

A primeira aproximação somente é adequada para o raciocínio matemático; em situações da vida cotidiana, ao qual não se tem um total conhecimento sobre o mundo, uma aproximação explosiva pode conduzir a resultados indesejáveis:

Exemplo 3.50 Considere o programa abaixo:

$$P = \begin{cases} voa(x) \leftarrow alado(x) \\ \neg voa(x) \leftarrow mamífero(x) \\ \neg alado(x) \leftarrow mamífero(x) \\ voa(x) \leftarrow alado(x) \end{cases}$$

Ao acrescentar as informações de que *morcegos* são *alados* e são *mamíferos*, deriva-se $voa(morcego)$, $\neg voa(morcego)$, $alado(morcego)$ e $\neg alado(morcego)$. No exemplo acima, uma contradição surgiu por conta da falta de conhecimento sobre as espécies animais ao qual *morcego* é uma exceção à regra geral de que *mamíferos* não voam. Na aproximação explosiva, entretanto, é o suficiente para trivializar toda a base de conhecimento. Ademais, nenhuma informação adicional é dada pela semântica sobre os literais dependentes de contradição. Agora, imagine que o programa do exemplo 3.50 esteja incluído numa grande base de conhecimento. Não faz sentido essa contradição, que pode ter apenas uma repercussão local, afetar informações não relacionadas, inutilizando toda a base de conhecimento.

A segunda aproximação é mais adequada quando a informação contraditória advém de um erro; em contrapartida, quando a informação é por natureza contraditória, então nenhum tipo de correção deve ser feito e a terceira aproximação deve ser adotada. Observe, porém, que para revisar uma crença, é necessário antes detectar a contradição e todos os literais que dependem dessa contradição. Desse modo, o raciocínio paraconsistente pode ser visto como um passo intermediário na revisão de crença. Além disso, revisão de crença é normalmente intratável e despende grandes recursos computacionais. Por essa razão, semânticas paraconsistentes, que têm um desempenho computacional bem melhor, são mais utilizadas.

A manutenção da contradição pode ser importante para a representação do conhecimento porque permite que informações importantes sobre a propagação de contradição sejam capturadas. Obviamente, isso não deve implicar grandes custos computacionais.

Portanto, numa semântica paraconsistente, não basta detectar a informação contraditória, mas também a informação dependente de contradição e ambas devem conviver com o restante do programa sem interferências mútuas. Em função das relações que surgem envolvendo operadores e literais na programação em lógica com a introdução do ¬, muitas semânticas foram desenvolvidas. As principais são apresentadas no próximo capítulo.

Capítulo 4

Semânticas Paraconsistentes para Programas em Lógica

O raciocínio paraconsistente está presente em muitas situações do dia-a-dia, sendo de importância fundamental para a compreensáo dos processos cognitivos humanos. Apesar de sua intuitividade e ampla aceitação em Inteligência Artificial, somente a partir de [11, 63, 64, 62, 96, 97], o raciocínio paraconsistente passou a ser sistematicamente estudado no âmbito da programação em lógica.

Neste capítulo, são apresentadas algumas das principais semânticas paraconsistentes para programas em lógica e segue proximamente de [20, 21]. Por sua naturalidade e intuitividade, um especial destaque é dado para a $WFSX_P$, que é considerada uma das melhores semânticas para programas em lógica estendidos. Inicialmente, na seção 4.1, é mostrada uma semântica para programas Horn generalizados de Blair e Subrahmanian [11]. A seção 4.2 refere-se a uma extensão da *WFS* para permitir raciocínio paraconsistente proposta por Sakama [85]. Em seguida, são mostradas as semânticas de Wagner [96, 97]: raciocínio liberal, crédulo, conservativo e cético. Na seção 4.4, é apresentada a única das semânticas paraconsistentes exibidas neste capítulo que está de acordo com o princípio da coerência, a $WFSX_P$. A seção subseqüente trata de extensões da semântica dos conjuntos resposta [29] que adimitem paraconsistência. Nesse sentido, são apresentadas a semântica dos modelos estáveis paraconsistentes [85] e a semântica dos modelos semi-estáveis [86]. Para concluir o capítulo, na seção 4.6, é feita uma análise comparativa de algumas das características marcantes dessas semânticas.

4.1 Programas Horn generalizados de Blair e Subrahmanian

Blair e Subrahmanian foram um dos precursores no estudo de semânticas paraconsistentes na programação em lógica. Em [11], eles utilizam uma lógica isomórfica à lógica de Belnap [9] com quatro valores verdade: $V = \{\bot, \mathbf{f}, \mathbf{t}, \top\}$, que significam respectivamente indefinido, falso, verdadeiro e superdefinido. Nas definições e resultados que seguem, as discussões serão restritas, sem perda de generalidade, aos programas básicos.

Definição 4.1 (Programas Horn generalizados) *[11] Um programa Horn generalizado (GHP)*[1] *é um conjunto de cláusulas gh em que uma cláusula gh é da forma*

$$A_0 : \mu_0 \Leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n,$$

com $A_0, \ldots, A_n$ sendo literais básicos (clássicos) e $\mu_0, \ldots, \mu_n$, valores verdade (ou anotações) de V.

Da lógica de Belnap, os autores ainda definiram a ordem de conhecimento $\preceq_G$ (ou $\succeq_G$) em V como usual: $\bot \preceq_G \mathbf{f}, \mathbf{t} \preceq_G \top$ (ou $\top \succeq_G \mathbf{f}, \mathbf{t} \succeq_G \bot$).

De posse dessas informações, uma interpretação *GHP* é definida como uma função da base de Herbrand para o conjunto de valores verdade V, formando um reticulado completo sobre a ordem de conhecimento $\preceq_G$: $I_1 \preceq_G I_2$ sss para todo átomo A no domínio das interpretações, $I_1(A) \preceq_G I_2(A)$. Uma operação de negação nas anotações é também definida como $\neg(\mathbf{t}) = \mathbf{f}$, $\neg(\mathbf{f}) = \mathbf{t}$, $\neg(\bot) = \bot$ e $\neg(\top) = \top$.

Para definir uma semântica para *GHP*s, necessita-se apenas das noções de satisfação envolvendo literais anotados, cláusula gh e conjunção de literais anotados.

Definição 4.2 (Relação de satisfatibilidade para GHPs) *[11] Seja I uma interpretação GHP. $I \models_G F$ quer dizer que I satisfaz F e F_i é um literal anotado do tipo $A_i : \mu_i$. Uma interpretação I*

$\vdots$

(2) *satisfaz o átomo anotado fechado $A : \mu$ sss $I(A) \succeq_G \mu$;*
(3) *satisfaz o literal anotado fechado $\neg A : \mu$ sss I satisfaz $A : \neg\mu$;*

$\vdots$

(6) *satisfaz a fórmula fechada $F_0 \Leftarrow F_1 \& \ldots \& F_n$ sss $I \not\models_G F_1 \& \ldots \& F_n$ ou $I \models_G F_1$;*
(7) *satisfaz a fórmula fechada $F_1 \& \ldots \& F_n$ sss $I \models_G F_i$ para todo $i = 1, \ldots, n$;*

$\vdots$

[1] *Do inglês, Generalized Horn Programs.*

A semântica de um programa Horn generalizado G é caracterizada pelo seu menor modelo sob o ordenamento $\preceq_G$. Para determinar esse modelo, é feita uma generalização do operador conseqüência imediata T_P para *GHP*s.

Definição 4.3 (Operador conseqüência imediata para GHPs) *[11]Suponha que G seja um GHP. Então T_G^{GHP} é um mapeamento das interpretações GHP de Herbrand de G para interpretações GHP de Herbrand de G definido por*

$$T_G^{GHP}(I)(A) = lub\{\mu \mid A : \mu \Leftarrow B_1 : \mu \ \& \ \ldots \ \& \ B_k : \mu_k \text{ é uma instância básica de uma cláusula } gh \text{ em } G \text{ e } I \models_G B_1 : \mu_1 \& \ldots \& B_k : \mu_k\}.$$

Na seqüência, o menor modelo de um *GHP* G é relacionado ao menor ponto fixo de T_G^{GHP}.

Teorema 4.4 (Computação do menor modelo de um GHP) *[11] Qualquer GHP G possui um menor modelo $\mathcal{M}_G$. Em adição, esse menor modelo é idêntico ao menor ponto fixo de T_G^{GHP}. Além do mais, esse menor ponto fixo pode ser computado em no máximo ω passos.*

Exemplo 4.5 Seja G o seguinte *GHP* básico:

$$\begin{array}{ll} p(a) : \bot \Leftarrow q(b) : \mathbf{f} \ \& \ q(c) : \mathbf{t} & q(a) : \top \Leftarrow \\ p(b) : \mathbf{t} \Leftarrow q(b) : \mathbf{f} \ \& \ q(c) : \mathbf{t} & q(a) : \mathbf{t} \Leftarrow \\ p(b) : \mathbf{f} \Leftarrow q(b) : \mathbf{t} & q(b) : \mathbf{f} \Leftarrow \\ & q(c) : \mathbf{t} \Leftarrow \end{array}$$

Disso, obtém-se os resultados

$$\begin{array}{lcccccccc} & & p(a) & p(b) & p(c) & q(a) & q(b) & q(c) \\ T_G^{GHP\uparrow 0} & = & \bot & \bot & \bot & \bot & \bot & \bot \\ T_G^{GHP\uparrow 1} & = & \bot & \bot & \bot & \top & \mathbf{f} & \mathbf{t} \\ T_G^{GHP\uparrow 2} & = & \bot & \mathbf{t} & \bot & \top & \mathbf{f} & \mathbf{t} \\ T_G^{GHP\uparrow 3} & = & T_G^{\uparrow 2} & & & & & \end{array}$$

Para efeito de comparação, em [20], um *GHP* G é transformado num programa em lógica definido $G^{\neg}$ de um modo tal que é possível determinar o menor modelo de G através do menor modelo de $G^{\neg}$.

Caracteristicamente, a representação do conhecimento em *GHP*s configura-se pouco natural quando comparado com os programas em lógica estendidos. Por sua vez, como não possui negação default, também não é possível representar a informação incompleta, delimitando o seu baixo poder de expressividade. Além disso, o mecanismo operacional necessário para implementar a semântica de Blair e Subrahmanian apresentado em [11] é complexo e com resultados pífios relacionados à corretude e completude.

4.2 Semântica bem fundada estendida de Sakama

Em [78], a partir do trabalho de Gelfond e Lifschitz envolvendo semântica dos conjuntos resposta [29], Przymusinski definiu a semântica estável estendida, que introduz a negação explícita em semânticas bem fundadas. Nessa semântica, os literais negados explicitamente são vistos simplesmente como novos átomos. Entretanto, por definição, todos os modelos contraditórios[2] são descartados.

Posteriormente, essa restrição foi excluída por Sakama em sua definição de semântica bem fundada estendida [85], obtendo, assim, uma versão paraconsistente da *WFS* para programas em lógica estendidos (*EWFS*).

Ao caracterizar uma teoria de modelos para a *EWFS*, Sakama utiliza a lógica **VII**, introduzida por Ginsberg [32, 33] para representar hipóteses default. Para determinar os modelos da *EWFS*, é utilizada uma definição construtiva baseada no operador Θ [76].

Definição 4.6 (Operadores Φ_I e Ψ_I) *[85] Sejam P um programa (estendido) e $I = \sigma \cup not\ \delta$ uma interpretação. Para os conjuntos T e F de literais básicos, o mapeamento Φ_I e ψ_I são definidos como segue:*

$\Phi_I(T) = \{A \mid$ *há uma cláusula básica* $A \leftarrow B_1 \wedge \ldots B_m \wedge not\ C_1, \ldots, not\ C_n$ *de* P *tal que* $\forall_{B_i} B_i \in \sigma \cup T$ *e* $\forall_{C_j} C_j \in \delta\}$;

$\Psi_I(F) = \{A \mid$ *para toda cláusula básica* $A \leftarrow B_1 \wedge \ldots B_m \wedge not\ C_1, \ldots, not\ C_n$ *de* P, $\exists_{B_i} B_i \in \delta \cup F$ *ou* $\exists_{C_j} C_j \in \sigma\}$.

Sakama utiliza dois operadores para determinar os modelos bem fundados de programas em lógica estendidos: um para extrair as conclusões positivas e um outro para extrair as conclusões referentes aos literais default.

Definição 4.7 (Operadores T_I e F_I) *[85] Seja I uma interpretação. Então*

$$T_I^{\uparrow 0} = \{\} \qquad F_I^{\downarrow 0} = \mathcal{B}_P \cup \neg\mathcal{B}_P \text{ em que } \neg\mathcal{B}_P{}^3 = \{\neg A \mid A \in \mathcal{B}_P\}$$
$$T_I^{\uparrow n+1} = \Phi_I(T_I^{\uparrow n}) \qquad F_I^{\downarrow n+1} = \Psi_I(F_I^{\downarrow n})$$
$$T_I = \cup_{n<\omega} T_I^{\uparrow n} \qquad F_I = \cap_{n<\omega} F_I^{\downarrow n}$$

A próxima definição foi mostrada em [20], corrigindo um erro no trabalho original, já que para computar o menor ponto fixo do operador Θ abaixo, mais do que ω passos podem ser necessários.

[2]Nesta seção, modelos contraditórios são aqueles que simultaneamente contêm A e $\neg A$ para algum átomo A.

Definição 4.8 (Modelo bem fundado estendido) *[85] Para toda interpretação* $I = \sigma \cup not\ \delta$*, um operador* Θ *é definido por*

$$\begin{aligned} & \Theta(I) = (\sigma \cup T_I) \cup not\ (\delta \cup F_I); \\ I^{\uparrow 0} &= \{\}; \\ I^{\uparrow n+1} &= \Theta(I^{\uparrow n}),\ \textit{para um sucessor ordinal } n+1; \\ I^{\uparrow \alpha} &= \cup_{\beta<\alpha} I^{\uparrow \alpha},\ \textit{para um limite ordinal } \alpha \end{aligned}$$

O modelo bem fundado estendido M_P *é o menor ponto fixo de* Θ*, que é igual ao limite da seqüência acima.*

Exemplo 4.9 Considere o programa P abaixo:

$$P = \begin{cases} q \leftarrow not\ \neg p & p & \neg s \leftarrow not\ \neg s \\ r \leftarrow q & \neg p & \end{cases}$$

Pela definição acima, tem-se que

$$EWFM(P) = \{p, \neg p\} \cup not\ \{q, \neg q, r, \neg r, s\}.$$

Em [20], é demonstrado que considerando os literais negados explicitamente como átomos ordinários, *WFS* e *EWFS* apresentam os mesmos resultados. Isso é equivalente a retirar a restrição que exclui os modelos contraditórios na semântica de Przymusinski [78].

Para determinar se um literal depende ou não de contradição, ainda em [85], Sakama apresentou uma variante de sua semântica, que divide os literais em suspeitos e seguros. Um literal é suspeito se ele depende de informação contraditória e é seguro nos demais casos.

Os modelos bem fundados podem então ser computados mantendo um registro de todos os literais envolvidos na prova de cada literal. Sakama argumenta que essa informação somente é necessária para literais objetivos: o valor verdade de literais default numa interpretação nunca pode depender do sucesso de um par de literais contraditórios desde que todas as "provas" para esse literal falham[4].

Exemplo 4.10 [85] Seja P o seguinte programa

$$inocente \leftarrow \neg culpado \qquad \neg culpado \leftarrow acusado \wedge not\ culpado \qquad acusado$$

Então M_P é $\{acusado, inocente, \neg culpado\} \cup not\ \{culpado, \neg inocente, \neg acusado\}$. Se for adicionado a P o fato $\neg inocente$, o valor verdade de *inocente* torna-se contraditório enquanto que os valores verdade de *acusado* e $\neg culpado$ permanecem inalterados.

[4]Essa afirmação é contestada em [20], desde que a falha de literais default também pode ser devida ao par de literais contraditórios.

Por outro lado, em vez disso, se $\neg acusado$ e $homem$ são adicionados a P, o valor verdade de $acusado$ torna-se contraditório, o de $homem$, verdadeiro (seguro) e os valores verdade dos outros literais permanecem os mesmos. A verdade de $inocente$ e de $\neg culpado$ é agora menos crível desde que elas são derivadas do literal contraditório $acusado$. Com isso, $inocente$ é verdadeiro com suspeição e $culpado$ é falso com suspeição.

Desses resultados, conclui-se que a semântica bem fundada de Sakama não está de acordo com o princípio da coerência nem permite detectar todos os literais que dependem de informação contraditória. Além disso, observe que é possível atribuir nove combinações diferentes de valores lógicos aos pares de literais $(A, \neg A)$, correspondendo às seguintes interpretações em semânticas bem fundadas estendidas:

$$\{\}\ \{A\}\ \{\neg A\}\ \{A, \neg A\}\ \{notA\}\ \{A, not\ \neg A\}\ \{\neg A, not\ \neg A\}\ \ \{not\ A, not\ \neg A\}\ \{not\ \neg A\}.$$

Entretanto, na caracterização de uma teoria de modelos para $EWFS$, essas nove combinações estão dispostas numa lógica com sete valores de verdade: as interpretações $\{A\}$ e $\{\neg A\}$ na lógica **VII** são agrupadas respectivamente com $\{A, not\ \neg A\}$ e $\{\neg A, not\ A\}$, correpondendo respectivamente aos valores verdadeiro e falso. Isso parece pouco natural e resultados indesejáveis podem surgir:

Exemplo 4.11 *[20]* Considere os seguintes dois programas em lógica estendidos:

$$P_1 = \begin{cases} b \leftarrow not\ a \\ \neg a \end{cases} \qquad P_2 = \begin{cases} b \leftarrow not\ a \\ a \leftarrow not\ a \\ \neg a \end{cases}$$

Em P_1, a e b são atribuídos pela semântica bem fundada estendida de Sakama os valores lógicos falso e verdadeiro respectivamente. No segundo programa, a ainda continua com o valor verdade falso, mas "estranhamente", b passa a ser verdadeiro por default ($not\ \neg b \in EWFM(P_2)$ e $\{b, \neg b, \not\beta\{\not\subseteq EWFM(P))$. Semanticamente, nenhuma mudança nos valores lógicos deveria ter acontecido, já que a mantém o seu valor verdade.

4.3 Semânticas de Wagner

Em [97], Wagner propôs várias semânticas para programas em lógica estendidos com a negação forte de Nelson [61] sob a forma de bases de conhecimento constituídas por regras vívidas. Basicamente, tais sistemas correspondem aos programas em lógica estendidos sem negação default ou negação fraca pela denominação de Wagner. A linguagem desses programas é definida da seguinte maneira:

Definição 4.12 (Programas em lógica com negação forte) *[97] A linguagem de programas em lógica com negação forte (LPSN[5]) consiste dos operadores lógicos $\wedge$, $\vee$, $\sim$ e 1 denotando respectivamente a conjunção, disjunção, negação forte e o verum.*

Um programa em lógica com negação forte é um conjunto de cláusulas da forma $l \leftarrow F$, em que l é um literal, ou seja, um átomo a ou sua negação forte $\sim a$ e F é uma fórmula arbitrária.

Wagner assume que essa linguagem possui um número finito de átomos e não contém símbolos funcionais. Sem perda de generalidade, as discussões pertinentes a programas em lógica com negação forte serão restritas à instanciação básica desses programas. Além disso, é assumido que a premissa F de uma formúla $l \leftarrow F$ é uma conjunção de literais. Como mostrado em [97], todo programa em lógica com negação forte na sintaxe completa possui uma versão equivalente nesse formato mais restrito.

A semântica pretendida de programas em lógica com negação forte é obtida a partir da noção de modelos parciais:

Definição 4.13 (Interpretação parcial de Herbrand) *[97] Uma interpretação parcial de Herbrand é um par de conjuntos de átomos $\mathcal{M} = \langle M^t, M^f \rangle$, em que M^t contém os átomos verdadeiros e M^f, os átomos falsos. Desse conceito de intepretação parcial, as relações de modelo ($\models$) e de contramodelo ($=\!\!|$) podem ser indutivamente definidas como segue:*

$M \models a$ *sss* $a \in M^t$	$M =\!\!| \; a$ *sss* $a \in M^f$
$M \models F \wedge G$ *sss* $M \models F$ *e* $M \models G$	$M =\!\!| \; F \wedge G$ *sss* $M =\!\!| \; F$ *ou* $M =\!\!| \; G$
$M \models F \vee G$ *sss* $M \models F$ *ou* $M \models G$	$M =\!\!| \; F \vee G$ *sss* $M =\!\!| \; F$ *e* $M =\!\!| \; G$
$M \models \sim F$ *sss* $M =\!\!| \; F$	$M =\!\!| \; \sim F$ *sss* $M \models F$

Definição 4.14 (Modelo de um programa) *[97] Uma intepretação parcial $\mathcal{M}$ é um modelo de um programa em lógica Π com negação forte, simbolicamente $\mathcal{M} \models \Pi$, se para todo $l \leftarrow F \in \Pi$, $\mathcal{M} \models F$ implica $\mathcal{M} \models l$.*

Em seguida, Wagner mostrou que todo programa em lógica com negação forte possui um menor modelo $\mathcal{M}_\Pi$ de modo que uma fórmula F é conseqüência lógica de todo modelo de Π sss F também for conseqüência lógica de $\mathcal{M}_\Pi$. Além disso, um *LPSN* Π pode ser mapeado para um programa em lógica definido P, permitindo, assim, determinar $\mathcal{M}_\Pi$ através do menor modelo bivalorado do programa em lógica definido correspondente [20].

Por ser monotônico, um programa em lógica com negação forte carece de mecanismos não monotônicos para tornar a sua linguagem mais expressiva. Com esse objetivo,

[5] *Do inglês, Logic Programs with Strong Negation.*

Wagner introduziu o operador negação fraca "$-$". O significado desse operador é definido estendendo as relações de modelo e contramodelo induzidas por uma interpretação parcial.

Definição 4.15 (Relações de modelo para negação fraca) *[97] Seja $\mathcal{M}$ um interpretação parcial de Herbrand. As relações de modelo e contramodelo são estendidas com os dois casos adicionais:*

$$\mathcal{M} \models -F \textit{ sss } \mathcal{M} \not\models F \; \mathcal{M} \dashv\models -F \textit{ sss } \mathcal{M} \models F.$$

Essa interpretação para negação fraca guarda muitas das similaridades com a interpretação para a "negação por falha", isto é, $-F$ é verdadeiro se F não é conseqüência de uma dada interpretação parcial. Da forma como uma intepretação para a negação fraca foi definida, alguns problemas podem surgir. Primeiro, em toda interpretação parcial, tem-se que $\mathcal{M} \models a \vee -a$, ou seja, a negação fraca é bivalorada. Além do mais, é impossível ter simultaneamente a e -a verdadeiros, isto é, $\mathcal{M} \models -(a \wedge -a)$.

Definição 4.16 (Base de Conhecimento Vívido) *[97] Uma base de conhecimento vívido (VKB[6]) é um conjunto de cláusulas da forma $l \leftarrow F$, em que l é um átomo a ou sua negação forte $\sim a$ e F é uma fórmula arbitrária construída da linguagem definida a partir dos operadores lógicos $\wedge$, $\vee$, $\sim$, $-$ e 1.*

Nesta seção, a discussão será restrita a VKBs constituídos somente por regras da forma $l \leftarrow E$, em que E é uma conjunção de literais (átomos ou a negação de átomos). Esses VKBs foram denominados de bancos de dados dedutivos estendidos (XDB)s[7] [96, 97]. Como demonstrado por Wagner, sem perda de generalidade, toda VKB pode ser convertida num XDB. A noção de modelo para XDB permanece inalterada.

Exemplo 4.17 *[20]* Considere o XDB X contendo a única regra $a \leftarrow -a$. Esta base de conhecimento tem dois modelos: $\langle\{a\}, \{\}\rangle$ e $\langle\{a\}, \{a\}\rangle$. Em programação em lógica, esse resultado é considerado pouco intuitivo já que a é verdadeiro, mas não há suporte para a, pois $-a$ não é veiculado.

Para evitar problemas como esse, Wagner impôs restrições sintáticas sobre teorias tal que as conclusões pretendidas para XDB possam ser definidas pela teoria da prova. Nesse sentido, ele propôs quatro diferentes formas de raciocínio definidas através de quatro diferentes relações de inferência: liberal ($\vdash_l$), crédula ($\vdash_{cr}$), conservativa ($\vdash_c$), cética($\vdash_s$). Em comum, todos esses sistemas possuem as regras de inferência abaixo:

[6] *Do inglês, vivid knowledge base.*
[7] Do inglês, extended deductive databases.

$$
\begin{array}{lll}
(1) & X \vdash 1 & \\
(--) & X \vdash --l & \text{se} \quad X \vdash l \\
(\wedge) & X \vdash E & \text{se} \quad \forall e \in E : X \vdash e \\
(-\wedge) & X \vdash -E & \text{se} \quad \exists e \in E : X \vdash -e
\end{array}
$$

em que X é um XDB, E é um conjunto de literais e literais negados fracamente e l é um literal:

4.3.1 Raciocínio liberal

Para definir o raciocínio liberal, é adicionada ao sistema acima as seguintes regras de inferência:

$$
\begin{array}{lll}
(l) & X \vdash_l l & \text{se} \quad \exists (l \leftarrow E) \in X : X \vdash_l E \\
(-l) & X \vdash_l -l & \text{se} \quad \forall (l \leftarrow E) \in X : X \vdash_l -E
\end{array}
$$

Essa forma de raciocínio é chamada de liberal porque permite que num modelo M, l e $\tilde{l}$ (o complemento com relação a negação forte de l) possam ser simultaneamente verdadeiros ou falsos. Isso quer dizer que a verdade ou falsidade fraca de $\tilde{l}$ não afeta a verdade ou falsidade fraca de l. Entretanto, como mostrado no exemplo 4.18, algumas dessas derivações são infinitas:

Exemplo 4.18 Considere o XDB $X = \{a \leftarrow b \wedge c; b \leftarrow a; c \leftarrow 1\}$. Para X, tem-se as seguintes derivações infinitas:

$$
\begin{array}{cc}
X \vdash_l a & X \vdash_l -a \\
| & | \\
X \vdash_l b \wedge c & X \vdash_l -(b \wedge c) \\
| & | \\
X \vdash_l b \ e \ X \vdash_l c & X \vdash_l -b \ ou \ X \vdash_l -c \\
| & | \\
X \vdash_l a \ e \ X \vdash_l 1 & X \vdash_l -a \ ou \ X \vdash_l -1 \\
| & | \\
X \vdash_l a & X \vdash_l -a \\
\vdots & \vdots
\end{array}
$$

Para resolver esse problema e garantir que todo XDB possua um único modelo desejado, Wagner impôs algumas restrições sintáticas:

Definição 4.19 (XDB bem fundado) *[96, 97] Para um literal l e um XDB X, define-se $Pre^1(l)$, o conjunto dos predecessores num passo do literal l e $Pre(l)$, o conjunto de*

todos os literais precedendo l em X, como segue, em que "e" é um átomo ou o complemento com relação a negação forte de um átomo:

$$\begin{aligned} Pos(l) &= \{e \mid (l \leftarrow E) \in X \ e \ (e \in E)\} \\ Neg(l) &= \{e \mid (l \leftarrow E) \in X \ e \ (-e \in E)\} \\ Pre^1(l) &= Pos(l) \cup Neg(l) \\ Pre(l) &= Pre^1(l) \cup \bigcup \{Pre(k) \mid k \in Pre^1(l)\} \end{aligned}$$

A base dedutiva estendida X é bem fundada sss para todo literal l, tem-se que $l \notin Pre(l)$.

Em $Pos(l)$ e $Neg(l)$, são agrupados respectivamente os literais que estão envolvidos em laços positivos e negativos. Portanto, num *XDB* bem fundado, não pode haver nenhum desses laços. Em [20] é mostrado que um *XDB* X bem fundado pode ser prontamente mapeado para um programa normal em lógica, designado por P_X^l. Tais programas são localmente estratificados, portanto, P_X^l possui um único modelo perfeito (cf definição 4.21), que é equivalente ao modelo bem fundado de P_X^l [76].

Wagner ainda definiu uma noção menos restritiva, chamada de *XDB* bem fundada fracamente, que rejeita os laços negativos sob a negação fraca, mas permite laços positivos. Dessa noção, o autor apresenta o conceito de modelos perfeitos, que segue imediatamente de Przymusinski [79].

Definição 4.20 (XDB bem fundado fracamente) *[97] Os predecessores fracos de um literal l em um XDB X, $WPre(l)$, são definidos como segue:*

$$\begin{aligned} Pos(l) &= \{e \mid (l \leftarrow E) \in X \ e \ (e \in E)\} \\ Neg(l) &= \{e \mid (l \leftarrow E) \in X \ e \ (-e \in E)\} \\ WPre^1(l) &= Neg(l) \cup \bigcup \{Pre(k) \mid k \in Neg(l)\} \\ WPre(l) &= WPre^1(l) \cup \bigcup \{WPre(k) \mid k \in Pos(l)\} \end{aligned}$$

A base dedutiva estendida é bem fundada fracamente sss para todo literal l, tem-se que $l \notin WPre(l)$.

Definição 4.21 (Modelo perfeito de um XDB) *[97] Um modelo M' é dito ser preferível a M (com respeito a um dado XDB) se $\forall l \in M' - M$, $\exists k \in M - M' : k \in WPre(l)$. $\mathcal{M}$ é chamado de perfeito se não há modelos preferíveis a ele.*

Wagner então prova que todo *XDB* X bem fundado fracamente possui um único modelo perfeito, designado por $\mathcal{M}_X$, que é equivalente ao conjunto de literais deriváveis de X pelo raciocínio liberal. Um importante resultado demonstrado em [20] mapeia o modelo $\mathcal{M}_X$ de um *XDB* X fracamente bem fundado em *WFS*.

Exemplo 4.22 *[96]* Seja X o seguinte banco de dados dedutivo estendido:

$$X = \begin{cases} p \leftarrow 1 & q \leftarrow p & r \leftarrow p \\ \sim p \leftarrow 1 & \sim q \leftarrow 1 & \sim r \leftarrow q \end{cases}$$

$\mathcal{M}_X = \{p, \sim p, q, \sim q, r, \sim r\}$. Isso equivale a dizer que os literais $p, \sim p, q, \sim q, r$ e $\sim r$ são verdadeiros em X sob a ótica do raciocínio liberal.

Desse modo, a partir de uma lógica construtiva paraconsistente, Wagner chegou a uma semântica que é um caso restrito da semântica bem fundada estendida de Sakama, justificando formalmente muitas das idéias fundamentais das semânticas para programas em lógica estendidos. Entretanto, *XDB* não fracamente bem fundados ainda carecem de um embasamento lógico-matemático que justifique uma semântica trivalorada para programas em lógica estendidos. Uma sugestão de Wagner é a de estender a interpretação parcial com mais dois conjuntos: M^{df} e M^{dt}, em que $-a$ e $- \sim a$ são verdadeiros se respectivamente $a \in M^{df}$ e $a \in M^{dt}$. Assim, dada uma interpretação parcial $\langle M^t, M^f \rangle$ no sentido de Wagner, pode-se definir uma interpretação contituída pela quátupla $N = \langle N^t, N^f, N^{dt}, N^{df} \rangle$, com $N^t = M^t$, $N^f = M^f$, $N^{dt} = At - M^f$ e $N^{df} = At = M^t$, em que At é o conjunto de átomos da linguagem subjacente. A única restrição imposta por Wagner a essa quátupla foi para assegurar a satisfatibilidade default:

$$N^{df} \cup N^t = \{\} \text{ e } N^{dt} \cup N^f = \{\}.$$

Por outro lado, a coerência pode ser expressa pela condição

$$N^t \subseteq N^{dt} \text{ e } N^f \subseteq N^{df}.$$

Disso, resulta que para modelos satisfatíveis por default e coerentes, $N^t \cap N^f = \{\}$, evitando, assim, qualquer forma de paraconsistência.

Ao definir o raciocínio liberal, Wagner dá preferência a satisfatibilidade default em vez de dar preferência ao princípio da coerência. Isso pode fazer sentido para as semânticas de Wagner desde que não há necessidade de sobrecarregar literais indefinidos por coerência, pois não há literais indefinidos nessas semânticas. Entretanto, para semânticas bem fundadas trivaloradas como a $WFSX_P$, esse sobrecarregamento não é somente desejável como intuitivo (Cf exemplo 4.25). Em [20], é argumentado que a coerência é mais fundamental do que satisfatibilidade por default; afinal, se é permitido uma forma de paraconsistência sobre literais objetivos, não é justificável proibir paraconsistência sobre literais subjetivos. Assim, em semânticas paraconsistentes trivaloradas para programas em lógica estendidos, é preferível abdicar da satisfatibilidade por default e garantir a obediência ao princípio da coerência.

4.3.2 Raciocínio crédulo, conservativo e cético

Mesmo diante de uma base de informação contraditória, o raciocínio liberal é bastante permissível como mecanismo de inferência. Isso pode conduzir a resultados indesejados. Para manipular a informação mais cautelosamente, Wagner definiu algumas novas formas de raciocínio. A idéia fundametal entre elas é que uma conclusão deve ser aceita somente se ela é suportada e não duvidada. Dependendo das noções utilizadas de suporte e dúvida, diferentes formas de raciocínio podem ser definidas. Wagner propôs o raciocínio crédulo ($\vdash_{cr}$), conservativo ($\vdash_c$) e cético ($\vdash_s$).

Essas formas de raciocínio são definidas com base na teoria da prova, adicionando (l) e $(-l)$ às regras de inferência $(1), (--)$, $\wedge$ e $-\wedge$ (pag. 74):

Raciocínio crédulo
(l) $X \vdash_{cr} l$ sss $X \vdash_l l$ e $\forall(\tilde{l} \leftarrow E) \in X : X \vdash_{cr} -E$
$(-l)$ $X \vdash_{cr} -l$ sss $X \vdash_l -l$ ou $\exists(\tilde{l} \leftarrow E) \in X : X \vdash_{cr} E$

No raciocínio crédulo, uma conclusão l é aceita se ela é liberalmente inferida, mas não credulosamente duvidada, ou seja, quando $\tilde{l}$ não é derivada credulosamente.

Raciocínio conservativo
(l) $X \vdash_c l$ sss $\exists(l \leftarrow E) \in X : X \vdash_c E$ e $\forall(\tilde{l} \leftarrow F) \in X : X \vdash_c -F$
$(-l)$ $X \vdash_c -l$ sss $\forall(l \leftarrow E) \in X : X \vdash_c -E$ ou $\exists(\tilde{l} \leftarrow F) \in X : X \vdash_c F$

No raciocínio conservativo, é atribuído um mesmo peso para o suporte e a dúvida. Uma conclusão é aceita se ela é conservativamente inferida, mas não conservativamente duvidada.

Raciocínio cético
(l) $X \vdash_s l$ sss $\exists(l \leftarrow E) \in X : X \vdash_s E$ e $X \vdash_l -\tilde{l}$
$(-l)$ $X \vdash_s -l$ sss $\forall(l \leftarrow E) \in X : X \vdash_s -E$ ou $X \vdash_l \tilde{l}$

No raciocínio cético, uma conclusão l é aceita se ela é ceticamente inferida, mas não liberalmente duvidada.

Exemplo 4.23 Seja X o *XDB* do exemplo 4.22:

$$X = \begin{cases} p \leftarrow 1 & q \leftarrow p & r \leftarrow p \\ \sim p \leftarrow 1 & \sim q \leftarrow 1 & \sim r \leftarrow q \end{cases}$$

Pelo raciocínio crédulo, p e $\sim p$ não são inferidos porque mesmo sendo liberalmente derivados, não há respectivamente uma derivação crédula para a negação fraca do corpo de $\sim p \leftarrow 1$ e $p \leftarrow 1$. A mesma argumentação pode-se aplicar ao literal q. Por sua vez, $\sim q$ é credulamente derivada, pois $\sim q$ é liberalmente inferida e $-p$ é credulamente

inferida. Esse mesmo raciocínio se aplica a r e $\sim r$, portanto ambos literais também são derivados.

Com relação ao raciocínio conservativo, os literais p e $\sim p$ não são derivados porque não há respectivamente uma prova sob o raciocínio conservativo para a negação fraca do corpo de $\sim p \leftarrow 1$ e $p \leftarrow 1$. Disso resulta que $r, q, \sim r$ não são conservativamente derivados. Entretanto, o literal $\sim q$ é derivado sob esse raciocínio, já que existe a regra $\sim q \leftarrow 1$ em X para $\sim q$, com $X \vdash_c 1$ e na única regra para q, $q \leftarrow p$, $X \vdash_c -p$.

Pelo raciocínio cético, nenhum dos literais $p, \sim p, q, \sim q, r, \sim r$ são derivados porque há respectivamente uma derivação liberal para $\sim p, p, \sim q, q, \sim r, r$.

De acordo com a noção de modelo liberal de Wagner, no exemplo acima, o conjunto de fórmulas derivadas sob o raciocínio crédulo não constitui um modelo, pois nas regras $p \leftarrow 1$ e $\sim p \leftarrow 1$, observa-se um corpo verdadeiro com uma cabeça falsa.

Para resolver situações como essa, Wagner descarta todos os *XDB*s em que haja uma dependência de raciocínio de l sobre $\tilde{l}$ e vice-versa. A classe dos *XDB*s restantes é chamada de *XDB* bem fundados fortemente.

Definição 4.24 (XDB bem fundados fortemente) *[96] Seja X um XDB. X é dito ser bem fundado fortemente sss para todo literal l, $l \notin Pre(l)$ e $\tilde{l} \notin Pre(l)$.*

Cada uma das três formas de raciocínio apresentadas acima para um *XDB* bem fundado fortemente pode ser mapeada num programa normal em lógica sob a *WFS* [20]. Utilizando programas normais em lógica sob *WFS* permite atribuir semântica para *XDB* não bem fundados fortemente ou combinar os quatro tipos de raciocínio num único programa. Além disso, pode-se detectar os problemas referentes à recursão mútua através dos literais que são indefinidos no modelo bem fundado.

Para um *XDB* X sem negação fraca, as seguintes relações são válidas, em que l é um átomo ou a negação forte de um átomo:

$$X \vdash_s l \Rightarrow X \vdash_c l \Rightarrow X \vdash_{cr} l \Rightarrow X \vdash_l l.$$

Com a introdução da negação fraca, essas relações não são mais mantidas. Um importante exemplo foi dado por Teusink [89] para o $XDB = \{\sim p \leftarrow -p; p \leftarrow 1\}$, em que p é ceticamente, mas não conservativamente nem credulamente derivado. Entretanto, substituindo num *XDB* X todo literal negado fortemente $\sim a$ por um novo literal a^n, o raciocínio liberal pode ser indiferentemente implementado com elementos do raciocínio crédulo, conservativo ou cético [21]. Desse modo, pode-se concluir que essas quatro relações de inferência possuem o mesmo poder de expressividade.

4.4 Semântica bem fundada paraconsistente com negação explícita ($\mathbf{WFSX_P}$)

Nas semânticas citadas anteriormente, não é proporcionada nenhuma relação entre os operadores negação default (*not*) e negação explícita ($\neg$). Como conseqüência, resultados pouco intuitivos são obtidos:

Exemplo 4.25 Considere o programa abaixo:

$a \leftarrow not\ b \qquad \neg b$
$b \leftarrow not\ a$

Em semânticas que estão de acordo com o princípio da coerência, é possível de $\neg b$, concluir $not\ b$ e conseqüentemente a. No entanto, por interpretar os literais a e $\neg a$ independentemente, nas semânticas que não estão de acordo com o princípio da coerências, tais resultados não são obtidos.

Para sobrepujar problemas como esse, em [68, 6] é proposta a semântica bem fundada com negação explícita ($WFSX$). Posteriormente, essa semântica é generalizada para admitir paraconsistências ($WFSX_P$) [20]. Em ambas, está embutido o princípio da coerência, que apropriadamente estabelece uma relação entre essas duas formas de negação: $\neg L \Rightarrow not\ L$ (similarmente $L \Rightarrow not\ \neg L$). Em outras palavras, pelo princípio da coerência é garantido que se há razões para acreditar na verdade de $\neg L$, mais razões há para acreditar na verdade de $not\ L$.

Definição 4.26 *[28] Sejam P um programa em lógica estendida e I uma interpretação bivalorada. A transformação GL produz o programa reduzido $\frac{P}{I}$, que é obtido de P removendo todas as regras contendo um literal default not A tal que $A \in I$ e então removendo todos os literais default restantes de P. Por definição, $\Gamma_P I = M_{\frac{P}{I}}$.*

A adequação ao princípio da coerência é garantida através da versão seminormal de um programa em lógica estendido:

Definição 4.27 *[68, 6] A versão seminormal P_s de um programa P é obtida de P adicionando ao corpo (possivelmente vazio) de cada regra $L \leftarrow$ corpo o default not $\neg L$, em que $\neg L$ é o complemento de L com relação a negação explícita.*

Essa versão seminormal introduz um novo operador anti-monotônico: $\Gamma_{P_s}(S)$. Por simplicidade, será utilizado $\Gamma(S)$ para denotar $\Gamma_P(S)$ e $\Gamma_s(S)$ para denotar $\Gamma_{P_s}(S)$. Em [4], é demonstrado que o operador $\Gamma\Gamma_s$ é monotônico [4], portanto todo programa P possui um menor ponto fixo de $\Gamma\Gamma_s$. Esse ponto fixo é definido como o modelo bem fundado paraconsistente de P ($WFM_P(P)$).

Definição 4.28 *[4] Seja P um programa em lógica estendido e T um ponto fixo de $\Gamma\Gamma_s$, então $T \cup not\ (\mathcal{B}_P - \Gamma_s T)$ é um modelo estável parcial paraconsistente de P (PSM_P). O modelo bem fundado paraconsistente de P é o menor PSM_P sob a ordem da inclusão de conjuntos.*

Para obter uma definição construtiva para $WFM_P(P)$, define-se a seqüência transfinita abaixo:

$$\begin{aligned} I_0 &= \{\} \\ I_{\alpha+1} &= \Gamma\Gamma_s I_\alpha \\ I_\delta &= \bigcup\{I_\alpha \mid \alpha < \delta\} \text{ para um limite ordinal } \delta. \end{aligned}$$

Há um menor ordinal λ para a seqüência acima tal que I_λ é o menor ponto fixo de $\Gamma\Gamma_s$ e $WFM_P(P) = I_\lambda \cup not\ (\mathcal{B}_P - \Gamma_s I_\lambda)$.

Exemplo 4.29 Seja P o programa em lógica estendido:

$$P = \left\{ \begin{array}{lll} c \leftarrow not\ b & a & d \leftarrow not\ d \\ b \leftarrow a & \neg a & \end{array} \right\}$$

A seqüência para determinar o menor ponto fixo de $\Gamma\Gamma_s$ do programa P é
$I_0 = \{\}$
$I_1 = \Gamma\Gamma_s\{\} = \Gamma\{a, \neg a, b, c, d\} = \{a, \neg a, b\}$
$I_2 = \Gamma\Gamma_s\{a, \neg a.b\} = \Gamma\{d\} = \{a, \neg a, b, c\}$
$I_3 = \Gamma\Gamma_s\{a, \neg a, b, c\} = \Gamma\{d\} = I_2$

Conseqüentemente, $WFM_P(P) = \{a, \neg a, b, c\} \cup not\ \{a, \neg a, b, \neg b, c, \neg c, \neg d\}$.

Como discutido na sub-seção 4.3.1, em geral, semânticas paraconsistentes não podem ser simultaneamente coerentes e consistentes por default. Assim, por dar preferência ao princípio da coerência, a $WFSX_P$ não assegura a consistência por default. No exemplo 4.29, observe que os modelos de P contêm os literais $a, \neg a$ e b. Por coerência, obtém-se os literais $not\ \neg a, not\ a$ e $not\ \neg b$. De $not\ a$ e da regra $b \leftarrow a$, obtém-se que $not\ b$ e conseqüentemente c devem pertencer a $WFM_P(P)$. Do mesmo modo, com relação a $not\ c$, já que b é verdadeiro em $WFM_P(P)$. Portanto, o literal b e o literal c são simultaneamente verdadeiros e falsos em $WFM_P(P)$. Isso ocorre sempre que o princípio da coerência é aplicado em programas contraditórios. A partir dessa característica da $WFSX_P$, pode-se estabelecer quais os literais contraditórios e os dependentes de contradição num modelo WFM_P de um programa em lógica estendido P.:

- Um literal objetivo L é contraditório num modelo $WFM_P(P)$ sss L e $\neg L$ pertencem a $WFM_P(P)$;

- Um literal arbitrário L ou $not\ L$ num modelo $WFM_P(P)$ é dependente de literais contraditórios se L e $not\ L$ pertencem a $WFM_P(P)$ e L não é contraditório.

Em algumas situações, por causa do princípio da coerência, o valor verdade indefinido da cabeça de uma regra com corpo indefinido (**u**) é sobrecarregado com falso (**f**). Como resultado, obtém-se uma regra cuja cabeça possui o valor falso, o corpo permanece com valor verdade indefinido e mesmo assim, a regra ($\mathbf{f} \leftarrow \mathbf{u}$) é ainda satisfeita. Para semânticas incoerentes, resultados como esse são insatisfatíveis.

Exemplo 4.30 Considere o programa abaixo:

$b \leftarrow a \qquad \neg b$

$a \leftarrow not\ a$

De acordo com a definição 4.28, $WFM_P(P) = \{\neg b, not\ \neg a, not\ b\}$, ou seja, o valor indefinido de a na regra para b é sobrecarregado pela falsidade explícita de b. Por outro lado, em semânticas não coerentes como a *EWFS*, o menor modelo é constituído pelos literais $\{\neg b, not\ \neg a\}$.

Uma das peculiaridades da $WFSX_P$ é a obediência para o caso paraconsistente das propriedades estruturais apresentadas por Dix em [25, 24]. Além disso, para programas em lógica estendidos não contraditórios, a $WFSX_P$ coincide com a *WFSX* e para programas normais em lógica, a $WFSX_P$ coincide com a *WFS*.

4.5 Semânticas paraconsistentes baseadas em conjuntos resposta para programas em lógica estendidos

Uma outra linha de pesquisa focalizou-se no desenvolvimento de semânticas paraconsistentes baseadas na semântica dos conjuntos resposta [29]. Nesse sentido, vários trabalhos foram independentemente propostos: ambientes estáveis [74], conjuntos resposta fracos [62] e modelos estáveis paraconsistentes [86, 87]. Posteriormente, Sakama e Inoue definiram os modelos semi-estáveis, que atribuem significado para todo programa e correspondem aos modelos estáveis paraconsistentes quando esses últimos existem.

Em comum, todas essas semânticas são bivaloradas, ou seja, L ou $not\ L$ é conseqüência lógica de um programa P para qualquer literal $L \in \mathcal{B}_P$. Desse modo, segundo a terminologia de Wagner, a negação default *not* é interpretada como a negação fraca. Entretanto, nenhuma das semânticas apresentadas nessa seção são restritas à classe de programas localmente estratificados.

4.5.1 Modelos estáveis paraconsistentes

Em trabalhos independentes, várias propostas que generalizam a semântica dos conjuntos resposta para o caso paraconsistente foram apresentadas [74, 62, 86, 87]. Nesta seção, ao se referenciar a tais semânticas, será utilizado o termo modelo estável paraconsistente, pinçado de [86, 87]. Além disso, por simplicidade, a discussão será restrita aos programas não disjuntivos. Na seqüência, interpretação é definida como um conjunto arbitrário de literais objetivos.

Definição 4.31 (Modelos estáveis paraconsistentes) *[85] Sejam P um programa em lógica estendido e I uma interpretação. A redução de P com respeito a I é o programa em lógica estendido P^I tal que a regra $L_0 \leftarrow L_1, \ldots, L_m$ está em P^I sss há uma regra da forma*

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

de P tal que $\{L_{m+1}, \ldots, L_n\} \cap I = \{\}$. Então I é chamado de modelo estável paraconsistente (modelo p-estável) de P se I é o modelo minimal de P^I.

Utilizando a notação de Gelfond e Lifschitz [29], I é um modelo p-estável de um programa P sss $I = \Gamma_P I$. Dessa definição, conclui-se que modelos p-estáveis são conjuntos resposta que eventualmente podem ser contraditórios.

Exemplo 4.32 Considere o programa abaixo:

$$P = \begin{cases} q \leftarrow not\ \neg p & p \\ r \leftarrow q & \neg p \end{cases}$$

Esse programa possui o único modelo estável $\{p, \neg p\}$.

Em [87], Sakama e Inoue apresentaram uma relação entre os modelos estáveis paraconsistentes e a semântica dos conjuntos resposta. Para tanto, eles utilizam o programa P_{tr}, que é obtido de um programa em lógica estendido P adicionando um conjunto de axiomas da forma $N \leftarrow L, \neg L$, em que L e N são literais objetivos arbitrários. Esses axiomas extras representam uma maneira de simular a regra da trivialização da semântica dos conjuntos resposta. Os modelos p-estáveis de P_{tr} estão numa correspondência um para um com os conjuntos resposta de P.

Modelos p-estáveis também podem ser mapeados em *WFS* e portanto em $WFSX_P$ [20]. Entretanto, os modelos p-estáveis e $WFSX_P$ são em geral incomparáveis:

Exemplo 4.33 *[20]* Considere o programa em lógica estendido P:

$$a \leftarrow not\ b \quad b \leftarrow not\ a \quad \neg a$$

$WFM_P(P) = \{\neg a, b, not\ a, not\ \neg b\}$. Enquanto isso, P possui dois modelos p-estáveis: $M_1 = \{a, \neg a\}$ e $M_2 = \{\neg a, b\}$. M_1 não é comparável ao modelo bem fundado acima. Isso ocorre porque o princípio da coerência não é imposto por Sakama.

Apesar disso, a seguinte propriedade pode ser estabelecida:

Proposição 4.34 *[20] Seja P um programa em lógica estendido. Se I é um modelo p-estável de P, então há um modelo estável parcial paraconsistente de P contendo I.*

Exemplo 4.35 Considere o programa do exemplo 4.33. A interpretação $\{a, \neg a, b, not\ a, not\ \neg a, not\ b, not\ \neg b\}$ é um modelo estável parcial paraconsistente de P. Esse modelo contém ambos os modelo p-estáveis de P.

Os modelos p-estáveis apresentam alguns problemas. Um deles é o de não estarem definidos para todo programa. Por exemplo, o programa constituído somente pela regra $a \leftarrow not\ a$ não possui modelos p-estáveis. Além disso, essa semântica não está de acordo com as propriedades da cumulatividade, racionalidade e relevância [25, 24].

4.5.2 Modelos semi-estáveis

Em [87], Sakama e Inoue definem a noção de modelos semi-estáveis, que ao contrário dos modelos estáveis, atribuem significado para todo programa. Para isso ser alcançado, programas em lógica estendidos (disjuntivos) devem ser convertidos em programas em lógica positivos. Mais uma vez, para efeito de comparação, a discussão será restrita aos programas em lógica estendidos não disjuntivos.

Definição 4.36 (Transforamação Epistêmica) *[86] Seja P um programa em lógica estendido. Sua transformação epistêmica é definida como o programa em lógica estendido disjuntivo positivo P^k obtido de P trocando cada regra da forma*

$L_0 \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n\ (0 \leq m \leq n)$

pela seguinte regra sem o operador not em P^k:

$\lambda \vee \mathbf{K}L_{m+1} \vee ... \vee \mathbf{K}L_n \leftarrow L_1, ..., L_m$

$L_0 \leftarrow \lambda$

$\leftarrow \lambda, L_j\ (m+1 \leq j \leq n)$

Cada regra que já não possui o operador not em P é incluída em P^k sem alteração alguma.

Para cada regra disjuntiva em P^k, um diferente λ é associado. Os literais $\mathbf{K}L_j$ são novos átomos na linguagem de P^k e podem ser lidos como "L_j é acreditado". Conseqüentemente, *not* L_j tem o significado de "L_j não é acreditado". Esses literais $\mathbf{K}L_j$ podem ser compartilhados por diferentes regras do programa.

Exemplo 4.37 *[20]* Seja P o programa em lógica estendido do exemplo 4.33.

$$a \leftarrow not\ b \qquad b \leftarrow not\ a \qquad \neg a$$

O programa P^k correspondente a P é

$$\begin{array}{ll} \lambda_1 \vee \mathbf{K}b & \lambda_2 \vee \mathbf{K}a \\ a \leftarrow \lambda_1 & b \leftarrow \lambda_2 \\ \leftarrow \lambda_1, b & \leftarrow \lambda_2, a \\ \neg a & \end{array}$$

Uma semântica é atribuída para esses programas definindo o significado da disjunção na cabeça das regras e o das restrições de integridade. Para tanto, Sakama usa a noção de programa dividido:

Definição 4.38 (Modelos p-possíveis) *[86] Dado um programa disjuntivo estendido positivo P, um programa dividido é definido como o programa em lógica estendido positivo obtido de P trocando cada regra disjuntiva*

$$L_1 \vee ... \vee L_l \leftarrow L_{l+1}, ..., L_m$$

pelas seguintes regras estendidas (regras divididas):

$$L_i \leftarrow L_{l+1}, \ldots, L_m \text{ para todo } L_i \in S,$$

em que S é algum subconjunto arbitrário não vazio de $\{L_1, ..., L_l\}$ *para cada regra disjuntiva. Então, um modelo p-possível é definido como o (único) modelo p-estável de qualquer programa dividido P. Escolhas diferentes de S engendram todos os programas divididos.*

Exemplo 4.39 *[20]* O programa P^k do exemplo 4.37 possui nove programas divididos, mas somente cinco deles têm modelos p-estáveis. Os modelos p-possíveis correspondentes são

$p_1 = \{\lambda_1, a, \mathbf{K}a, \neg a\}$
$p_2 = \{\lambda_1, a, \mathbf{K}a, \mathbf{K}b, \neg a\}$
$p_3 = \{\mathbf{K}a, \mathbf{K}b, \neg a\}$
$p_4 = \{\lambda_2, b, \mathbf{K}b, \neg a\}$

$p_5 = \{\lambda_2, b, \mathbf{K}b, \mathbf{K}a, \neg a\}$

De posse desses p-modelos possíveis, o autor introduziu critérios para selecionar quais deles irão ser utilizados para definir os modelos semi-estáveis de um programa em lógica estendido.

Definição 4.40 (Modelos semi-estáveis) *[86] Seja P um programa em lógica estendido e $\mathcal{I}_{p^k}$ o conjunto de modelos p-possíveis minimais de P^k. Uma interpretação $I^k \in \mathcal{I}_{p^k}$ é dita ser maximalmente canônica sss não há interpretação $J^k \in \mathcal{I}_{p^k}$ tal que $\{\mathbf{K}L \mid \mathbf{K}L \in J^k$ e $L \notin J^k\} \subseteq \{\mathbf{K}L \mid \mathbf{K}L \in I^k$ e $L \notin I^k\}$. Os modelos semi-estáveis de P são as interpretações maximalmente canônicas em $\mathcal{I}_{p^k}$ com os literais λ_i removidos.*

Exemplo 4.41 Considere o programa do exemplo 4.39. Os modelos p-possíveis minimais de P^k são $p_1 = \{\lambda_1, a, \mathbf{K}a, \neg a\}$, $p_3 = \{\mathbf{K}a, \mathbf{K}b, \neg a\}$ e $p_4 = \{\lambda_2, b, \mathbf{K}b, \neg a\}$ Os modelos semi-estáveis são $\{a, \mathbf{K}a, \neg a\}$ e $\{b, \mathbf{K}b, \neg a\}$ correspondendo respectivamente aos modelos p-possíveis p_1 e p_4. Neste caso, os modelos semi-estáveis são isomórficos aos modelos p-estáveis do programa.

Sakama mostra em seu trabalho que os modelos p-estáveis são as interpretações canônicas de $\mathcal{I}_{p^k}$ (interpretações tal que para todo $\mathbf{K}L \in I_k$, $L \in I^k$ ou equivalentemente, para todo $L \notin I^k$, $\mathbf{K}L \notin I^k$). Suponha, por simplicidade, que *not* a é o único literal default no corpo de uma regra. A condição canônica garante que se a é falso, então $\mathbf{K}a$ é falso. Desse modo, se o corpo da regra é conseqüência do programa, a cabeça dessa regra também será, ou seja, *not* a é verdadeiro. Se a é verdadeiro, então as restrições de integridade não permitem que o literal λ seja verdadeiro, portanto a cabeça da regra não pode ser veiculada, mas $\mathbf{K}a$ pode, infligindo a condição canônica. Essencialmente, tem-se as duas condições abaixo:

- Se $a \notin I$, então *not* a é verdadeiro (Condição canônica);
- Se $a \in I$, então *not* a é falso (Restrição de integridade sobre literais λ).

A segunda condição sempre é verificada. No tocante a condição canônica, observe que ela não é satisfeita para programas sem modelos p-estáveis. Assim, para definir os modelos semi-estáveis, são aceitos como modelos interpretações que violam a condição canônica, mas preservam a segunda condição.

Como pode ser facilmente constatado, modelos semi-estáveis não estão de acordo com o princípio da coerência. Essa semântica também não obedece aos princípios da cumulatividade, racionalidade e relevância [25, 24]. Além do mais, nos modelos semi-estáveis, literais indefinidos recebem um tratamento pouco intuitivo.

Exemplo 4.42 Considere o programa em lógica estendido $\{b \leftarrow not\ d; c \leftarrow d; d \leftarrow not\ c\}$. Seu único modelo semi-estável é $\{b, not\ \neg b, not\ \neg c, not\ d, not\ \neg d\}$. Com isso, c é indefinido, mas $not\ d$ é verdadeiro e portanto b é também verdadeiro.

Observe que apenas parte dos literais envolvidos no laço negativo tornaram-se indefinidos nessa semântica. Como conseqüência, no exemplo acima, tem-se um valor indefinido implicando um falso, que é estranho para um programa normal em lógica.

Também se for adicionado ao programa acima a regra "$a \leftarrow b, not\ a$", haverá dois modelos semi-estáveis:$\{not\ \neg a, b, not\ \neg b, not\ \neg c, not\ d, not\ \neg d\}$ e $\{not\ a, not\ \neg a, not\ b, not\ \neg b, not\ \neg c, not\ \neg d\}$. Com isso, obtém-se um novo modelo em que d é também indefinido (além de c), por causa de uma regra não relacionada a d.

4.6 Discussão e conclusões

Costuma-se nominar Blair e Subrahmanian como os responsáveis pela introdução do raciocínio paraconsistente em programação em lógica ao apresentarem uma semântica para programas Horn generalizados [11]. Posteriormente, muitas semânticas paraconsistentes para programas em lógica foram definidas. Entretanto, com exceção dos raciocínios crédulo, conservativo e cético, todas as semânticas paraconsistentes apresentadas nesta seção atribuem o mesmo significado para programas em lógica estendidos sem negação default.

Um outro ponto de discussão refere-se à utilização da coerência ou da satisfação por default como princípio basilar. Na primeira classe, o princípio da coerência é utilizado como base para relacionar as duas formas de negação e permitir uma expansão localizada das conseqüências de programas contraditórios. Das semânticas apresentadas, apenas a $WFSX_P$ está de acordo com o princípio da coerência. Assim, nas demais, há uma total independência entre um átomo A e sua negação explícita $\neg A$.

Seguindo outra linha, foram definidas semânticas paraconsistentes baseadas em conjuntos resposta [86, 87], porém não atribuem significado para todo programa. Para resolver esse problema, Sakama definiu a semântica dos modelos semi-estáveis, que além de não obedecer ao princípio da coerência, apresenta um tratamento inadequado para os literais indefinidos.

Como demonstrado em [20, 21], para a maioria dos tipos de programas em lógica estendidos, existem transformações polinomiais para programas normais em lógica sob a WFS que preservam o significado original. Com isso, pode-se comparar aspectos envolvendo poder de expressão e complexidade a partir dos resultados relacionados aos programas normais em lógica. Além do mais, pode-se utilizar implementações de sistemas em programação em lógica normal para implementar essas semânticas paraconsistentes.

No que concerne ao poder expressivo de cada semântica, pode-se estabelecer a seguinte classe de equivalência por ordem crescente de expressividade:

- Programas Horn generalizados e programas em lógica com negação forte;
- Raciocínio liberal, crédulo, conservativo e cético;
- $WFSX_P$, semântica bem fundada estendida de Sakama;
- Modelos estáveis paraconsistentes e modelos semi-estáveis paraconsistentes.

Das semânticas paraconsistentes apresentadas, apenas a $WFSX_P$ e os modelos semi-estáveis estão definidos para todos os programas. Desse modo, por obedecer ao princípio da coerência, estar definida para todo programa e pelo fato de todas as outras semânticas baseadas em WFS serem facilmente simuladas em $WFSX_P$, a $WFSX_P$ é a candidata preferencial para ser a semântica paraconsistente utilizada em programas em lógica estendidos (não disjuntivos) [20].

Capítulo 5

Programação em Lógica Estendida da Inconsistência Epistêmica

Como já citado, a introdução da negação explícita em programas em lógica conduz muito naturalmente à utilização de formas de raciocínio paraconsistente. Várias semânticas foram propostas para tais programas (cf capítulo 4), sendo que $WFSX_P$ é considerada uma das mais adequadas [21]. No entanto, por interpretar regras com literais default no sentido de um default de Reiter [82], $WFSX_P$ apresenta algumas deficiências. Para sobrepujar essas deficiências, é proposto interpretar essas regras como defaults *IDL-LEI* [66, 67]. Nesse sentido, na seção 5.1 é definido o conceito de programas em lógica estendidos da inconsistência epistêmica. Em seguida, na seção 5.2, são apresentadas três definições equivalentes de semântica bem fundada da inconsistência epistêmica ($WFSX_{EI}$[1]).

Baseado nas ponderações de Marcelino Pequeno em [65], na seção 5.3, é argumentado que os problemas apresentados em $WFSX_P$ e na lógica default de Reiter ocorrem porque esses formalismos não estão de acordo com o princípio das exceções primeiro[2]. Assim, são elencados alguns problemas clássicos envolvendo raciocínio não monotônico e as soluções propostas em $WFSX_{EI}$ e em $WFSX_P$; um paralelo é traçado sobre como *IDL-LEI* e a lógica default de Reiter lidam com esses problemas. Por estarem de acordo com o princípio das exceções primeiro, *IDL-LEI* e $WFSX_{EI}$ apresentam soluções mais significativas. Em seguida, na seção 5.4, é demostrado que para qualquer programa em lógica da inconsistência epistêmica P, $WFSX_{EI}$ é correta com relação a intersecção de todas as extensões de uma teoria *IDL-LEI* correspondente a P. Fechando o capítulo, na seção 5.5, é demonstrada a natureza polinomial de $WFSX_{EI}$ para programas sem símbolos funcionais.

[1] Well-Founded Semantics for eXtended logic programs of Epistemic Inconsistency.

[2] Do inglês, exceptions-first principle.

5.1 Sintaxe de um programa em lógica estendido da inconsistência epistêmica

Seja $\mathcal{L}$ uma linguagem cujo *alfabeto* $\mathcal{A}$ é definido como uma classe disjunta dos seguintes elementos:

- **constantes:** $c_1, c_2...$;
- **variáveis:** $v_1, v_2...$;
- **símbolos predicados:** $p_1^n, p_2^m...$;
- **símbolos funcionais:** $f_1^n, f_2^m...$;
- **operadores:** $\leftarrow, not, \neg, ?$;
- **vírgula:** ",";
- **parênteses:** "(" e ")".

Além disso, é assumido que o conjunto de variáveis é infinito contável ao passo que os conjuntos de constantes, símbolos predicados e símbolos predicados são finitos ou infinitos contáveis. *Termos* e *átomos* sobre $\mathcal{A}$ são definidos como usualmente. Por sua vez, um *literal* sobre $\mathcal{A}$ pode ser objetivo, plausível ou default. Um literal *objetivo* é um átomo A ou sua negação explícita $\neg A$[3]; um literal *plausível* é da forma $L?$, em que L é um literal objetivo; finalmente, literais *defaults* são da forma *not* α, em que α é um literal objetivo ou plausível.

Uma *fórmula* sobre $\mathcal{A}$ é um literal objetivo ou plausível sobre $\mathcal{A}$, uma cláusula $\leftarrow \alpha_1, \ldots, \alpha_n$ ou uma cláusula do tipo $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$, em que α e α_i, $1 \leq i \leq n$, são literais objetivos ou plausíveis sobre $\mathcal{A}$. Dado esse alfabeto, uma *linguagem* $\mathcal{L}$ é o conjunto de todas as fórmulas sobre $\mathcal{A}$.

Por convenção, serão utilizadas letras gregas minúsculas para denotar indistintamente literais objetivos ou plausíveis em $\mathcal{L}$ enquanto letras romanas irão denotar literais sem "?". Em algumas situações, não será utilizada essa convenção. Nesses casos, eventuais ambigüidades serão dirimidas pelo contexto. Um programa em lógica estendido da inconsistência epistêmica é então definido da seguinte maneira:

Definição 5.1 *Um programa em lógica estendido da inconsistência epistêmica definido sobre uma linguagem $\mathcal{L}$ é um conjunto finito de regras fechadas universalmente da forma*

[3]Ressaltando que $\neg\neg A = A$.

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n\ (m, n \geq 0),$$

em que α, cada β_i, $1 \leq i \leq m$, e cada γ_j, $1 \leq j \leq n$, são literais objetivos ou plausíveis. O literal α representa a cabeça da regra enquanto que os demais literais constituem o corpo da regra. Se m e $n = 0$, então o símbolo "$\leftarrow$" é omitido e essa regra é chamada de fato.

Quando isto não causar ambigüidades, um programa em lógica estendido da inconsistência epistêmica será referido por programa em lógica da inconsistência epistêmica ou ainda simplesmente por programa.

Como estabelecido usualmente, as constantes, símbolos funcionais e símbolos predicados do alfabeto para um programa P são restritos àqueles que explicitamente ocorrem em P. Além disso, um termo ou um literal são chamados de *básicos* se eles não contêm variáveis. O *universo de Herbrand* é formado por todos os termos básicos que podem ser obtidos do alfabeto de P. O conjunto de todos os literais objetivos e plausíveis básicos que podem ser obtidos do alfabeto de P constitui a *base de Herbrand* de P, denotada por $\mathcal{B}_\mathcal{P}$. A *versão básica* de um programa P é o conjunto (possivelmente infinito) de todas as regras básicas obtidas de P substituindo cada uma das variáveis de P por elementos do seu universo de Herbrand. Por comodidade, em muitas circunstâncias, os literais presentes em tais programas serão denotados simplesmente por símbolos predicados.

Um programa em lógica da inconsistência epistêmica P precisa estar em consonância com certos princípios de *IDL-LEI*; caso não esteja, é necessário que P passe pelas transformações T_1 e T_2 abaixo. O programa resultante é denotado por P_{EI}.

- T_1 - **Transformação de um default genérico em um default IDL:** qualquer default *IDL-LEI* é do tipo normal ou seminormal com a conclusão sufixada por um "?". Por analogia, num programa em lógica da inconsistência epistêmica, qualquer regra da forma $L \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ ou da forma $L? \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ que tenha pelo menos um literal default em seu corpo $(n > 0)$ é substituída por $L? \leftarrow \beta_1, \ldots, \beta_n, not\ \gamma_1, \ldots, not\ \gamma_n, not\ \neg L$;

- T_2 - **Adequação de um programa à regra de inferência de LEI $\frac{(\alpha \rightarrow \beta)}{(\alpha? \rightarrow \beta?)}$:** Para toda regra $\alpha \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$, com $m, n \geq 0$, se $m = 1$, é adicionada a regra $\alpha? \leftarrow \beta_1?, not\ \gamma_1, \ldots, not\ \gamma_n$. Caso contrário, $\alpha? \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ é adicionado. Em particular, se L é um fato de P, então $L?$ é incluído no programa resultante P_{EI}.

Por essa regra de inferência de *LEI*, para cada regra $\alpha \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ em P, $\alpha? \leftarrow (\beta_1, \ldots, \beta_m)?, not\ \gamma_1, \ldots, not\ \gamma_n$ deveria ser adicionado a P_{EI}. Entretanto, tal regra não está definida na linguagem de um programa em lógica da inconsistência epistêmica, pois o operador "?" somente é aplicado em literais objetivos. Além disso,

$\alpha_1?, \ldots, \alpha_m? \Rightarrow (\alpha_1, \ldots, \alpha_m)?$ não é um teorema de *LEI*. Por tais motivos, apenas a versão restrita desse axioma de *LEI*, que é apresentada na transformação T_2, é aplicável num programa em lógica da inconsistência epistêmica.

Exemplo 5.2 Seja P o seguinte programa:

$a \leftarrow c, not\ b$
$b? \leftarrow d$
$\neg d \leftarrow e, \neg f$
$c? \leftarrow not\ a$

Aplicando as transformações definidas acima, obtém-se que

$$P_{EI} = \begin{cases} a? \leftarrow c, not\ b, not\ \neg a & a? \leftarrow c?, not\ b, not\ \neg a \\ b? \leftarrow d & b? \leftarrow d? \\ \neg d \leftarrow e, \neg f & \neg d? \leftarrow e, \neg f \\ c? \leftarrow not\ a, not\ \neg c & \end{cases}$$

Eventualmente, de um literal plausível $L?$ em P, obtém-se $L??$ em P_{EI}. Entretanto, $L?? \Rightarrow L?$ é um teorema de LEI. Assim, como pode ser verificado no exemplo 5.2, em situações como essa, $L??$ é substituído por $L?$.

Essas transformações podem ser compreendidas como uma forma de impor restrições sintáticas para ajustar um programa aos princípios de *IDL-LEI* acima mencionados. Para isso, parte-se da idéia que num programa em lógica da inconsistência epistêmica, o construtor do programa ao escrever uma regra da forma $A \leftarrow not\ B$, por exemplo, deve ter em mente uma regra da forma $A? \leftarrow not\ B, not\ \neg A$. Então, apesar de serem sintaticamente distintos, os programas P e P_{EI} possuem o mesmo significado. Na análise semântica que segue a partir da próxima seção, é assumido que um programa P já esteja no formato de P_{EI}. Por simplicidade, tais programas serão denotados apenas por P.

5.2 $WFSX_{EI}$ – Uma semântica bem fundada estendida da inconsistência epistêmica

Esta seção inicia com a definição de interpretação e modelo para programas em lógica da inconsistência epistêmica. Em seguida, é definido $WFSX_{EI}$(*Well-Founded Semantics for eXtended logic programs of Epistemic Inconsistency*). Trata-se de uma semântica trivalorada para programas em lógica da inconsistência epistêmica, que atribui um único significado para cada programa. Como em $WFSX_P$, o princípio da coerência também é válido para todas as conseqüências de um programa, inclusive àquelas que são contraditórias ou obtidas a partir de contradição.

Para tanto, são exibidas três definições equivalentes para $WFSX_{EI}$, que seguem de definições similares para $WFSX_P$ apresentadas por Damásio em [20]. Na subseção 5.2.2, $WFSX_{EI}$ é caracterizado com base numa divisão não determinística de programas; uma outra definição, baseada no operador de Przmusinski [80], é exibida na subseção 5.2.3; na subseção 5.2.4, $WFSX_{EI}$ é apresentado em termos de pontos fixos alternantes [92].

Vale ressaltar que, na seqüência abaixo, todos esses resultados são definidos para programas básicos. Por sua vez, esses programas podem ter uma quantidade de regras infinita, mas denumerável. Esse tipo de abordagem é comum no estudo de semânticas para programas em lógica [76] e não ocasiona perda de generalidade.

5.2.1 Interpretações e modelos

Uma interpretação para um programa em lógica da inconsistência epistêmica pode ser definida da seguinte maneira:

Definição 5.3 (Interpretação) *Uma interpretação I de uma linguagem $\mathcal{L}$ para um programa em lógica da inconsistência epistêmica é qualquer conjunto*

$$T \cup not\ F$$

tal que T e F são subconjuntos da base de Herbrand $\mathcal{B}_P$ e satisfazem as seguintes condições:

- *Se $L \in T$, então $L? \in T$ (Teorema de LEI)*
- *Se $L? \in F$, então $L \in F$ (Teorema de LEI)*
- $$\left.\begin{array}{l} \text{Se } \neg L \in T, \text{ então } L \in F \text{ e } L? \in F \\ \text{Se } \neg L? \in T, \text{ então } L \in F \end{array}\right\} \text{(Princípio da Coerência)}$$

Dessa definição pode-se intuitivamente afirmar que

- Um literal α é verdadeiro em I sss $\alpha \in I$;
- Um literal L é falso (por default) em I sss $not\ L \in I$;
- Um literal $L?$ é falso (por default) em I sss $not\ L \in I$ e $not\ L? \in I$[4];
- Um literal α é indefinido em I nos demais casos, ou seja, quando simultaneamente $\alpha \notin I$ e $not\ \alpha \notin I$.

Por seu turno, a formulação acima para o princípio da coerência é resultado do novo conceito de contradição com relação ao operador $\neg$ (contradição explícita) que segue da definição de interpretação contraditória para teorias *IDL-LEI*.

[4]De $not\ L? \in I$, tem-se que $not\ L \in I$, logo é suficiente considerar o caso em que $not\ L? \in I$.

Definição 5.4 (Interpretacões contraditórias) *Uma interpretação I para um programa em lógica da inconsistência epistêmica P é contraditória sss pelo menos um dos seguintes itens ocorrerem:*

- **Contradição forte:** $\{A, \neg A\} \subset I$;
- **Contradição fraca:** $\{A, \neg A?\} \subset I$ *ou* $\{A?\neg A\} \subset I$,

em que A é um átomo qualquer da linguagem de P.

Pelo princípio da coerência, se um literal $\neg\alpha$ é verdadeiro, os literais que entram em contradição explícita com $\neg\alpha$ devem ser falsos por default. Sabendo que na programação em lógica estendida não existe o conceito de contradição fraca, o princípio da coerência pode ser formulado como $\neg L \Rightarrow not\ L$.

Na programação em lógica da inconsistência epistêmica, entretanto, existe tanto a contradição forte quanto a contradição fraca. Com isso, o princípio da coerência deve ser formulado como segue, em que L é um literal objetivo:

- Os literais que entram em contradição explícita com $\neg L$ são L e $L?$. Portanto, se $\neg L$ é verdadeiro, então L e $L?$ são falsos por default. No entanto, como se $L?$ é falso por default, L é também falso por default, basta considerar o caso em que $L?$ é falso por default. Assim, se $\neg L \in T$, então $L? \in F$;
- O único literal que entra em contradição explícita com $\neg L?$ é L. Com isso, se $\neg L?$ é verdadeiro, então L é falso por default, ou seja, se $\neg L? \in T$, então $L \in F$[5].

Além da contradição explícita, uma outra classe de contradição pode ser encontrada quando um literal é simultaneamente verdadeiro e falso por default:

Definição 5.5 (Interpretações contraditórias por default) *Uma interpretação I para um programa em lógica da inconsistência epistêmica P é contraditória por default sss pelo menos um dos seguintes itens ocorrerem:*

- $\{A, not\ A\} \subset I$
- $\{A?, not\ A? \subset I\}$ *ou* $\{A, not\ A? \subset I\}$.

Em contraste, uma interpretação I é consistente por default sss I não é contraditória por default.

Conseqüentemente, para interpretações consistentes por default, as seguintes relações envolvendo o operador *not* são válidas:

[5]Naturalmente, se $L \in T$, então $\neg L? \in F$ e se $L? \in T$, então $\neg L \in F$.

- Se $not\ L \in I$, então $L \notin I$;
- Se $not\ L? \in I$, então $L \notin I$ e $L? \notin I$. Como $L \Rightarrow L?$, sabe-se que $L? \notin I \Rightarrow L \notin I$. Portanto é suficiente considerar o caso em que $L? \notin I$. Desse modo, se $not\ L? \in I$, então $L? \notin I$.

Sob a luz dessas considerações, é possível simular o comportamento de um default *IDL-LEI*. Assim, seja $\frac{\alpha : B; C}{B?}$ um default *IDL-LEI*. Sua representação num programa em lógica da inconsistência epistêmica é dada pela regra

$$B? \leftarrow \alpha, not\ \neg B, not\ \neg C?.$$

De um modo similar ao que ocorre com um default *IDL-LEI*, $\neg B? \in I$[6] não bloqueia a aplicação dessa regra; para tanto, é necessário que $\neg B \in I$. No entanto, $\neg C?$ é o suficiente para bloquear a sua aplicação.

Pela definição de interpretação aqui apresentada, um literal α e seu complemento explícito $\neg\alpha$ são independentes, exceto pela relação impetrada pelo princípio da coerência. Isso permite a existência de interpretações contraditórias. A presença de pares de literais contraditórios por default é também tolerada. Além do mais, não é exigido que numa interpretação I para um programa P, se $L \notin I$, então $not\ L \in I$ ou que se $L? \notin I$, então $not\ L? \in I$ para um literal objetivo $L \in \mathcal{B}_P$. Como conseqüência, o princípio do terceiro excluído e o princípio que da contradição, tudo se conclui não são obedecidos.

Como em [76], será apresentada uma definição equivalente de interpretação com base na relação $I : \mathcal{B}_P \rightarrow V$ definida abaixo para um programa em lógica da inconsistência epistêmica P, em que $\mathcal{B}_P$ é a base de Herbrand para P e $V = \{0, \frac{1}{2}, 1\}$[7].

Definição 5.6 *Qualquer interpretação $I = T \cup not\ F$ pode ser equivalentemente definida como uma relação $I : \mathcal{B}_P \rightarrow V$ tal que para todo $\alpha \in \mathcal{B}_P$,*

$$\begin{aligned} I(\alpha) &= 1 \qquad sss\ \alpha \in I \\ I(\alpha) &= 0 \qquad sss\ \ not\ \alpha \in I \\ I(\alpha) &= \frac{1}{2} \qquad nos\ demais\ casos \end{aligned}$$

Feitas essas observações, uma definição de valoração verdade para uma fórmula é, então, apresentada como segue:

Definição 5.7 (Valoração verdade) *Se I é uma interpretação, a valoração verdade $\hat{I}$ correspondente a I é a relação $\hat{I} : C \rightarrow V$, em que $V = \{0, \frac{1}{2}, 1\}$ e C é o conjunto de todas as fórmulas da linguagem. Para todo elemento de C, $\hat{I}$ é definida como segue:*

[6] Em que I é uma interpretação consistente por default.

[7] Como de praxe, $0, \frac{1}{2}, 1$ denotam respectivamente os valores lógicos falso, indefinido e verdadeiro.

1. *Se* α *é um literal objetivo ou plausível, então* $\hat{I}(\alpha) = I(\alpha)$;

2. *Para um literal default not* α, $\hat{I}(not\ \alpha) = 1 - \hat{I}(\alpha)$;

3. *Se* $\mathcal{R}$ *é uma fórmula do tipo* $\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$, *então*

$$\hat{I}(\mathcal{R}) = min(\hat{I}(\alpha_1), \ldots, \hat{I}(\alpha_m), \hat{I}(not\ \beta_1), \ldots, \hat{I}(not\ \beta_n));$$

4. *Se* L *é um literal objetivo e* $\mathcal{S}$ *é uma conjunção de literais, então*

$$\hat{I}(L \leftarrow \mathcal{S}) = \begin{cases} 1, & se\ \hat{I}(\mathcal{S}) \leq \hat{I}(L)\ ou\ \hat{I}(\neg L?) = 1\ e\ \hat{I}(\mathcal{S}) \neq 1 \\ 0, & nos\ demais\ casos \end{cases}$$

5. *Se* $L?$ *é um literal plausível e* $\mathcal{S}$ *é uma conjunção de literais, então*

$$\hat{I}(L? \leftarrow \mathcal{S}) = \begin{cases} 1, & se\ \hat{I}(\mathcal{S}) \leq \hat{I}(L?)\ ou\ \hat{I}(\neg L) = 1\ e\ \hat{I}(\mathcal{S}) \neq 1 \\ 0, & nos\ demais\ casos \end{cases}$$

6. $\hat{I}(\Box) = 0$

A condição $\hat{I}(\neg\alpha) = 1$ e $\hat{I}(\mathcal{S}) \neq 1$ segue da definição de *WFSX* [68, 6] e da nova interpretação para o operador *not* e é um reflexo do princípio da coerência. Disso resulta que a negação explícita $\neg$ sobrecarrega com falso o valor verdade indefinido das conclusões de regras com corpo indefinido. Observe ainda que se uma interpretação I não é contraditória nem contraditória por default, a relação de interpretação e de valoração verdade são funções, ou seja, um único valor verdade é atribuído a cada elemento de $\mathcal{B}_P$.

Definição 5.8 (Modelo) *Uma interpretação I é um modelo para um programa em lógica da inconsistência epistêmica P sss para toda instância básica* $\alpha \leftarrow \mathcal{S}$ *de toda regra de P,* $\hat{I}(\alpha \leftarrow \mathcal{S}) = 1$.

Exemplo 5.9 Seja P o programa abaixo:

$a?$
$b? \leftarrow not\ b, not\ \neg b$
$\neg a \leftarrow b$
$\neg a? \leftarrow b?$

Como exemplos de modelos para P, tem-se
$M_1 = \{a?\} \cup not\ \{\neg a\}$
$M_2 = \{a?, \neg b?\} \cup not\ \{\neg a, b\}$
$M_3 = \{a?, \neg a?, b?\} \cup not\ \{a, \neg a, b, \neg b\}$

Observe que M_1 não corresponde a um modelo no sentido clássico, pois o corpo da regra $\neg a \leftarrow b$ é indefinido enquanto a cabeça possui valor falso. Entretanto, como $a?$ é verdadeiro em M_1, na definição 5.7 de valoração verdade, a condição $\hat{I}(\neg L?) = 1$ e $\hat{I}(\mathcal{S}) \neq 1$ permite que $\neg a$ seja falso independentemente do valor indefinido de b em um modelo para P.

5.2.2 $WFSX_{EI}$ como uma divisão não determinística de programas

O significado pretendido para um programa em lógica definido P é representado pelo menor modelo bivalorado de P e pode ser adequadamente calculado pelo menor ponto fixo do operador conseqüência imediata T_P de Van Emden e Kowalski [91].

Em [76], é feita uma generalização do operador T_P para programas não negativos. Um programa não negativo P é aquele constituído somente por regras cujos literais em seus corpos são átomos ou a proposição $\mathbf{u}$. Por sua vez, $\mathbf{u}$ pode ser considerada como um átomo tal que para qualquer interpretação I, $\mathbf{u} \notin I$ e $not\ \mathbf{u} \notin I$. Em outras palavras, $\mathbf{u}$ possui valor verdade indefinido para qualquer interpretação I para P.

Ainda em [76], é provado que todo programa não negativo possui um menor modelo trivalorado $\mathcal{M}_P$ sob a ordem de verdade padrão. Também é definido um novo operador de conseqüência imediata, designado por Ψ^*_P, que é monotônico sob essa mesma ordem para programas não negativos consistentes por default[8]. Assim, para tais programas, Ψ^*_P sempre possui um menor ponto fixo sob a ordem de verdade padrão, que é demonstrado ser igual a $\mathcal{M}_P$ [76].

Devido à presença do operador "?" em programas em lógica da inconsistência epistêmica, é definido um operador conseqüência imediata similar a Ψ^*_P. Esse novo operador, designado por Ψ^{ei}_P, é aplicável somente a programas não negativos plausíveis. Ambos esses conceitos são exibidos abaixo:

Definição 5.10 (Programa não negativo plausível) *Um programa não negativo plausível P é um conjunto finito de cláusulas fechadas universalmente da forma*

$$\alpha \leftarrow \beta_1, ..., \beta_n,$$

com $n \geq 0$, α é um literal objetivo ou plausível e $\beta_1, ..., \beta_n$ são literais objetivos, literais plausíveis ou a proposição $\mathbf{u}$, em que $\mathbf{u}$ possui valor verdade indefinido para qualquer interpretação para P.

[8]Para um programa não negativo P, uma interpretação I para P é consistente por default sss A e $not\ A$ não simultaneamente pertencem a I para qualquer $A \in \mathcal{B}_P$.

Para programas não negativos plausíveis, é definido um tipo especial de interpretação e modelo que segue respectivamente das definições de interpretação e modelo para programas em lógica da inconsistência epistêmica, exceto que não é exigido que estejam de acordo com o princípio da coerência. Em contrapartida, é imposto que sejam consistentes por default. Para desanuviar qualquer confusão, tais interpretações e modelos serão referidas respectivamente como p-interpretações[9] e p-modelos[10].

As definições de p-interpretação contraditória permanece inalterada. Quanto a definição de valoração verdade para p-interpretações, ela é idêntica à definição 5.7, exceto que no item 4, é retirada a condição $\hat{I}(\neg\alpha) = 1$ e $\hat{I}(\mathcal{S}) \neq 1$, que está relacionada ao princípio da coerência. Além disso, por ser consistente por default, a relação de valoração verdade para p-interpretações é uma função.

Uma p-interpretação I é, portanto, um *p-modelo* de um programa não negativo plausível sss para toda instância $\alpha \leftarrow \mathcal{S}$ de P, $\hat{I}(\alpha \leftarrow \mathcal{S}) = 1$ segundo essa nova definição de valoração verdade. Baseado nisso, será provado que todo programa não negativo plausível possui um menor p-modelo trivalorado sob a ordem de verdade padrão. Antes, porém, é demonstrado o seguinte lema:

Lema 5.11 *Seja $\mathcal{I}_P$ o conjunto de p-interpretações para um programa não negativo plausível P e $M \subseteq \mathcal{I}_P$ tal que $M = \{M_i : M_i$ é um p-modelo para $P\}$ é o conjunto de todos os p-modelos para P com $M_i =< T_i, F_i >$ e $S = \{i : M_i \in M\}$. Então o maior limite inferior $\mathcal{M} =< \bigcap_{i\in S} T_i, \bigcup_{i\in S} F_i >$ de M é um p-modelo para P, logo $\mathcal{M} \in M$.*

(*Prova*)

Seja $\hat{\mathcal{M}} : C \rightarrow V$ a relação de valoração verdade correspondente a $\mathcal{M}$, em que C é o conjunto de todas as fórmulas da linguagem e $V = \{0, \frac{1}{2}, 1\}$. Suponha, por absurdo, que $\mathcal{M} = \langle \bigcap_{i\in S} T_i, \bigcup_{i\in S} F_i \rangle$ não seja um p-modelo para P. Então existe pelo menos uma cláusula $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ em P tal que $\hat{\mathcal{M}}(\alpha \leftarrow \alpha_1, \ldots, \alpha_n) = 0$. Assim, $\hat{\mathcal{M}}(\alpha) < \hat{\mathcal{M}}(\alpha_1, \ldots, \alpha_n)$. Há, portanto, três possibilidades de valoração:

$\hat{\mathcal{M}}(\alpha) = 0$ e $\hat{\mathcal{M}}((\alpha_1, \ldots, \alpha_n)) = 1$
$\hat{\mathcal{M}}(\alpha) = \frac{1}{2}$ e $\hat{\mathcal{M}}((\alpha_1, \ldots, \alpha_n)) = 1$
$\hat{\mathcal{M}}(\alpha) = 0$ e $\hat{\mathcal{M}}((\alpha_1, \ldots, \alpha_n)) = \frac{1}{2}$

Se $\hat{\mathcal{M}}((\alpha_1, \ldots, \alpha_n)) = 1$, então $\forall\alpha_j$ com $1 \leq j \leq n, \hat{\mathcal{M}}(\alpha_j) = 1$. Com isso, baseado na definição 5.6, $\{\alpha_1, \ldots, \alpha_n\} \subseteq \bigcap_{i\in S} T_i$. Desse modo, $\{\alpha_1, \ldots, \alpha_n\} \subseteq T_i$, $\forall i \in S$ ou alternativamente obtém-se que $\forall i \hat{M}_i((\alpha_1, \ldots, \alpha_n)) = 1$, em que $\hat{M}_i$ é a função de valoração verdade associada a M_i.

[9] De pseudo-interpretações.
[10] De pseudo-modelos.

Por essa hipótese, $\hat{\mathcal{M}}(\alpha) = 0$ ou $\hat{\mathcal{M}}(\alpha) = \frac{1}{2}$. Se $\hat{\mathcal{M}}(\alpha) = 0$, então $\alpha \in \bigcup_{i \in S} F_i$. Portanto existe $M_k \in M$ tal que $\alpha \in F_k$, ou seja, $\hat{M}_k(\alpha) = 0$. Como, por hipótese, $\hat{M}_k((\alpha_1, \ldots, \alpha_n)) = 1$, tem-se que $\hat{M}_k((\alpha \leftarrow \alpha_1, \ldots, \alpha_n)) = 0$. Isso é um absurdo, pois M_k é um p-modelo.

Por outro lado, se $\hat{\mathcal{M}}(\alpha) = \frac{1}{2}$, então $\alpha \notin \bigcap_{i \in S} T_i$ e $\alpha \notin \bigcup_{i \in S} F_i$. Isso quer dizer que existe $M_k \in M$ tal que $\alpha \notin T_k$ e $\alpha \notin F_k$, ou seja, $\hat{M}_k(\alpha) = \frac{1}{2}$. Com isso, tem-se que $\hat{M}_k(\alpha \leftarrow \alpha_1, \ldots, \alpha_n) = 0$. Novamente, está-se diante de um absurdo.

A última possibilidade é que $\hat{\mathcal{M}}((\alpha_1, \ldots, \alpha_n)) = \frac{1}{2}$ e $\hat{\mathcal{M}}(\alpha) = 0$. Assim, existe $1 \leq j \leq n$ tal que $\hat{\mathcal{M}}(\alpha_j) = \frac{1}{2}$ e para qualquer $1 \leq k \leq n$, $\hat{\mathcal{M}}(\alpha_k) \neq 0$. Desse modo, existe $1 \leq j \leq n$ tal que $\alpha_j \notin \bigcap_{i \in S} T_i$ e para qualquer $1 \leq k \leq n$, $\alpha_k \notin \bigcup_{i \in S} F_i$. Portanto existe $M_l \in M$ tal que $\alpha_j \notin T_l$ com $1 \leq j \leq n$ e para todo p-modelo $M_i \in M$, $\alpha_k \notin F_i$ para qualquer $1 \leq k \leq n$. Com isso, para qualquer $M_i \in M$, $\hat{M}_i((\alpha_1, \ldots, \alpha_n)) = 1$ se $\{\alpha_1, \ldots, \alpha_n\} \subseteq T_i$ e $\hat{M}_i((\alpha_1, \ldots, \alpha_n)) = \frac{1}{2}$ caso contrário, ou seja, se existe $1 \leq j \leq n$ tal que $\alpha_j \notin T_i$. Por hipótese, $\hat{\mathcal{M}}(\alpha) = 0$. Assim, tem-se que $\alpha \in \bigcup_{i \in S} F_i$, portanto existe $M_m \in M$ tal que $\alpha \in F_m$, ou seja, $\hat{M}_m(\alpha) = 0$. Como $\hat{M}_m((\alpha_1, \ldots, \alpha_n)) = 1$ ou $\hat{M}_m((\alpha_1, \ldots, \alpha_n)) = \frac{1}{2}$, $\hat{M}_m((\alpha \leftarrow \alpha_1, \ldots, \alpha_n)) = 0$ e um absurdo é encontrado.

Com isso, está provado que $\mathcal{M}$ é um p-modelo e conseqüentemente $\mathcal{M} \in M$. ◊

Teorema 5.12 (Existência de um menor p-modelo) *Todo programa não negativo plausível P possui um menor p-modelo sob a ordem de verdade padrão. Além disso, esse menor p-modelo para P é igual ao conjunto $\mathcal{M}$ definido acima.*

(*Prova*)

Inicialmente será provado que $\mathcal{M}$ é o menor p-modelo sob a ordem de verdade padrão. Assim, seja $M = \{M_i : M_i \text{ é um p-modelo para } P\}$ com $M_i =< T_i, F_i >$ e $S = \{i : M_i \in M\}$. Por absurdo, suponha que o maior limite inferior $\mathcal{M} = \langle \bigcap_{i \in S} T_i, \bigcup_{i \in S} F_i \rangle$ de M não seja o menor p-modelo para P sob a ordem de verdade padrão. Então, existe um p-modelo $\mathcal{M}' = \langle T', F' \rangle$ diferente de $\mathcal{M}$ tal que $\mathcal{M}' \leq_T \mathcal{M}$, ou seja, $T' \subseteq \bigcap_{i \in S} T_i$ e $\bigcup_{i \in S} F_i \subseteq F'$. Contudo, se $T' \subseteq \bigcap_{i \in S} T_i$, resulta que $T' = \bigcap_{i \in S} T_i$, pois $\mathcal{M}'$ é um p-modelo; similarmente, se $\bigcup_{i \in S} F_i \subseteq F'$, obtém-se que $\bigcup_{i \in S} F_i = F'$. Isso é um absurdo, pois é assumido que $\mathcal{M}$ é diferente de $\mathcal{M}'$. Portanto $\mathcal{M}$ é o (único) menor p-modelo sob a ordem de verdade padrão.

Ademais, como é garantido que M é não vazio já que pelo menos o p-modelo $\langle \mathcal{B}_P, \{\} \rangle$ deve estar presente, um menor p-modelo trivalorado $\mathcal{M}$ sempre existe para qualquer programa não negativo plausível. ◊

Para determinar os p-modelos para um programa não negativo plausível P, inicialmente, é apresentado o operador Ψ^{ei}_P. Em sua definição, buscou-se assegurar a leitura natural de como se conclui literais das cláusulas de P.

Definição 5.13 (O operador Ψ_P^{ei}) *Sejam P um programa não negativo plausível e $I = T \cup not\ F$, uma p-interpretação para P com α e todos os α_is sendo átomos básicos ou átomos plausíveis*[11] *básico. $\Psi_P^{ei}(I)$ é o conjunto definido como segue:*

- *$\alpha \in \Psi_P^{ei}(I)$ sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ em P tal que $\alpha_i \in I$ para todo $i \leq n$.*
- *not $\alpha \in \Psi_P^{ei}(I)$ sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ em P, existe um $i \leq n$ tal que not $\alpha_i \in I$ (vacuosamente verdadeiro se não há regra para α em P).*

Exemplo 5.14 Sejam P o programa não negativo plausível abaixo e $I = \{a, a?, \neg c?\} \cup not\ \{\neg a, \neg a?, \neg b, \neg c\}$.

$$P = \begin{cases} b \leftarrow a \\ b? \leftarrow a? \\ \neg c? \leftarrow b? \\ d? \leftarrow \mathbf{u} \end{cases}$$

Pela definição 5.13, $\Psi_P^{ei}(I) = \{b, b?\} \cup not\{a, \neg a, a?, \neg a?, \neg b, \neg b?, c, \neg c, c?, \neg c?, d, \neg d, \neg d?\}$.

No próximo teorema, os p-modelos para um programa não negativo plausível são caracterizados através do operador $\Psi_P^{ei}(I)$:

Teorema 5.15 (Caracterização de $\Psi_P^{ei}(I)$) *Uma p-interpretação I é um modelo para um programa não negativo plausível P sss $\Psi_P^{ei}(I) \preceq_T I$.*

(*Prova*)

Sejam $I = T_I \cup not\ F_I$ e $\Psi_P^{ei}(I) = T_\Psi \cup not\ F_\Psi$. A prova desse teorema será feita demonstrando que $\Psi_P^{ei}(I) \not\preceq_T I$ sss I não é um modelo para P. Por definição, $\Psi_P^{ei}(I) \not\preceq_T I$ sss $T_\Psi \not\subseteq T_I$ ou $F_I \not\subseteq F_\Psi$.

Suponha que $T_\Psi \not\subseteq T_I$. Então, tem-se que $T_\Psi \not\subseteq T_I$ sss existe $\alpha \in T_\Psi$ e $\alpha \notin T_I$ sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ tal que para todo $1 \leq i \leq n$, $\alpha_i \in I$ e $\alpha \notin I$ sss I não é um p-modelo.

Por outro lado, suponha que $F_I \not\subseteq F_\Psi$. Então, tem-se que $F_I \not\subseteq F_\Psi$ sss existe $\alpha \in F_I$ e $\alpha \notin F_\Psi$ sss $\alpha \in F_I$ e existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ tal que para todo $1 \leq i \leq n$, $not\ \alpha_i \notin I$ ou α_i é $\mathbf{u}$ sss $not\ \alpha \in I$ e $\hat{I}(\{\alpha_1, \ldots, \alpha_n\}) \geq \frac{1}{2}$ sss $\alpha \notin I$, já que I é consistente por default, e $\hat{I}(\{\alpha_1, \ldots, \alpha_n\}) \geq \frac{1}{2}$ sss $\hat{I}(\alpha) = 0$ e $\hat{I}(\{\alpha_1, \ldots, \alpha_n\}) \geq \frac{1}{2}$ sss I não é um p-modelo.

Disso resulta que I é um p-modelo para P sss $\Psi_P^{ei}(I) \preceq_T I$. ◊

[11] *Um átomo plausível é do tipo A?, em que A é um átomo.*

Teorema 5.16 (Monotonicidade de Ψ_P^{ei} sob a ordem de verdade padrão) *Sejam P um programa não negativo plausível e I_1 e I_2 duas p-interpretações para P. Se $I_1 \preceq_T I_2$, então $\Psi_P^{ei}(I_1) \preceq_T \Psi_P^{ei}(I_2)$.*

(*Prova*)

Sejam $I_1 = T_1 \cup not\ F_1$ e $I_2 = T_2 \cup not\ F_2$. Como, por hipótese, $I_1 \preceq_T I_2$, tem-se que $T_1 \subseteq T_2$ e $F_2 \subseteq F_1$.

Suponha que $\alpha \in \Psi_P^{ei}(I_1)$, em que α é um literal objetivo ou plausível. Pela definição 5.13, existe uma regra $\alpha \leftarrow \alpha_1, ..., \alpha_n$ em P tal que $\alpha_i \in T_1$ para todo $i \leq n$. Como $T_1 \subseteq T_2$, tem-se que $\alpha_i \in T_2$ para todo $i \leq n$. Desse modo, $\alpha \in \Psi_P^{ei}(I_2)$.

Agora, suponha que $not\ \alpha \in \Psi_P^{ei}(I_2)$. Pela definição 5.13, para toda regra $\alpha \leftarrow \alpha_1, ..., \alpha_n$ em P, existe $i \leq n$, tal que $not\ \alpha_i \in F_2$. Como $F_2 \subseteq F_1$, existe $i \leq n$, tal que $not\ \alpha_i \in F_1$. Conseqüentemente, $not\ \alpha \in \Psi_P^{ei}(I_1)$. ◊

Teorema 5.17 (Ψ_P^{ei} é finitário sob a ordem de verdade padrão) *Seja P um programa não negativo plausível. Para todo conjunto direto Y sob a ordem de verdade padrão,*

$$\Psi_P^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right) \preceq_T \bigsqcup_{y\in Y}^{T} \Psi_P^{ei}(y).$$

(*Prova*)

Seja α um literal objetivo ou plausível. Se $\alpha \in \Psi_P^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right)$, então há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em P tal que para todo $i \leq m$, $\alpha_i \in \bigsqcup_{y\in Y}^{T} y$. Como Y é um conjunto direto sob a ordem de verdade padrão, há um $y' \in Y$ tal que para todo $i \leq m$, $\alpha_i \in y'$. Conseqüentemente, $\alpha \in \bigsqcup_{y\in Y}^{T} \Psi_P^{ei}(y)$.

Com relação a um literal default $not\ \alpha$, deve-se provar que

$$not\ \alpha \in \bigsqcup_{y\in Y}^{T} \Psi_P^{ei}(y) \Rightarrow not\ \alpha \in \Psi_P^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right).$$

Suponha que $not\ \alpha \in \bigsqcup_{y\in Y}^{T} \Psi_P^{ei}(y)$ e por absurdo, $not\ \alpha \notin \Psi_P^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right)$. Se $not\ \alpha \notin \Psi_P^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right)$, então existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em P tal que para todo $i \leq m$, $not\ \alpha_i \notin \bigsqcup_{y\in Y}^{T} y$. Como Y é um conjunto direto, existe $y' \in Y$ tal que para todo $i \leq m$, $not\ \alpha_i \notin y'$. Com isso, obtém-se que $not\ \alpha \notin \Psi_P^{ei}(y')$. Portanto, $not\ \alpha \notin \bigsqcup_{y\in Y}^{T} \Psi_P^{ei}(y)$,

que é um absurdo, pois por hipótese, $not\ \alpha \in \bigsqcup_{y\in Y}{}_T\ \Psi_P^{ei}(y)$. Conseqüentemente, $not\ \alpha \in \Psi_P^{ei}\left(\bigsqcup_{y\in Y}{}_T\ y\right)$. ◊

Teorema 5.18 (Menor p-modelo trivalorado) *O menor p-modelo trivalorado sob a ordem de verdade padrão de um programa não negativo plausível unicamente existe e é definido por* $least(P) = \Psi_P^{ei\uparrow\omega}$, *em que*

$$\begin{aligned}\Psi_P^{ei\uparrow 0} &= not\ \mathcal{B}_P\\ \Psi_P^{ei\uparrow n+1} &= \Psi_P^{ei}(\Psi_P^{ei\uparrow n})\\ \Psi_P^{ei\uparrow\omega} &= \bigsqcup_{n<\omega}{}_T\ \Psi_P^{ei\uparrow n}\end{aligned}$$

(*Prova*)

Esse resultado segue diretamente dos teoremas 5.12 e 5.15 e da prova que o operador Ψ^{ei} é monotônico (teorema 5.16) e finitário (teorema 5.17). ◊

Exemplo 5.19 Seja P o seguinte programa não negativo:

$$P = \begin{cases} b \leftarrow \neg d? & b? \leftarrow \neg d?\\ c \leftarrow a?, b & c? \leftarrow a?, b\\ a? \leftarrow \mathbf{u} & \neg d?\end{cases}$$

O menor p-modelo trivalorado para P pode então ser calculado:

$\Psi_P^{ei\uparrow 0} = not\ \{a, \neg a, a?, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, c?, \neg c?, d, \neg d, d?, \neg d?\}$
$\Psi_P^{ei\uparrow 1} = \{\neg d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, c?, \neg c?, d, \neg d, d?\}$
$\Psi_P^{ei\uparrow 2} = \{b, b?, \neg d?\} \cup not\ \{a, \neg a, \neg a?, \neg b, \neg b?, c, \neg c, c?, \neg c?, d, \neg d, d?\}$
$\Psi_P^{ei\uparrow 3} = \{b, b?, \neg d?\} \cup not\ \{a, \neg a, \neg a?, \neg b, \neg b?, \neg c, \neg c?, d, \neg d, d?\}$
$\Psi_P^{ei\uparrow\omega} = \Psi_P^{ei\uparrow 4} = \Psi_P^{ei\uparrow 3}$

Assim, $least(P) = \Psi_P^{ei\uparrow\omega} = \{b, b?, \neg d?\} \cup not\ \{a, \neg a, \neg a?, \neg b, \neg b?, \neg c, \neg c?, d, \neg d, d?\}$, ou seja, os literais $b, b?$ e $\neg d?$ são verdadeiros; $a?, c$ e $c?$ são indefinidos e os restantes são falsos em $least(P)$.

Dos teoremas referentes a pontos fixos, sabe-se que $lfp(T) = glb(x : T(x) \leq x)$[12], em que T é um reticulado completo e $\leq$ é uma relação de ordem parcial qualquer. Conseqüentemente, $least(P) = glb\{I : \Psi_P^{ei}(I) \leq_T I\}$. Do teorema 5.15, obtém-se que

[12] Confira proposição 2.80 no capítulo 2.

$least(P) = glb\{M' : M'$ é um p-modelo para $P\}$, que corresponde a definição apresentada no lema 5.11 e no teorema 5.12.

Como Ψ^{ei} somente é aplicável aos programas não negativos plausíveis e não está de acordo com o princípio da coerência, ele não pode ser diretamente usado para calcular o menor modelo de um programa em lógica da inconsistência epistêmica P. Para tanto, é necessário que P seja convertido num programa não negativo plausível e algum mecanismo seja introduzido para impor o princípio da coerência.

Para efetuar essa conversão, deve-se avaliar todos os literais default de um programa em lógica da inconsistência epistêmica P segundo uma determinada interpretação. Quando isso ocorre, diz-se que o programa resultante sofreu uma divisão por uma interpretação. Esse conceito foi inicialmente apresentado por Gelfond e Lifschitz na definição de semântica dos modelos estáveis [28] e na definição do operador de Gelfond-Lifschitz estendido Γ^* [76]. Uma versão para programas estendidos é mostrada em [68], sendo generalizada em [2, 6] para admitir contradições. Em programas em lógica da inconsistência epistêmica, será feita uma generalização dessa última versão para também admitir literais com "?". Como será exemplificado posteriormente, antes de fazer essa divisão, é exigido por causa do princípio da coerência que os programas em lógica da inconsistência epistêmica estejam na forma canônica:

Definição 5.20 (Forma canônica) *[68, 2, 6] Seja P um programa em lógica da inconsistência epistêmica. A sua versão na forma canônica é obtida de P acrescentando em cada regra R de P um literal default not $\neg L?$ para todo literal objetivo L e um literal default not $\neg L$ para todo literal plausível $L?$ que ocorram no corpo de R.*

Sem perda de generalidade, um programa em lógica da inconsistência epistêmica P pode ser transladado para a sua versão na forma canônica P^c. Como pode ser conferido no teorema 5.57, P e P^c possuem os mesmos modelos bem fundados da inconsistência epistêmica. Intuitivamente, isso pode ser explicado como segue, assumindo que as valorações dos literais estão referenciadas a uma interpretação I para P em cada item abaixo:

- Se um literal L (resp. $L?$) é verdadeiro, então, por coerência, *not* $\neg L?$ (resp. *not* $\neg L$) também é verdadeiro. Assim, o valor verdade da versão canônica dessa regra não é alterado;
- Se L (resp. $L?$) é falso, então o corpo será falso e o seu valor verdade não será alterado com a inclusão de *not* $\neg L?$ (resp. *not* $\neg L$);
- Se L é indefinido, as situações abaixo podem ocorrer:
  - Se $\neg L?$ é falso ou indefinido, o valor verdade do corpo não será alterado na versão canônica, pois *not* $\neg L?$ é respectivamente verdadeiro ou indefinido;

  - Se $\neg L?$ é verdadeiro, então, por coerência, obtém-se que *not* L é verdadeiro. Assim, o valor verdade de L é sobrecarregado passando a ser falso. Desse modo, a regra em P passa a ser falsa. Por outro lado, como $\neg L?$ é verdadeiro, *not* $\neg L?$ é falso. Com isso, a regra correspondente em P^c também é falsa.

- Se $L?$ é indefinido, um raciocínio similar pode ser feito.

A divisão de um programa por uma interpretação é então definida:

Definição 5.21 (A divisão $\frac{P^c}{I}$) *Sejam P^c um programa em lógica da inconsistência epistêmica na forma canônica e I uma interpretação para P^c. Um programa $\frac{P^c}{I}$ é o programa obtido de P^c executando não deterministicamente as seguintes operações até não mais serem aplicáveis:*

- *Remova todas as regras contendo um literal default not α tal que $\alpha \in I$;*
- *Remova das regras os literais default not α tal que not $\alpha \in I$.*

Em seguida, todos os literais default restantes são trocados pela proposição **u**.

A divisão de P^c por I torna-se não determinística quando uma contradição por default ocorre em I permitindo que as duas regras da definição 5.21 sejam simultaneamente aplicáveis.

Exemplo 5.22 Seja P o programa em lógica na forma canônica

$$P = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c?, not\ \neg c & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

Suponha que $I = \{c, c?, \neg c?, d, d?\} \cup\ not\{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$. O programa P pode ser dividido por I de quatro maneiras:

$$P_1 = \left\{ \begin{array}{ll} d? \leftarrow c? & c \\ a? \leftarrow \mathbf{u} & c? \\ \neg c? & \end{array} \right\} \quad P_2 = \left\{ \begin{array}{ll} d? \leftarrow c? & c \\ a? \leftarrow \mathbf{u} & c? \\ b? & \neg c? \end{array} \right\}$$

$$P_3 = \left\{ \begin{array}{ll} d \leftarrow c & c \\ d? \leftarrow c? & c? \\ a? \leftarrow \mathbf{u} & \neg c? \end{array} \right\} \quad P_4 = \left\{ \begin{array}{ll} d \leftarrow c & c \\ d? \leftarrow c? & c? \\ a? \leftarrow \mathbf{u} & \neg c? \\ b? & \end{array} \right\}$$

Como toda divisão $\frac{P}{I}$ é um programa não negativo plausível, um menor modelo trivalorado $least(\frac{P}{I})$ pode ser determinado. Entretanto, uma pequena alteração é feita na

definição 5.18 do operador *least* para lidar com as divisões $\frac{P}{I}$. Em vez de fazer $\Psi^{ei\uparrow 0} = \mathcal{B}_{\frac{P}{I}}$, será utilizado $\Psi^{ei\uparrow 0} = \mathcal{B}_P$. Como pode ser facilmente verificado, em cada $\Psi^{ei\uparrow i}_{\frac{P}{I}} = T_i \cup not\ F_i$ com $0 \leq i \leq w$, por essa nova definição, os literais pertencentes a T_i serão os mesmos da definição original. A diferença é que cada F_i será acrescido dos literais que pertencem a $\mathcal{B}_P$ e não pertencem a $\mathcal{B}_{\frac{P}{I}}$. Portanto, o valor lógico dos literais que ocorrem em $\frac{P}{I}$ não será alterado.

Exemplo 5.23 Sejam P_1, P_2, P_3 e P_4 os programas não negativos plausíveis do exemplo 5.22. Os seus respectivos menores p-modelos podem ser calculados utilizando o teorema 5.18 e as considerações acima:

$least(P_1) = \{c, c?, \neg c?, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, \neg c, d, \neg d, \neg d?\}$
$least(P_2) = \{b?, c, c?, \neg c?, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, d, \neg d, \neg d?\}$
$least(P_3) = \{c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, \neg c, \neg d, \neg d?\}$
$least(P_4) = \{b?, c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, \neg d, \neg d?\}$

Feito isso, ainda resta impor o princípio da coerência aos menores p-modelos. Com esse objetivo, é feita uma generalização do operador Coh^P [68, 2, 6] para um programa em lógica da inconsistência epistêmica.

Definição 5.24 (O operador Coh^{ei}) *Seja $I' = T' \cup not\ F'$ um conjunto de literais. Coh^{ei} é definido como a interpretação $T \cup not\ F$ tal que*

$$T = T' \text{ e } F = F' \cup \{\neg L? \mid L \in T\} \cup \{\neg L \mid L? \in T\}.$$

Exemplo 5.25 Sejam $least(P_1)$, $least(P_2)$, $least(P_3)$ e $least(P_4)$ os menores p-modelos apresentados no exemplo 5.23. Ao aplicar o operador Coh^{ei}, obtém-se

$$\begin{aligned}
Coh^{ei}(least(P_1)) &= \{c, c?, \neg c?, d?\} \cup \\
&\quad not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
Coh^{ei}(least(P_2)) &= \{b?, c, c?, \neg c?, d?\} \cup \\
&\quad not\{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, \neg d, \neg d?\} \\
Coh^{ei}(least(P_3)) &= \{c, c?, \neg c?, d, d?\} \cup \\
&\quad not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, \neg d, \neg d?\} \\
Coh^{ei}(least(P_4)) &= \{b?, c, c?, \neg c?, d, d?\} \cup \\
&\quad not\{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, \neg d, \neg d?\}
\end{aligned}$$

Na seqüência, os resultados envolvendo o operador Coh^{ei} aplicado aos menores p-modelos das divisões do programa são agrupados pelo operador Φ^{ei}, que é uma adaptação de Φ^p [68, 2, 20].

Definição 5.26 (O operador Φ^{ei}) *Sejam P um programa em lógica da inconsistência epistêmica na forma canônica, I uma interpretação, K um conjunto tal que $k \in K$ sss $\frac{P}{I}$ é uma divisão do programa P por I. Então*

$$\Phi^{ei}_P(I) = \cup_{k \in K} Coh^{ei}(least(P_k)).$$

Exemplo 5.27 Sejam P e I respectivamente o programa canônico e a interpretação apresentados no exemplo 5.22. Dos resultados obtidos no exemplo 5.25, Φ^{ei}_P pode ser determinado da seguinte maneira:

$$\begin{aligned}
\Phi^{ei}_P(I) &= Coh^{ei}(least(P_1)) \cup\ Coh^{ei}(least(P_2)) \cup Coh^{ei}(least(P_3)) \cup\ Coh^{ei}(least(P_4)) \\
\Phi^{ei}_P(I) &= \{b?, c, c?, \neg c?, d, d?\} \cup \\
&\quad not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}
\end{aligned}$$

Assim como em [20], não é necessário considerar todos os programas $P_K s$ obtidos das divisões de programa por uma interpretação para definir o operador Φ^{ei}, mas apenas a menor e a maior dessas divisões. Antes, porém, é preciso provar que elas sempre existem para qualquer programa:

Proposição 5.28 (Programas $P_I^{\perp}$ e $P_I^{\top}$) *Sejam P um programa em lógica da inconsistência epistêmica na forma canônica e I uma interpretação. Considere todos os programas $P_k s$ obtidos de $\frac{P}{I}$. Esses programas formam um reticulado completo sob a ordem de inclusão de conjunto. Isso significa que há um menor e um maior programa que pode ser determinado como segue:*

- *O menor programa $P_I^{\perp}$ é obtido sempre escolhendo para aplicação a primeira regra na definição de $\frac{P}{I}$, quando ambas são aplicáveis;*
- *O maior programa $P_I^{\top}$ é obtido sempre escolhendo para aplicação a segunda regra na definição de $\frac{P}{I}$, quando ambas são aplicáveis.*

(*Prova*)

A demonstração dessa proposição é análoga à mostrada em [20] para $P_I^{\perp}$ e $P_I^{\top}$ obtidos a partir de um programa em lógica estendido.

Todas as divisões de um programa podem ser calculadas aplicando inicialmente as regras para os casos determinísticos, ou seja, para os literais default $not\ \alpha$ quando α e $not\ \alpha$ não simultaneamente pertencem a I. Um único programa P_d é obtido nesse passo. Seja $P_d = P' \cup P''$, em que P' contém todas as regras de P_d que não possuem qualquer literal default. O programa P_r é obtido de P'' removendo todos os literais default dos corpos das regras de P''. Todos os programas resultantes de $\frac{P}{I}$ podem ser obtidos adicionando a P' subconjuntos de P_r. Em particular, $P' = P_I^{\perp}$ e $P' \cup P_r = P_I^{\top}$. $\Diamond$

Exemplo 5.29 Sejam I a interpretação

$$I = \{c, c?, \neg c?, d, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$$

e P o programa em lógica na forma canônica abaixo:

$$P = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c?, not\ \neg c & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

que foram apresentados no exemplo 5.22. Aplicando as regras determinísticas, obtém-se o programa

$$P_d = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c? & a? \leftarrow \mathbf{u} & c? & \end{array} \right\}.$$

Das ponderações balisadas na prova da proposição 5.28, conclui-se que

$$P' = \left\{ \begin{array}{ll} d? \leftarrow c? & c \\ a? \leftarrow \mathbf{u} & c? \\ \neg c? & \end{array} \right\} \quad P_r = \left\{ \begin{array}{l} d \leftarrow c \\ b? \end{array} \right\}$$

Os programas resultantes da divisão de P por I podem ser obtidos, ao adicionar a P' subconjuntos de P_r, gerando, portanto, os quatro programas P_1, P_2, P_3 e P_4 elencados no exemplo 5.22, sendo que $P_I^{\perp} = P_1 = P'$ e $P_I^{\top} = P_4 = P' \cup P_r$.

Observe que se P não estivesse na forma canônica, o programa P' teria a regra $d \leftarrow c$ e d seria verdadeiro em vez de falso em $least(P_1)$ e $least(P_2)$. Conseqüentemente, d não seria falso em P. Isso contraria o princípio da coerência, pois em P, $\neg c?$ implica $not\ c$. Como a única regra para d em P é $d \leftarrow c$ (supondo que P não esteja na forma canônica), então $not\ d$ deveria também ser concluído[13]. Com o programa na forma canônica, é assegurada a propagação do valor falso como conseqüência de qualquer $not\ L?$ (resp. $not\ L$) implicado por $\neg L$ (resp. $\neg L?$) através do princípio da coerência.

Antes de demonstrar que na definição do operador Φ_P^{ei}, é suficiente levar em consideração apenas os programas $P_I^{\perp}$ e $P_I^{\top}$, é apresentado o seguinte lema:

Lema 5.30 (O operador least é monotônico sob a ordem de verdade padrão) *Sejam P_1 e P_2 dois programas não negativos plausíveis com a mesma base de Herbrand $\mathcal{B}_P$. Se $P_1 \subseteq P_2$, então $least(P_1) \preceq_T least(P_2)$.*

(*Prova*)

Sejam $least(P_1) = T_1 \cup not\ F_1$ e $least(P_2) = T_2 \cup not\ F_2$. Deve-se demonstrar que $T_1 \subseteq T_2$ e $F_2 \subseteq F_1$. Inicialmente, por indução, será mostrado que

[13] Além de também se concluir d a partir de c.

$$\alpha \in \Psi_{P_1}^{ei\uparrow n} \Rightarrow \alpha \in \Psi_{P_2}^{ei\uparrow n} \text{ e } not\ \alpha \in \Psi_{P_2}^{ei\uparrow n} \Rightarrow not\ \alpha \in \Psi_{P_1}^{ei\uparrow n}.$$

Base de indução: Trivial, já que $\Psi_{P_1}^{ei\uparrow 0} = \Psi_{P_2}^{ei\uparrow 0} = not\ \mathcal{B}_P$.

Passo de indução: Assume-se que $\Psi_{P_1}^{ei\uparrow n} \preceq_T \Psi_{P_2}^{ei\uparrow n}$ para um n arbitrário. $\alpha \in \Psi_{P_1}^{ei\uparrow n+1}$ sss existe uma regra em P_1 da forma $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ tal que para cada $1 \leq i \leq m$, $\alpha_i \in \Psi_{P_1}^{ei\uparrow n}$. Como $P_1 \subseteq P_2$, essa regra também pertence a P_2. Desse modo, por hipótese de indução, para cada $1 \leq i \leq m$, α_i pertence a $\Psi_{P_2}^{ei\uparrow n}$. Conseqüentemente $\alpha \in \Psi_{P_2}^{ei\uparrow n+1}$.

Por outro lado, um literal $not\ \alpha \in \Psi_{P_2}^{ei\uparrow n+1}$ sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$, existe um $1 \leq i \leq m$ tal que $not\ \alpha_i \in \Psi_{P_2}^{ei\uparrow n}$. Por hipótese de indução, existe um $1 \leq i \leq m$ tal que $not\ \alpha_i \in \Psi_{P_1}^{ei\uparrow n}$. Como $P_1 \subseteq P_2$, um subconjunto dessas regras de P_2 para α está em P_1 (eventualmente esse subconjunto é vazio). Assim, $not\ \alpha \in \Psi_{P_1}^{ei\uparrow n+1}$.

Dessa indução sobre n, resulta que $\Psi_{P_1}^{ei\uparrow \omega} \preceq_T \Psi_{P_2}^{ei\uparrow \omega}$, ou seja, $least(P_1) \preceq_T least(P_2)$. $\Diamond$

Teorema 5.31 (Somente $\mathbf{P_I^\perp}$ e $\mathbf{P_I^\top}$ são necessários para computar Φ^{ei}) *Sejam P um programa em lógica da inconsistência epistêmica na forma canônica e I uma interpretação. Então*

$$\Phi_P^{ei}(I) = Coh^{ei}(least(P_I^\perp)) \cup Coh^{ei}(least(P_I^\top)).$$

Além disso, $\alpha \in \Phi_P^{ei}(I)$ sss $\alpha \in least(P_I^\top)$ e $not\ \alpha \in \Phi_P^{ei}(I)$ sss $not\ \alpha \in least(P_I^\perp)$ ou $\neg L? \in least(P_I^\top)$ se α é um literal objetivo L (resp. $\neg L \in least(P_I^\top)$ se α é um literal plausível $L?$).

(Prova)

Pela proposição 5.28, para todo programa P_k obtido a partir da divisão não determinística $\frac{P}{I}$, sabe-se que $P_I^\perp \subseteq P_k$ e $P_k \subseteq P_I^\top$. Sejam $least(P_k) = T_k \cup not\ F_k$, $least(P_I^\perp) = T^\perp \cup not\ F^\perp$ e $least(P_I^\top) = T^\top \cup not\ F^\top$. Pelo lema 5.30, tem-se que $\forall_k F_k \subseteq F^\perp$ e $\forall_k T_k \subseteq T^\top$. Os resultados seguem diretamente dessas inclusões. $\Diamond$

Exemplo 5.32 Sejam P e I respectivamente o programa canônico e a interpretação apresentados no exemplo 5.22. Do exemplo 5.29, obtém-se que

$$P_I^\perp = \left\{ \begin{array}{ll} d? \leftarrow c? & c \\ a? \leftarrow \mathbf{u} & c? \\ \neg c? & \end{array} \right\} \quad P_I^\top = \left\{ \begin{array}{ll} d \leftarrow c & c \\ d? \leftarrow c? & c? \\ a? \leftarrow \mathbf{u} & \neg c? \\ b? & \end{array} \right\}$$

Dos resultados obtidos no exemplo 5.25, sabe-se que

$$\begin{aligned} Coh^{ei}(least(P_I^{\perp})) &= \{c, c?, \neg c?, d?\} \cup \\ &\quad not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\ Coh^{ei}(least(P_I^{\top})) &= \{b?, c, c?, \neg c?, d, d?\} \cup \\ &\quad not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, \neg d, \neg d?\} \end{aligned}$$

Desde que $\Phi_P^{ei}(I) = Coh^{ei}(least(P_I^{\perp})) \cup Coh^{ei}(least(P_I^{\top}))$,

$$\Phi_P^{ei}(I) = \{b?, c, c?, \neg c?, d, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}.$$

Como era esperado, esse resultado é idêntico ao obtido no exemplo 5.27.

Teorema 5.33 (Monotonicidade de Φ^{ei}) *O operador Φ^{ei} é monotônico sob o ordenamento de Fitting.*

(*Prova*)

Sejam P um programa em lógica da inconsistência epistêmica no formato canônico e $I_1 = T_1 \cup not\ F_1$ e $I_2 = T_2 \cup not\ F_2$ interpretações para P. Provar que o operador Φ^{ei} é monotônico sob o ordenamento de Fitting ($\preceq_F$) é demonstrar que se $I_1 \preceq_F I_2$, então $\Phi_P^{ei}(I_1) \preceq_F \Phi_P^{ei}(I_2)$. Como $\Phi_P^{ei}(I) = Coh^{ei}(least(P_I^{\perp})) \cup Coh^{ei}(least(P_I^{\top}))$, pode-se provar esse teorema provando separadamente os dois lemas abaixo referentes à monotonicidade de $(least(P_I^{\perp}) \cup least(P_I^{\top}))$ (lema 5.34) e à do operador Coh_P^{ei} (lema 5.35).

Lema 5.34 *Sejam P um programa em lógica da inconsistência epistêmica na forma canônica e $I_1 = T_1 \cup not\ F_1$ e $I_2 = T_2 \cup not\ F_2$ interpretações para P. Se $I_1 \preceq_F I_2$ então $(least(P_{I_1}^{\perp}) \cup least(P_{I_1}^{\top})) \preceq_F (least(P_{I_2}^{\perp}) \cup least(P_{I_2}^{\top}))$.*

(*Prova*)

Suponha que $I_1 \preceq_F I_2$, ou seja, $T_1 \subseteq T_2$ e $F_1 \subseteq F_2$. Quer-se demonstrar que se $\alpha \in (least(P_{I_1}^{\perp}) \cup least(P_{I_1}^{\top}))$ então $\alpha \in (least(P_{I_2}^{\perp}) \cup least(P_{I_2}^{\top}))$ e se $not\ \alpha \in (least(P_{I_1}^{\perp}) \cup least(P_{I_1}^{\top}))$ então $not\ \alpha \in (least(P_{I_2}^{\perp}) \cup least(P_{I_2}^{\top}))$.

Sejam $least(P_I^{\perp}) = T^{\perp} \cup not\ F^{\perp}$ e $least(P_I^{\top}) = T^{\top} \cup not\ F^{\top}$. Do teorema 5.30, obtém-se que $T^{\perp} \subseteq T^{\top}$ e $F^{\top} \subseteq F^{\perp}$. Conseqüentemente, a prova desse lema resume-se a demonstração de que

$$\alpha \in least(P_{I_1}^{\top}) \Rightarrow \alpha \in least(P_{I_2}^{\top}) \text{ e } not\ \alpha \in least(P_{I_1}^{\perp}) \Rightarrow not\ \alpha \in least(P_{I_2}^{\perp}).$$

Por sua vez, $least(P) = \Psi_P^{ei\uparrow\omega}$ em que P é um programa não negativo plausível. Assim, por indução, deve-se demonstrar que

$$\alpha \in \Psi^{ei\uparrow n}_{P^{\top}_{I_1}} \Rightarrow \alpha \in \Psi^{ei\uparrow n}_{P^{\top}_{I_2}} \text{ e } not\ \alpha \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_1}} \Rightarrow not\ \alpha \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_2}}.$$

Base de indução: Trivial, já que $\Psi^{ei\uparrow 0}_{P^{\top}_{I_1}} = \Psi^{ei\uparrow 0}_{P^{\top}_{I_2}} = \Psi^{ei\uparrow 0}_{P^{\bot}_{I_1}} = \Psi^{ei\uparrow 0}_{P^{\bot}_{I_2}} = not\ \mathcal{B}_P$.

Passo de indução: Assume-se que $\alpha \in \Psi^{ei\uparrow n}_{P^{\top}_{I_1}} \Rightarrow \alpha \in \Psi^{ei\uparrow n}_{P^{\top}_{I_2}}$ e $not\ \alpha \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_1}} \Rightarrow not\ \alpha \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_2}}$ para um n arbitrário.

$\alpha \in \Psi^{ei\uparrow n+1}_{P^{\top}_{I_1}}$ sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P^{\top}_{I_1}$, com $m \geq 0$, tal que para todo $1 \leq i \leq m$, $\alpha_i \in \Psi^{ei\uparrow n}_{P^{\top}_{I_1}}$ sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$, com $m, k \geq 0$, em P tal que para todo $1 \leq i \leq m$, $\alpha_i \in \Psi^{ei\uparrow n}_{P^{\top}_{I_1}}$ e para todo $1 \leq j \leq k$, $not\ \beta_j \in I_1$. Como $I_1 \preceq_F I_2$, para todo $1 \leq j \leq k$, $not\ \beta_j \in I_2$. Portanto, existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P^{\top}_{I_2}$. Por hipótese de indução, para todo $1 \leq i \leq m$, $\alpha_i \in \Psi^{ei\uparrow n}_{P^{\top}_{I_2}}$. Com isso, tem-se que $\alpha \in \Psi^{ei\uparrow n+1}_{P^{\top}_{I_2}}$.

Se $not\ \alpha \in \Psi^{ei\uparrow n+1}_{P^{\bot}_{I_1}}$ sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P^{\bot}_{I_1}$, existe $1 \leq i \leq m$, tal que $not\ \alpha_i \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_1}}$ sss para cada uma dessas regras de $P^{\bot}_{I_1}$, existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$, com $m, k \geq 0$, em P tal que $not\ \alpha_i \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_1}}$ e para todo $1 \leq j \leq k$, $not\ \beta_j \in I_1$. Como $I_1 \preceq_F I_2$, para todo $1 \leq j \leq k$, $not\ \beta_j \in I_2$ em cada uma dessas regras. Portanto, toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P^{\bot}_{I_1}$ também pertence a $P^{\bot}_{I_2}$. Por hipótese de indução, para cada uma dessas regras de $P^{\bot}_{I_2}$, existe $1 \leq i \leq m$, tal que $not\ \alpha_i \in \Psi^{ei\uparrow n}_{P^{\bot}_{I_2}}$. Conseqüentemente, $not\ \alpha \in \Psi^{ei\uparrow n+1}_{P^{\bot}_{I_2}}$.

Desse modo, se $I_1 \leq_F I_2$, então $(least(P^{\bot}_{I_1}) \cup least(P^{\top}_{I_1})) \preceq_F (least(P^{\bot}_{I_2}) \cup least(P^{\top}_{I_2}))$.
$\Diamond$

Lema 5.35 *Sejam P um programa em lógica da inconsistência epistêmica na forma canônica, $I_1 = T_1 \cup not\ F_1$ e $I_2 = T_2 \cup not\ F_2$ duas interpretações para P. Se $I_1 \preceq_F I_2$, então $Coh^{ei}(I_1) \preceq_F Coh^{ei}(I_2)$.*

(*Prova*)

Suponha que $I_1 \preceq_F I_2$, ou seja, $T_1 \subseteq T_2$ e $F_1 \subseteq F_2$. Provar que $Coh^{ei}(I_1) \leq_F Coh^{ei}(I_2)$ é por definição equivalente a provar que $T_1 \cup not\ (F_1 \cup \{\neg L? \mid L \in T_1\} \cup \{\neg L \mid L? \in T_1\}) \leq_F T_2 \cup not\ (F_2 \cup \{\neg L? \mid L \in T_2\} \cup \{\neg L \mid L? \in T_2\})$.

Como por hipótese, $T_1 \subseteq T_2$, é suficiente provar que $F_1 \cup \{\neg L? \mid L \in T_1\} \cup \{\neg L \mid L? \in T_1\} \subseteq F_2 \cup \{\neg L? \mid L \in T_1\} \cup \{\neg L \mid L? \in T_1\} \cup \{\neg L? \mid L \in T_2 - T_1\} \cup \{\neg L \mid L? \in T_2 - T_1\}$, que é equivalente a

$$F_1 \subseteq F_2 \cup \{\neg L? \mid L \in T_2 - T_1\} \cup \{\neg L \mid L? \in T_2 - T_1\}.$$

Por hipótese, já é sabido que $F_1 \subseteq F_2$. Portanto, $F_1 \subseteq F_2 \cup \{\neg L? \mid L \in T_2 - T_1\} \cup \{\neg L \mid L? \in T_2 - T_1\}$. $\Diamond$

Retornando à demonstração do teorema 5.33, do lema 5.34, obtém-se que $I_1 \preceq_F I_2 \Rightarrow (least(P_{I_1}^{\top}) \cup least(P_{I_1}^{\bot})) \preceq_F (least(P_{I_2}^{\top}) \cup least(P_{I_1}^{\bot}))$, resultando, pelo lema 5.35, que $Coh^{ei}(least(P_{I_1}^{\top}) \cup least(P_{I_1}^{\bot})) \preceq_F Coh^{ei}(least(P_{I_2}^{\top}) \cup least(P_{I_2}^{\bot}))$. Daí, conclui-se que

$$(Coh^{ei}(least(P_{I_1}^{\top})) \cup Coh(least(P_{I_1}^{\bot}))) \preceq_F (Coh^{ei}(least(P_{I_2}^{\top})) \cup Coh(least(P_{I_2}^{\bot}))).$$

Portanto, $\Phi^{ei}(I_1) \preceq_F \Phi^{ei}(I_2)$. $\Diamond$

A semântica bem fundada da inconsistência epistêmica pode então ser definida como segue:

Definição 5.36 (XSM_{EI}, WFM_{EI}, $WFSX_{EI}$) *Uma interpretação I para um programa em lógica da inconsistência epistêmica na forma canônica P é chamado de modelo estável da inconsistência epistêmica (XSM_{EI}[14]) de P sss $\Phi_P^{ei}(I) = I$. O menor XSM_{EI} sob a ordem de Fitting é chamado de modelo bem fundado da inconsistência epistêmica (WFM_{EI}[15]) de P. A semântica bem fundada da inconsistência epistêmica $WFSX_{EI}$ para um programa P é constituída de todo XSM_{EI} de P.*

Teorema 5.37 (Todo XSM_{EI} é um modelo) *Todo XSM_{EI} para um programa em lógica da inconsistência epistêmica P é um modelo para P.*

(*Prova*)

Suponha por absurdo que I é um XSM_{EI} para P e I não é um modelo para P. Desse modo, existe pelo menos uma regra $\alpha \leftarrow \mathcal{C}$ pertencente a P tal que $\hat{I}(\alpha \leftarrow \mathcal{C}) \neq 1$ com $\mathcal{C}$ representando o corpo da regra. Isso ocorre se e somente se
$\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ e $\hat{I}(\mathcal{C}) = 1$ ou $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ e $\hat{I}(\neg L?) \neq 1$ se α é um literal objetivo L (resp. $\hat{I}(\neg L) \neq 1$ se α é um literal plausível $L?$).

Se $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ e $\hat{I}(\mathcal{C}) = 1$, então desde que $\hat{I}(\mathcal{C}) = 1$ e I é um XSM_{EI} para P, $\alpha \in I$, ou seja, $\hat{I}(\alpha) = 1$. Conseqüentemente, $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ e $\hat{I}(\mathcal{C}) = 1$ não é possível.

Se $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ e $\hat{I}(\neg L?) \neq 1$ se α é um literal objetivo L (resp. $\hat{I}(\neg L) \neq 1$ se α é um literal plausível $L?$), existem duas possibilidades:

$$\hat{I}(\mathcal{C}) = 1 \text{ ou } \hat{I}(\mathcal{C}) = \frac{1}{2}.$$

Como demonstrado no caso anterior, se $\hat{I}(\mathcal{C}) = 1$, $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ não é possível. Por outro lado, se $\hat{I}(\mathcal{C}) = \frac{1}{2}$, tem-se que o valor verdade de α em $least(P_I^{\top})$ e em $least(P_I^{\bot})$ é igual a $\frac{1}{2}$. Desde que $\hat{I}(\neg L?) \neq 1$ se α é um literal objetivo L (resp. $\hat{I}(\neg L) \neq 1$ se α é um literal plausível $L?$), o valor verdade de α em $Coh^{ei}(least(P_I^{\top}))$ e em $Coh^{ei}(least(P_I^{\bot}))$ é

[14] *De eXtended Stable Models of Epistemic Inconsistency.*
[15] *De Well-Founded Model of Epistemic Inconsistency.*

também igual a $\frac{1}{2}$. Como I é um $XSM_{EI}(P)$, $\hat{I}(\alpha) = \hat{I}(\mathcal{C}) = \frac{1}{2}$, logo, $\hat{I}(\alpha) < \hat{I}(\mathcal{C})$ não é possível.

Conseqüentemente, todo XSM_{EI} para P é um modelo para P. $\Diamond$

Como Φ^{ei} é monotônico sob a ordem de Fitting, todo programa em lógica da inconsistência epistêmica P possui um único WFM_{EI}, que pode ser construtivamente calculado da seguinte maneira:

Teorema 5.38 *O modelo bem fundado da inconsistência epistêmica de um programa em lógica P na forma canônica, $WFM_{EI}(P)$, é o menor ponto fixo sob o ordenamento de Fitting, que pode ser obtido através da seqüência abaixo:*

$$\begin{aligned} I_0 &= \{\} \\ I_i &= \Phi_P^{ei}(I_{i-1}) \\ I_\delta &= \bigcup \{I_\alpha \mid \alpha < \delta\} \text{ para um limite ordinal } \delta \end{aligned}$$

$WFM_{EI}(P) = I_\lambda$, em que I_λ é o menor ponto fixo de Φ_P^{ei}.

Exemplo 5.39 Seja P o programa canônico do exemplo 5.22:

$$P = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c?, not\ \neg c & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

O seu modelo bem fundado da inconsistência epistêmica pode ser obtido da seguinte maneira:

$I_0 = 0$

$I_1 = \Phi_P^{ei}(I_0) = Coh^{ei}(least(P_{I_0}^{\perp})) \cup Coh^{ei}(least(P_{I_0}^{\top}))$

De $I_0 = \{\}$, obtém-se que

$$P_{I_0}^{\perp} = P_{I_0}^{\top} = \left\{ \begin{array}{llll} d \leftarrow c, \mathbf{u} & b? \leftarrow \mathbf{u}, \mathbf{u} & c & \neg c? \\ d? \leftarrow c?, \mathbf{u} & a? \leftarrow \mathbf{u}, \mathbf{u} & c? & \end{array} \right\}$$

Dessa maneira,

$I_1 = \{c, c?, \neg c?\} \cup\ not\{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, \neg d, \neg d?\}.$

$I_2 = \Phi_P^{ei}(I_1) = Coh^{ei}(least(P_{I_1}^{\perp})) \cup Coh^{ei}(least(P_{I_1}^{\top}))$

Na segunda iteração, os seguintes programas são obtidos:

$$P_{I_1}^{\perp} = \left\{ \begin{array}{ll} d? \leftarrow c? & c? \\ b? \leftarrow \mathbf{u} & c? \\ a? \leftarrow \mathbf{u} & \neg c? \end{array} \right\} \quad P_{I_1}^{\top} = \left\{ \begin{array}{lll} d \leftarrow c & a? \leftarrow \mathbf{u} & \neg c? \\ d? \leftarrow c? & c & c? \\ b? \leftarrow \mathbf{u} & & \end{array} \right\}$$

Resultando em

$I_2 = \{c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$.

No tocante a terceira iteração, sabe-se que

$I_3 = \Phi_P^{ei}(I_2) = Coh^{ei}(least(P_{I_2}^{\perp})) \cup Coh^{ei}(least(P_{I_2}^{\top}))$

De I_2, obtém-se os programas

$$P_{I_2}^{\perp} = \left\{ \begin{array}{ll} d? \leftarrow c? & c? \\ a? \leftarrow \mathbf{u} & \neg c? \\ c & \end{array} \right\} \qquad P_{I_2}^{\top} = \left\{ \begin{array}{ll} d \leftarrow c & c \\ d? \leftarrow c? & c? \\ a? \leftarrow \mathbf{u} & \neg c? \\ b? & \end{array} \right\}$$

Por conseqüinte,

$I_3 = \{b?, c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$

Como pode ser verificado, $I_4 = I_3$, portanto I_4 é o menor ponto fixo para essa seqüência e assim $WFM_{EI}(P) = I_4 = \{b?, c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$. Desse resultado, conclui-se que

$b?, c, c?, \neg c?, d$ e $d?$ são verdadeiros em $WFM_{EI}(P)$;

$a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d$ e $\neg d?$ são falsos em $WFM_{EI}(P)$;

$a?$ é indefinido em $WFM_{EI}(P)$.

5.2.3 Uma definição de $WFSX_{EI}$ baseada numa construção de menor ponto fixo iterado

Em [80, 76], é apresentada uma definição alternativa de *WFS*, usando para tanto uma construção de menor ponto fixo iterado. Posteriormente, essa definição foi generalizada em [68, 6] para caracterizar *WFSX* e em [20] para a sua versão paraconsistente, $WFSX_P$. Nesta subseção, é utilizada uma construção similar para definir $WFSX_{EI}$, que é demonstrada ser equivalente à exibida na subseção anterior.

Definição 5.40 (O operador θ^{ei}) *Seja P um programa em lógica da inconsistência epistêmica, J uma interpretação e $\mathcal{I}_P$ o conjunto de todas as interpretações trivaloradas para P. O operador $\theta_J^{ei} : \mathcal{I}_P \rightarrow \mathcal{I}_P$ é definido como segue: se $I \in \mathcal{I}_P$ é uma interpretação de P e α é um literal objetivo ou plausível básico, então $\theta_J^{ei}(I)$ é a interpretação dada por*

1. *$\alpha \in \theta_J^{ei}(I)$ sss há uma regra $\alpha \leftarrow \alpha_1, ..., \alpha_m, not\ \beta_1 \ldots not\ \beta_n$ em P tal que para todo $i \leq n$, $not\ \beta_i \in J$ e para todo $j \leq m$, $\alpha_j \in I$;*

2. *$not\ \alpha \in \theta_J^{ei}(I)$ sss pelo menos um dos seguintes itens ocorrer:*

(a) *Para toda regra* $\alpha \leftarrow \alpha_1, ..., \alpha_m, not\ \beta_1 \ldots not\ \beta_n$ *em* P*, há um* $i \leq n$ *tal que* $\beta_i \in J$ *ou há um* $j \leq m$ *tal que* $not\ \alpha_j \in I$*;*

(b) *Um literal objetivo* $\neg L? \in J$ *se* α *é um literal objetivo* L *ou* $\neg L \in J$ *se* α *é um literal plausível* $L?$*.*

Dessa definição, pode-se afirmar que o operador $\theta_J^{ei}(I)$ determina os literais verdadeiros e os falsos que são obtidos de imediato a partir de J e I. Além disso, $\theta_J^{ei}(I)$ é demonstrado ser monotônico e possuir um único ponto fixo sob a ordem de verdade padrão:

Proposição 5.41 (Monotonicidade do operador θ^{ei}) *Para toda interpretação* J*, o operador* θ_J^{ei} *é monotônico e possui um único menor ponto fixo sob a ordem de verdade padrão dado por* $\theta_J^{ei\uparrow\omega}(not\ \mathcal{B}_P)$*. Esse ponto fixo é também denotado por* $\omega^{ei}(J)$*.*

(*Prova*)

Inicialmente, é demonstrado que θ_J^{ei} é monotônico sob a ordem de verdade padrão. Assim, seja $I_1 = T_1 \cup not\ F_1$ e $I_2 = T_2 \cup not\ F_2$ duas interpretações tal que $T_1 \subseteq T_2$ e $F_2 \subseteq F_1$. Deseja-se mostrar que para qualquer literal α,

$$\alpha \in \theta_J^{ei}(I_1) \Rightarrow \alpha \in \theta_J^{ei}(I_2) \text{ e } not\ \alpha \in \theta_J^{ei}(I_2) \Rightarrow not\ \alpha \in \theta_J^{ei}(I_1).$$

Suponha que $\alpha \in \theta_J^{ei}(I_1)$. Pela definição 5.40, há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ em P tal que para qualquer $i \leq n$, $not\ \beta_i \in J$ e para qualquer $j \leq m$, $\alpha_j \in I_1$. Como J é o mesmo para ambas as aplicações do operador θ_J^{ei} e por hipótese se $\alpha_i \in I_1 \Rightarrow \alpha_i \in I_2$, conclui-se que $\alpha \in \theta_J^{ei}(I_2)$.

Agora, suponha que $not\ \alpha \in \theta_J^{ei}(I_2)$. Pela definição 5.40, $\neg L? \in J$ se α é um literal objetivo L ou $\neg L \in J$ se α é um literal plausível $L?$. Uma outra possibilidade é que para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ em P, há um $i \leq n$ tal que $\beta_i \in J$ ou existe $j \leq m$ tal que $not\ \alpha_j \in I_2$. Como J é o mesmo para ambas as aplicações de θ_J^{ei} e por hipótese se $not\ \alpha \in I_2 \Rightarrow not\ \alpha \in I_1$, então $not\ \alpha \in \theta_J^{ei}(I_1)$.

Para demonstrar que em θ_J^{ei} há um único menor ponto fixo sob a ordem de verdade padrão, deve-se provar a continuidade de θ_J^{ei}. Como já foi estabelecido sua monotonicidade, resta provar que θ_J^{ei} é um operador finitário para todo conjunto direto Y sob a ordem de verdade padrão, ou seja,

$$\theta_J^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right) \preceq_T \bigsqcup_{y\in Y}^{T} \theta_J^{ei}(y).$$

Se $\alpha \in \theta_J^{ei}\left(\bigsqcup_{y\in Y}^{T} y\right)$, então há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ em P tal que para qualquer $i \leq n$, $not\ \beta_i \in J$ e para todo $j \leq m$, $\alpha_j \in \bigsqcup_{y\in Y}^{T} y$. Como Y é um

conjunto direto sob a ordem de verdade padrão, existe $y' \in Y$ tal que para todo $j \leq m$, $\alpha_j \in y'$. Isso quer dizer que existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ em P tal que para todo $i \leq n$, $not\ \beta_i \in J$ e para todo $j \leq m$, $\alpha_j \in y'$. Dessa forma, $\alpha \in \bigsqcup_{y \in Y}^{T} \theta_J^{ei}(y)$.

No que tange a um literal default $not\ \alpha$, deve-se provar que

$$not\ \alpha \in \bigsqcup_{y \in Y}^{T} \theta_J^{ei}(y) \Rightarrow not\ \alpha \in \theta_J^{ei}\left(\bigsqcup_{y \in Y}^{T} y\right).$$

Seja α um literal objetivo L e suponha que $not\ L \in \bigsqcup_{y \in Y}^{T} \theta_J^{ei}(y)$. Se $\neg L? \in J$, então a equação acima é válida já que o o conjunto J é o mesmo para ambas as aplicações do operador θ^{ei}. Por outro lado, suponha por absurdo que $not\ L \notin \theta_J^{ei}\left(\bigsqcup_{y \in Y}^{T} y\right)$. Nesse caso, existe uma regra $L \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ tal que para todo $i \leq n$, $\beta_i \notin J$ e para todo $j \leq m$, $not\ \alpha_j \notin \bigsqcup_{y \in Y}^{T} y$. Desde que Y é um conjunto direto sob a ordem de verdade padrão, então há um $y' \in Y$ tal que para todo literal α_j, $not\ \alpha_j \notin y'$. Com isso, obtem-se que $not\ L \notin \bigsqcup_{y \in Y}^{T} \theta_J^{ei}(y)$. Isso é uma contradiçao, já que por hipótese, $not\ L \in \bigsqcup_{y \in Y}^{T} \theta_J^{ei}(y)$. Um raciocínio similar é válido para o caso de α ser um literal plausível. $\Diamond$

Antes de estabelecer uma relação que permite definir WFM_{EI} com base nos pontos fixos de ω^{ei}, é demonstrado que o operador ω^{ei} é monotônico:

Teorema 5.42 (Monotonicidade de ω^{ei}) *Seja P um programa em lógica da inconsistência epistêmica. O operador ω^{ei} é monotônico sob a ordem de Fitting, ou seja, $J_1 \preceq_F J_2 \Rightarrow \omega^{ei}(J_1) \preceq_F \omega^{ei}(J_2)$.*

(*Prova*)

Seja J_1 e J_2 duas interpretações para um programa P tal que $J_1 \preceq_F J_2$. Deseja-se provar que $\omega^{ei}(J_1) \preceq_F \omega^{ei}(J_2)$, ou seja, $\theta_{J_1}^{ei\uparrow\omega} \preceq_F \theta_{J_2}^{ei\uparrow\omega}$.

Por indução, será mostrado que para qualquer n, $\theta_{J_1}^{ei\uparrow n} \preceq_F \theta_{J_2}^{ei\uparrow n}$.

Base de indução: Trivial, desde que $\theta_{J_1}^{ei\uparrow 0} = \theta_{J_2}^{ei\uparrow 0} = not\ \mathcal{B}_P$.

Passo de indução: Assume-se que $\theta_{J_1}^{ei\uparrow n} \preceq_F \theta_{J_2}^{ei\uparrow n}$ é válido para um certo n. Assim, seja α um literal em $\theta_{J_1}^{ei\uparrow n+1}$, portanto há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ tal que para todo $i \leq k$, $not\ \beta_i \in J_1$ e para todo $j \leq m$, $\alpha_j \in \theta_{J_1}^{ei\uparrow n}$. Como $J_1 \preceq_F J_2$ e por hipótese de indução, $\theta_{J_1}^{ei\uparrow n} \preceq_F \theta_{J_2}^{ei\uparrow n}$, conclui-se que para todo $i \leq k$, $not\ \beta_i \in J_2$ e para todo $j \leq m$, $\alpha_j \in \theta_{J_2}^{ei\uparrow n}$. Portanto $\alpha \in \theta_{J_2}^{ei\uparrow n+1}$.

Agora, suponha que $not\ \alpha \in \theta_{J_1}^{ei\uparrow n+1}$. Dessa forma, $\neg L? \in J_1$ se α é um literal objetivo L (resp. $\neg L \in J_1$ se α é um literal plausível $L?$) ou para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P, existe um $i \leq k$ tal que $\beta_i \in J_1$ ou existe $j \leq m$ tal

que *not* $\alpha_j \in \theta_{J_1}^{ei\uparrow n}$. Desde que $J_1 \preceq J_2$ e por hipótese de indução, $\theta_{J_1}^{ei\uparrow n} \preceq_F \theta_{J_2}^{ei\uparrow n}$, conclui-se que $\neg L? \in J_2$ (resp. $\neg L \in J_2$) ou para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P existe um $i \leq k$ tal que $\beta_i \in J_2$ ou existe $j \leq m$ tal que *not* $\alpha_j \in \theta_{J_2}^{ei\uparrow n}$. Disso resulta que *not* $\alpha \in \theta_{J_2}^{ei\uparrow n+1}$.

Como $\theta_{J_1}^{ei\uparrow n} \preceq_F \theta_{J_2}^{ei\uparrow n}$ é válido para qualquer n, tem-se que se $J_1 \preceq_F J_2$, $\theta_{J_1}^{ei\uparrow \omega} \preceq_F \theta_{J_2}^{ei\uparrow \omega}$, ou seja, $\omega^{ei}(J_1) \preceq_F \omega^{ei}(J_2)$. ◊

Teorema 5.43 (Os pontos fixos de Φ^{ei} coincidem com os pontos fixos de ω^{ei}) *Seja P um programa em lógica da inconsistência epistêmica na forma canônica. Então a interpretação I é um ponto fixo de Φ^{ei} sss I é um ponto fixo de ω^{ei}.*

(*Prova*)

Para um literal objetivo ou plausível α, deve-se provar que

$$\alpha \in \Phi_P^{ei}(I) \text{ sss } \alpha \in \omega^{ei}(I).$$

No teorema 5.31, é estabelecido que $\alpha \in \Phi_P^{ei}(I)$ sss $\alpha \in least(P_I^{\top})$. Por definição, $\alpha \in least(P_I^{\top})$ sss $\alpha \in \Psi_{P_I^{\top}}^{ei\uparrow\omega}$. Além disso, $\alpha \in \omega^{ei}(I)$ sss $\alpha \in \theta_I^{ei\uparrow\omega}$, em que θ_I^{ei} está aplicado ao programa P. Desse modo, a prova da equivalência acima resume-se a verificar a validade de

$$\alpha \in \Psi_{P_I^{\top}}^{ei\uparrow\omega} \text{ sss } \alpha \in \theta_I^{ei\uparrow\omega}.$$

Por indução, obtém-se os seguintes resultados:

Base de indução: Trivial, desde que $\Psi_{P_I^{\top}}^{ei\uparrow 0} = \theta_I^{ei\uparrow 0} = not\ \mathcal{B}_P$.

Passo de indução: Assume-se que $\alpha \in \Psi_{P_I^{\top}}^{ei\uparrow n}$ sss $\alpha \in \theta_I^{ei\uparrow n}$ é válido para um n qualquer. Por outro lado, $\alpha \in \Psi_{P_I^{\top}}^{ei\uparrow n+1}$

sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P_I^{\top}$ tal que para todo $j \leq m$, $\alpha_j \in \Psi_{P_I^{\top}}^{ei\uparrow n}$

sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P tal que para todo $i \leq k$, *not* $\beta_i \in I$ e para todo $j \leq m$, $\alpha_j \in \Psi_{P_I^{\top}}^{ei\uparrow n}$

sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P tal que para todo $i \leq k$, *not* $\beta_i \in I$ e para todo $j \leq m$, $\alpha_j \in \theta_I^{ei\uparrow n}$

sss $\alpha \in \theta_I^{ei\uparrow n+1}$.

No que concerne a um literal default *not* α, deve-se provar que

$$not\ \alpha \in \Phi_P^{ei}(I) \text{ sss } not\ \alpha \in \omega^{ei}(I).$$

Novamente, no teorema 5.31, é estabelecido que *not* $\alpha \in \Phi_P^{ei}(I)$ sss *not* $\alpha \in least(P_I^{\perp})$ $\cup$ $Coh^{ei}(least(P_I^{\top}))$. Por definição, *not* $\alpha \in least(P_I^{\perp})$ sss *not* $\alpha \in \Psi_{P_I^{\perp}}^{ei\uparrow\omega}$. Por seu turno,

not $\alpha \in \omega^{ei}(I)$ sss *not* $\alpha \in \theta_I^{ei\uparrow\omega}$, em que θ_I^{ei} está aplicado ao programa P. Desse modo, a prova da equivalência acima resume-se a verificação da validade de

$$not\ \alpha \in \Psi_{P_I^{\perp}}^{ei\uparrow\omega} \cup Coh^{ei}(least(P_I^{\top})) \text{ sss } not\ \alpha \in \theta_I^{ei\uparrow\omega}.$$

Para demonstrar essa equivalência, dois casos podem ser divisados:

Uma possibilidade é *not* α pertencer a I somente por causa do princípio da coerência, ou seja, porque $\neg L? \in I$ se α é um literal objetivo L (resp. $\neg L \in I$ se α é um literal plausível $L?$). Disso resulta que *not* $\alpha \in Coh^{ei}(least(P_I^{\top}))$ sss $\neg L? \in least(P_I^{\top})$ (resp. $\neg L \in least(P_I^{\top})$) sss $\neg L? \in I$ (resp.$\neg L \in I$) sss *not* $\alpha \in \theta_I^{ei\uparrow\omega}$ por causa da condição 2b da definição 5.40.

Agora, será demonstrado que *not* $\alpha \in \Psi_{P_I^{\perp}}^{ei\uparrow\omega}$ sss *not* $\alpha \in \theta_I^{ei\uparrow\omega}$ pela condição 2a da definição 5.40. Por indução, tem-se os seguintes resultados:

Base de indução: Trivial, já que $\Psi_{P_I^{\perp}}^{ei\uparrow 0} = \theta_I^{ei\uparrow 0} = not\ \mathcal{B}_P$.

Passo de indução: Assume-se que *not* $\alpha \in \Psi_{P_I^{\perp}}^{ei\uparrow n}$ sss *not* $\alpha \in \theta_I^{ei\uparrow n}$ pela condição 2a da definição 5.40 é válido para um n arbitrário. Com isso, sabe-se que *not* $\alpha \in \Psi_{P_I^{\perp}}^{ei\uparrow n+1}$

sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $P_I^{\perp}$, existe um $j \leq m$ tal que *not* $\alpha_j \in \Psi_{P_I^{\perp}}^{ei\uparrow n}$

sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P, existe um $i \leq k$ tal que $\beta_i \in I$ ou existe um $j \leq m$ tal que *not* $\alpha_j \in \Psi_{P_I^{\perp}}^{ei\uparrow n}$

sss para toda regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\,\beta_1, \ldots, not\,\beta_k$ em P, existe um $i \leq k$ tal que $\beta_i \in I$ ou existe $j \leq m$ tal que pela condição 2a da definição 5.40, *not* $\alpha_j \in \theta_I^{ei\uparrow n}$

sss *not* $\alpha \in \theta_I^{ei\uparrow n+1}$ pela condição 2a da definição 5.40. ◊

Desde que os pontos fixos de ω^{ei} coincidem com os pontos fixos de Φ^{ei}, em particular, o menor ponto fixo de ω^{ei} sob a ordem de Fitting é igual ao menor ponto fixo de Φ^{ei} sob a ordem de Fitting. Como ω^{ei} é monotônico sob essa ordem, ele sempre possui um único menor ponto fixo para cada programa. Isso quer dizer que um literal qualquer pertence a todo ponto fixo de ω^{ei} para um programa P sss esse mesmo literal pertence ao menor ponto fixo de ω^{ei} para P. Desse modo, $WFSX_{EI}$ pode ser equivalentemente definido através dos pontos fixos de ω^{ei}.

Definição 5.44 (XSM_{EI}, WFM_{EI}, $WFSX_{EI}$) *Uma interpretação J de um programa em lógica da inconsistência epistêmica P é chamada de modelo estável parcial estendido da inconsistência epistêmica (XSM_{EI})de P sss*

$$\omega^{ei}(J) = J.$$

O menor XSM_{EI} de P sob a ordem de Fitting é chamado de modelo bem fundado da inconsistência epistêmica de $P(WFM_{EI}(P))$.

A semântica $WFSX_{EI}$ de P é determinada pelo conjunto de todo XSM_{EI} de P.

Exemplo 5.45 Seja P o programa em lógica da inconsistência epistêmica na forma canônica abaixo:

$$P = \left\{ \begin{array}{ll} a? \leftarrow not\ b?, not\ \neg a & c? \leftarrow b?, not\ \neg b \\ b? \leftarrow not\ a?, not\ \neg b & c? \leftarrow a?, not\ \neg a \end{array} \right\}$$

Seja $J = not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?\}$. Disso obtém-se que

$\theta_J^{ei\uparrow 0} = not\ \mathcal{B}_P$
$\theta_J^{ei\uparrow 1} = \theta_J^{ei}(\theta_J^{ei\uparrow 0}) = not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, c?, \neg c?\}$
$\theta_J^{ei\uparrow 3} = \theta_J^{ei\uparrow 2} = not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?\}$
$\omega(J) = \theta_J^{ei\uparrow \omega} = \theta_J^{ei\uparrow 3}$

Desse resultado, sabe-se que J é XSM_{EI} de P. Como pode ser verificado, esse programa ainda possui como XSM_{EI}, $\{a?, c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?\}$ e $\{b?, c?\} \cup not\ \{a, \neg a, a?, \neg a?,\ b, \neg b, \neg b?, c, \neg c, \neg c?\}$. Portanto, $J = not\ \{a, \neg a, \neg a?, b, \neg b,\ \neg b?, c, \neg c, \neg c?\}$ é o menor XSM_{EI} de P sob a ordem de Fitting, ou seja, J é o WFM_{EI} de P.

O modelo WFM_{EI} pode ser determinado construtivamente da seguinte maneira:

Teorema 5.46 (Construção iterativa de WFM_{EI}) *O WFM_{EI} de um programa em lógica da inconsistência epistêmica P pode ser determinado construtivamente através da seqüência transfinita abaixo:*

$$\begin{array}{rcll} I_0 & = & \{\} & \\ I_{\alpha+1} & = & \omega^{ei}(I_\alpha) = \theta_{I_\alpha}^{ei\uparrow\omega} & \\ I_\delta & = & \bigcup_{\alpha<\delta} I_\alpha & \textit{para um limite ordinal } \delta \end{array}$$

Pelo teorema 5.42 e de acordo com as propriedades de operadores monotônicos, deve existir um menor λ tal que I_λ é um ponto fixo de ω^{ei}. Por definição, $WFM_{EI}(P) = I_\lambda$.

Exemplo 5.47 Seja P o programa em lógica da inconsistência epistêmica na forma canônica abaixo apresentado no exemplo 5.22:

$$P = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c?, not\ \neg c & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

Usando a definição 5.46, obtém-se os seguintes resultados:

$$
\begin{aligned}
I_0 &= \{\} \\
&\quad \theta_{I_0}^{ei\uparrow 0} = not\ \mathcal{B}_P \\
&\quad \theta_{I_0}^{ei\uparrow 1} = \theta_{I_0}^{ei}(\theta_{I_0}^{ei\uparrow 0}) = \{c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, d, \neg d, d?, \neg d?\} \\
&\quad \theta_{I_0}^{ei\uparrow 3} = \theta_{I_0}^{ei\uparrow 2} = \theta_{I_0}^{ei}(\theta_{I_0}^{ei\uparrow 1}) = \\
&\quad \{c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, \neg d, \neg d?\} \\
I_1 &= \omega^{ei}(I_0) = \theta_{I_0}^{ei\uparrow\omega} = \theta_{I_0}^{ei\uparrow 3} = \\
&\quad \{c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, \neg d, \neg d?\} \\
&\quad \theta_{I_1}^{ei\uparrow 0} = not\ \mathcal{B}_P \\
&\quad \theta_{I_1}^{ei\uparrow 1} = \theta_{I_1}^{ei}(\theta_{I_1}^{ei\uparrow 0}) = \\
&\quad \{c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, d?, \neg d?\} \\
&\quad \theta_{I_1}^{ei\uparrow 3} = \theta_{I_1}^{ei\uparrow 2} = \theta_{I_1}^{ei}(\theta_{I_1}^{ei\uparrow 1}) = \\
&\quad \{c, c?, \neg c?, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
I_2 &= \omega^{ei}(I_1) = \theta_{I_1}^{ei\uparrow\omega} = \theta_{I_1}^{x\uparrow 3} = \\
&\quad \{c, c?, \neg c?, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
&\quad \theta_{I_2}^{ei\uparrow 0} = not\ \mathcal{B}_P \\
&\quad \theta_{I_2}^{ei\uparrow 1} = \theta_{I_2}^{ei}(\theta_{I_2}^{ei\uparrow 0}) = \\
&\quad \{b?, c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, d?, \neg d?\} \\
&\quad \theta_{I_2}^{ei\uparrow 3} = \theta_{I_2}^{ei\uparrow 2} = \theta_{I_2}^{ei}(\theta_{I_2}^{ei\uparrow 2}) = \\
&\quad \{b?, c, c?, \neg c?, d, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
I_3 &= \omega^{ei}(I_2) = \theta_{I_2}^{ei\uparrow\omega} = \theta_{I_2}^{ei\uparrow 3} = \\
&\quad \{b?, c, c?, \neg c?, d, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
&\quad \theta_{I_3}^{ei\uparrow 0} = not\ \mathcal{B}_P \\
&\quad \theta_{I_3}^{ei\uparrow 1} = \theta_{I_3}^{ei}(\theta_{I_3}^{ei\uparrow 0}) = \\
&\quad \{b?, c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, d?, \neg d?\} \\
&\quad \theta_{I_3}^{ei\uparrow 3} = \theta_{I_3}^{ei\uparrow 2} = \theta_{I_3}^{ei}(\theta_{I_3}^{ei\uparrow 2}) = \\
&\quad \{b?, c, c?, \neg c?, d, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
I_4 &= \omega^{ei}(I_3) = \theta_{I_3}^{ei\uparrow\omega} = \theta_{I_3}^{ei\uparrow 3} = \\
&\quad \{b?, c, c?, \neg c?, d, d?\} \cup not\ \ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
I_5 &= \omega^{ei}(I_4) = I_4
\end{aligned}
$$

Disso, resulta que

$$\begin{aligned} WFM_{EI}(P) &= \{b?, c, c?, \neg c?, d, d?\} \cup \\ &\quad not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}. \end{aligned}$$

Como pode ser observado, θ^{ei} usa as regras de $P^{\top}$ para derivar os literais verdadeiros e usa as regras de $P^{\perp}$ para derivar os falsos. Vale também ressaltar que o princípio da coerência já está inserido na definição de ω^{ei} através da condição 2b da definição 5.40. Em contrapartida, esse princípio precisa ser explicitamente imposto mediante o operador Coh^{ei} na definição de Ψ^{ei}. Dessa diferença, resulta que nem sempre na seqüência transfinita que gera ω^{ei}, as interpretações intermediárias estejam de acordo com o princípio da coerência. No entanto, é garantido pelo teorema 5.43 que a coerência é obedecida pelos pontos fixos de ω^{ei} e em particular, pelo seu menor ponto fixo.

5.2.4 Uma definição de $WFSX_{EI}$ baseada em ponto fixos alternados

Em [92], é apresentada uma das definições mais simples e elegantes de semântica bem fundada. Essa definição está baseada na aplicação alternada do operador Γ de Gelfond e Lifschitz [28]. O nome alternado foi escolhido porque a transformação é feita em dois estágios. Inicialmente, Γ é aplicado a um conjunto subestimado de literais verdadeiros segundo uma interpretação I. Como Γ é antimonotônico [28], o resultado será um conjunto (intermediário) superestimado de literais verdadeiros segundo I; no segundo estágio, Γ é aplicado a esse conjunto superestimado, obtendo novamente um conjunto subestimado. Esse processo continua até um ponto fixo ser alcançado. Da antimonotonicidade de Γ, obtém-se que a composição desses dois passos é monotônica, logo $\Gamma\Gamma$ possui um menor ponto fixo que representa o modelo bem fundado (WFM). Essa definição de WFM é demonstrada em [92] ser equivalente à original apresentada em [93].

Posteriormente, em [2, 5, 6], foi feita uma definição de $WFSX$ baseada em pontos fixos alternados dos operadores $\Gamma\Gamma_s$, sendo que Γ_s é o operador Γ aplicado a versão seminormal de um programa. Em [20], é feita uma generalização dessa definição para o caso paraconsistente, obtendo $WFSX_P$.

Nesta subseção, é elaborada uma definição para $WFSX_{EI}$ baseada em pontos fixos alternados que segue muito proximamente à apresentada para $WFSX_P$ [20]. As diferenças que surgem são por causa das novas relações decorrentes da introdução do operador "?".

Para tanto, inicialmente é feita uma redefinição do operador Γ. Na definição abaixo, é assumido que para todo átomo A, não há nenhuma relação existente entre A, $\neg A$, $A?$ e $\neg A?$. Assim, esses literais podem ser tratados simplesmente como se fossem átomos.

Além disso, é assumido que uma interpretação I é bivalorada e consistente, portanto I pode ser representada por um conjunto de átomos.

Definição 5.48 (Operador Γ) *Sejam P um programa em lógica da inconsistência epistêmica e I uma interpretação bivalorada. A transformação GL $\frac{P}{I}$ é o programa obtido de P da seguinte maneira:*

- *Remover todas as regras de P contendo um literal default not α tal que $\alpha \in I$.*
- *Remover todos os literais default restantes de P.*

Como $\frac{P}{I}$ é um programa em lógica definido, ele possui um menor modelo bivalorado, que é denotado por $\Gamma_P(I)$.

O menor modelo bivalorado de um programa em lógica definido pode ser obtido iteragindo o operador conseqüência imediata T_p de Van Emden e Kowalski até atingir o seu menor ponto fixo. Como esse operador é contínuo, o ponto fixo é alcançado em no máximo ω passos, sendo que ω é um limite ordinal. Conseqüentemente, $\Gamma_P(I) = T_{\frac{P}{I}}^{\uparrow\omega}$.

Exemplo 5.49 Seja P o programa abaixo:

$$P = \left\{ \begin{array}{llll} d \leftarrow c & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c? & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

e $I = \{a?, b?, c, c?, \neg c?\}$

Com isso, obtém-se que

$$\frac{P}{I} = \left\{ \begin{array}{lll} d \leftarrow c & b? & c \\ d? \leftarrow c? & \neg c? & c? \end{array} \right\}$$

Aplicando o operador T_P, obtém-se como resultados

$T_{\frac{P}{I}}^{\uparrow 0} = \{\}$
$T_{\frac{P}{I}}^{\uparrow 1} = \{b?, c, c?, \neg c?\}$
$T_{\frac{P}{I}}^{\uparrow 2} = \{b?, c, c?, \neg c?, d, d?\}$
$T_{\frac{P}{I}}^{\uparrow 3} = T_{\frac{P}{I}}^{\uparrow 2}$
$T_{\frac{P}{I}}^{\uparrow \omega} = T_{\frac{P}{I}}^{\uparrow 3} = \{b?, c, c?, \neg c?, d, d?\}$

Assim, o menor modelo bivalorado de $\frac{P}{I}$ é igual a $\{b?, c, c?, \neg c?, d, d?\}$.

Baseado em [2, 6, 20], para impor o princípio da coerência, é utilizada uma versão seminormal de um programa em lógica da inconsistência epistêmica.

Definição 5.50 (Versão seminormal de um programa) *A versão seminormal de um programa em lógica da inconsistência epistêmica P é o programa P_s obtido da seguinte maneira:*

- *Para cada regra $L \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ de P, adiciona-se ao seu corpo (possivelmente vazio) o literal default $not\ \neg L?$, em que $\neg L?$ é o complemento de $L?$ com relação a negação explícita;*
- *Para cada regra $L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ de P, adiciona-se ao seu corpo (possivelmente vazio) o literal default $not\ \neg L$, em que $\neg L$ é o complemento de L com relação a negação explícita.*

Exemplo 5.51 Sejam P e $I = \{a?, b?, c, c?, \neg c?\}$ respectivamente o programa e a interpretação apresentada no exemplo 5.49. A versão seminormal P_s de P é

$$P_s = \left\{ \begin{array}{lll} d \leftarrow c, not\ \neg d? & b? \leftarrow not\ d, not\ \neg b & c \leftarrow not\ \neg c? \\ d? \leftarrow c?, not\ \neg d & a? \leftarrow not\ a?, not\ \neg a & c? \leftarrow not\ \neg c \\ \neg c? \leftarrow not\ c & & \end{array} \right\}$$

Da definição 5.48, resulta que

$$\frac{P_s}{I} = \left\{ \begin{array}{llll} d? \leftarrow c? & d \leftarrow c & b? & c? \end{array} \right\}.$$

Aplicando o operador T_P, obtém-se

$T_{\frac{P_s}{I}}^{\uparrow 0} = \{\}$
$T_{\frac{P_s}{I}}^{\uparrow 1} = \{b?, c?\}$
$T_{\frac{P_s}{I}}^{\uparrow 2} = \{b?, c?, d?\}$
$T_{\frac{P_s}{I}}^{\uparrow 3} = T_{\frac{P_s}{I}}^{\uparrow 2}$
$T_{\frac{P_s}{I}}^{\uparrow \omega} = T_{\frac{P_s}{I}}^{\uparrow 3} = \{b?, c?, d?\}$

Assim, o menor modelo bivalorado de $\frac{P_s}{I}$ é igual a $\{b?, c?, d?\}$.

A versão seminormal de um programa introduz o operador $\Gamma_{P_s}(S)$. Doravante, por simplicidade será utilizado ΓS para denotar $\Gamma_P(S)$ e $\Gamma_s S$ para denotar $\Gamma_{P_s}(S)$.

Para definir WFM_{EI} com base na aplicação alternada dos operadores Γ e Γ_s, é demonstrada a validade do seguinte teorema:

Teorema 5.52 (Monotonicidade de $\Gamma\Gamma_s$) *O operador $\Gamma\Gamma_s$ é monotônico sob a ordem de inclusão de conjuntos para conjuntos arbitrários de literais objetivos e plausíveis.*

(*Prova*)

Provar a monotonicidade de $\Gamma\Gamma_s$ equivale a mostrar que dado dois conjuntos arbitrários A e B, $A \subseteq B \Rightarrow \Gamma\Gamma_s A \subseteq \Gamma\Gamma_s B$.

Suponha que $A \subseteq B$. Pela definição de transformação GL, quanto maior um conjunto I menor será o conjunto $\frac{P}{I}$, em que P é um programa em lógica da inconsistência epistêmica. Assim, $\frac{P}{B} \subseteq \frac{P}{A}$(resp. $\frac{P_s}{B} \subseteq \frac{P_s}{A}$). Como esses programas transformados não possuem literais default e as cláusulas Horn são monotônicas, segue-se que $\Gamma B \subseteq \Gamma A$(resp. $\Gamma_s B \subseteq \Gamma_s A$). Isso prova a anti-monotonicidade de Γ e de Γ_s.

Assim, se $A \subseteq B$, por anti-monotonicidade de Γ_s, $\Gamma_s B \subseteq \Gamma_s A$ e por anti-monotonicidade de Γ, $\Gamma\Gamma_s A \subseteq \Gamma\Gamma_s B$. ◊

Desse modo, todo programa possui um menor ponto fixo de $\Gamma\Gamma_s$. No próximo teorema será provada a equivalência entre a definição de $WFSX_{EI}$ baseada em ω^{ei} e uma definição baseada em Γ e Γ_s:

Teorema 5.53 (Descrição de ω^{ei} em termos de Γ e Γ_s) *Seja P um programa em lógica da inconsistência epistêmica e $I = T \cup not\ F$ uma interpretação para P. O operador ω^{ei} pode ser descrito por*

$$\omega^{ei}(T \cup notF) = \mathcal{T}(F) \cup not\ \mathcal{F}(T).$$

em que os operadores $\mathcal{T}$ e $\mathcal{F}$ são definidos pelas seguintes equações:

$$\mathcal{T}(\mathcal{F}) = -(\mathcal{B}_\mathcal{P} - \mathcal{F})\ e\ \mathcal{F}(T) = \mathcal{B}_P - \Gamma_s T.$$

(*Prova*)

Para provar este teorema, deve-se demonstrar as seguintes equivalências:

$$\alpha \in \theta_I^{ei\uparrow\omega} \text{ sss } \alpha \in T^{\uparrow\omega}_{\frac{P}{\mathcal{B}_P - F}} \text{ e } not\ \alpha \notin \theta_I^{ei\uparrow\omega} \text{ sss } \alpha \in T^{\uparrow\omega}_{\frac{P_s}{T}}.$$

Por indução, é demonstrada a primeira equivalência:

Base de indução: Trivial, desde que $\theta_I^{ei\uparrow 0} = not\ \mathcal{B}_P$ e como nenhum literal objetivo pertence a $not\ \mathcal{B}_P$, por vacuidade tem-se que $\alpha \in \theta_I^{ei\uparrow 0}$ sss $\alpha \in T^{\uparrow 0}_{\frac{P}{\mathcal{B}_P - F}} = \{\}$.

Passo de indução: Admite-se que $\alpha \in \theta_I^{ei\uparrow n}$ sss $\alpha \in T^{\uparrow n}_{\frac{P}{\mathcal{B}_P - F}}$. Os seguintes resultados são obtidos:
$\alpha \in \theta_I^{ei\uparrow n+1}$
sss há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P tal que para todo $i \leq k$, $not\ \beta_i \in I$ e para todo $j \leq m$, $\alpha_j \in \theta_I^{ei\uparrow n}$
sss há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P tal que para todo $i \leq k$, $\beta_i \notin (\mathcal{B}_P - F)$ e para todo $j \leq m$, $\alpha_j \in \theta_I^{ei\uparrow n}$

sss há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $\frac{P}{\mathcal{B}_{P-F}}$ tal que para todo $j \leq m$, $\alpha_j \in \theta_I^{ei\uparrow n}$
sss há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $\frac{P}{\mathcal{B}_{P-F}}$ tal que para todo $j \leq m$, $\alpha_j \in T^{\uparrow n}_{\frac{P}{\mathcal{B}_{P-F}}}$
sss $\alpha \in T^{\uparrow n+1}_{\frac{P}{\mathcal{B}_{P-F}}}$.

A prova da segunda equivalência também pode ser feita por indução:

Base de indução: Trivial, desde que $\theta_I^{ei\uparrow 0} = not\ \mathcal{B}_P$. Assim, tem-se que $not\ \alpha \notin \theta_I^{ei\uparrow 0}$ sss $\alpha \in T^{\uparrow 0}_{\frac{P_s}{T}} = \{\}$.

Passo de indução: Admite-se que $not\ \alpha \notin \theta_I^{ei\uparrow n}$ sss $\alpha \in T^{\uparrow n}_{\frac{P_s}{T}}$ para um certo $n > 0$. Os seguintes resultados são válidos:
$not\ \alpha \notin \theta_I^{ei\uparrow n+1}$
sss $\neg L? \notin I$ se α é um literal objetivo L (resp. $\neg L \notin I$ se α é um literal plausível $L?$) e existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_k$ em P tal que para todo $i \leq k$, $\beta_i \notin I$ e para todo $j \leq m$, $not\ \alpha_j \notin \theta_I^{ei\uparrow n}$
sss existe uma regra $L \leftarrow \alpha_1, \ldots \alpha_m, not\ \beta_1, \ldots, not\ \beta_k, not\ \neg L?$ (resp. $L? \leftarrow \alpha_1, \ldots \alpha_m$, $not\ \beta_1, \ldots, not\ \beta_k, not\ \neg L$) em P_s tal que $\neg L? \notin T$ (resp. $\neg L \notin T$) e para todo $i \leq k$, $\beta_i \notin I$ e para todo $j \leq m$, $not\ \alpha_j \notin \theta_I^{ei\uparrow n}$
sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $\frac{P_s}{T}$ tal que para todo $j \leq m$, $not\ \alpha_j \notin \theta_I^{ei\uparrow n}$
sss existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em $\frac{P_s}{T}$ tal que para todo $j \leq m$, $\alpha_j \in T^{\uparrow n}_{\frac{P_s}{T}}$
sss $\alpha \in T^{\uparrow n+1}_{\frac{P_s}{T}}$. ◊

Exemplo 5.54 Seja P o programa na forma canônica abaixo apresentado no exemplo 5.22. De acordo com o teorema 5.53, WFM_{EI} de P pode ser calculado como segue:

$$P = \left\{ \begin{array}{llll} d \leftarrow c, not\ \neg c? & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c?, not\ \neg c & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

$$\begin{array}{rcll} I_0 & = & \{\} & \\ & & \frac{P}{\mathcal{B}_P} = \{c; c?; \neg c?\} & \Gamma\mathcal{B}_P = \{c, c?, \neg c?\} \\ & & \frac{P_s}{\{\}} = \{d \leftarrow c; d? \leftarrow c?; c; c?; \neg c?; a?; b?\} & \Gamma_s\{\} = \{a?, b?, c, c?, \neg c?, d, d?\} \end{array}$$

$$
\begin{aligned}
I_1 &= \omega^{ei}(I_0) = \Gamma\mathcal{B}_P \cup not(\mathcal{B}_P - \Gamma_s\{\}) = \\
&\{c, c?, \neg c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, \neg c, \neg d, \neg d?\} \\
&T = \{c, c?, \neg c?\} \qquad \mathcal{B}_P - F = \{a?, b?, c, c?, \neg c?, d, d?\} \\
&\frac{P}{\mathcal{B}_P - F} = \{d? \leftarrow c?; c; c?; \neg c?\} \qquad \Gamma(\mathcal{B}_P - F) = \{c, c?, \neg c?, d?\} \\
&\frac{P_s}{T} = \{d? \leftarrow c?; b?; a?; c?\} \qquad \Gamma_s T = \{a?, b?, c?, d?\} \\
I_2 &= \omega^{ei}(I_1) = \Gamma(\mathcal{B}_P - F) \cup not(\mathcal{B}_P - \Gamma_s T) = \\
&\{c, c?, \neg c?, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
&T = \{c, c?, \neg c?, d?\} \qquad \mathcal{B}_P - F = \{a?, b?, c?, d?\} \\
&\frac{P}{\mathcal{B}_P - F} = \{d \leftarrow c; d? \leftarrow c?; b?; c; c?; \neg c?\} \\
&\Gamma(\mathcal{B}_P - F) = \{b?, c, c?, \neg c?, d, d?\} \\
&\frac{P_s}{T} = \{d? \leftarrow c?; b?; a?; c?\} \qquad \Gamma_s T = \{a?, b?, c?, d?\} \\
I_3 &= \omega^{ei}(I_2) = \Gamma(\mathcal{B}_P - F) \cup not(\mathcal{B}_P - \Gamma_s T) = \\
&\{b?, c, c?, \neg c?, d, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
&T = \{b?, c, c?, \neg c?, d, d?\} \qquad \mathcal{B}_P - F = \{a?, b?, c?, d?\} \\
&\frac{P}{\mathcal{B}_P - F} = \{d \leftarrow c; d? \leftarrow c?; b?; c; c?; \neg c?\} \\
&\Gamma(\mathcal{B}_P - F) = \{b?, c, c?, \neg c?, d, d?\} \\
&\frac{P_s}{T} = \{d? \leftarrow c?; a?; c?\} \qquad \Gamma_s T = \{a?, c?, d?\} \\
I_4 &= \omega^{ei}(I_3) = \Gamma(\mathcal{B}_P - F) \cup not(\mathcal{B}_P - \Gamma_s T) = \\
&\{b?, c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} \\
&T = \{b?, c, c?, \neg c?, d, d?\} \qquad \mathcal{B}_P - F = \{a?, c?, d?\} \\
&\frac{P}{\mathcal{B}_P - F} = \{d \leftarrow c; d? \leftarrow c?; b?; c; c?; \neg c?\} \\
&\Gamma(\mathcal{B}_P - F) = \{b?, c, c?, \neg c?, d, d?\} \\
&\frac{P_s}{T} = \{d? \leftarrow c?; a?; c?\} \qquad \Gamma_s T = \{a?, c?, d?\} \\
I_5 &= \omega^{ei}(I_4) = \Gamma(\mathcal{B}_P - F) \cup not(\mathcal{B}_P - \Gamma_s T) = \\
&\{b?, c, c?, \neg c?, d, d?\} \cup not\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\} = I_4
\end{aligned}
$$

Como pode ser verificado, essa seqüência de interpretações coincide com as apresentadas no exemplo 5.47. Do teorema 5.53, obtém-se os seguintes resultados:

Teorema 5.55 (Condições de ponto fixo para ω^{ei}) *Seja P um programa em lógica da inconsistência epistêmica e $I = T \cup not\ F$ uma interpretação na linguagem de P. I é um ponto fixo de ω^{ei} sss uma das seguintes condições (equivalentes) forem verificadas:*

- $T = \Gamma\Gamma_s T$ *e* $F = \mathcal{B}_P - \Gamma_s T$*;*
- $F = \mathcal{B}_P - \Gamma_s\Gamma(\mathcal{B}_P - F)$ *e* $T = \Gamma(\mathcal{B}_P - F)$*, que é equivalente a* $\mathcal{B}_P - F$ *ser um ponto fixo de* $\Gamma_s\Gamma$*.*

(*Prova*)

I é um ponto fixo de ω^{ei} sss $I = \omega^{ei}(I)$ sss $T \cup not\ F = \mathcal{T}(F) \cup not\ \mathcal{F}(T)$, ou seja,

$$T = \Gamma(\mathcal{B}_P - F) \text{ e } F = \mathcal{B}_P - \Gamma_s T. \tag{5.1}$$

Substituindo F na primeira equação de 5.1, tem-se $T = \Gamma(\mathcal{B}_P - (\mathcal{B}_P - \Gamma_s T)) = \Gamma(\Gamma_s T)$ $= \Gamma\Gamma_s T$ e $F = \mathcal{B}_P - \Gamma_s T$

Por outro lado, substituindo T na segunda equação de 5.1, tem-se

$$F = \mathcal{B}_P - \Gamma_s\Gamma(\mathcal{B}_P - F) \text{ e } T = \Gamma(\mathcal{B}_P - F). \tag{5.2}$$

Da equação 5.2, resulta que $\mathcal{B}_P - F = \Gamma_s\Gamma(\mathcal{B}_P - F)$ e $T = \Gamma(\mathcal{B}_P - F)$, ou seja, $\mathcal{B}_P - F$ é um ponto fixo de $\Gamma_s\Gamma$. ◊

Corolário 5.56 (Equivalência de definições $WFSX_{EI}$**)** *Seja P um programa em lógica da inconsistência epistêmica na forma canônica. A interpretação* $I = T \cup not\ F$ *é um ponto fixo de* Φ_P^{ei} *sss* $T = \Gamma\Gamma_s T$ *e* $F = \mathcal{B}_P - \Gamma_s T$*.*

(*Prova*)

Segue imediatamente dos teoremas 5.43 e 5.53. ◊

No próximo teorema será demonstrado que não é necessário restringir um programa a sua forma canônica para definir WFM_{EI} usando o operador ω^{ei} ou os pontos fixos de $\Gamma\Gamma_s$. Para atingir esse objetivo, é provado que os pontos fixos de $\Gamma\Gamma_s$ são preservados sob a transformação canônica, seguindo um resultado apresentado em [2, 20].

Teorema 5.57 (Preservação dos pontos fixos sob a transformação canônica) *Seja P um programa em lógica da inconsistência epistêmica e* P^c *o seu programa canônico associado. Então T é um ponto fixo de* $\Gamma_P\Gamma_{P_s}$ *sss T é um ponto fixo de* $\Gamma_{P^c}\Gamma_{P_s^c}$*.*

Na prova deste teorema, são utilizados os seguintes lemas:

Lema 5.58 *Para todo T,* $\Gamma_{P_s} T = \Gamma_{P_s^c} T$*.*

(*Prova*)

Por definição, $\frac{P_s^c}{T} \subseteq \frac{P_s}{T}$ para todo T. Assim, se uma regra $\alpha \leftarrow \alpha_1 \ldots \alpha_m \in \frac{P_s^c}{T}$, ela também pertence a $\frac{P_s}{T}$. Seja $\beta \leftarrow \beta_1, \ldots, \beta_m$, uma regra de $\frac{P_s}{T}$ que não pertence a $\frac{P_s^c}{T}$. Então, existe $i \leq m$ tal que $\neg L_i? \in T$ se β_i é um literal objetivo L_i (resp. $\neg L_i \in T$ se β_i é um literal plausível $L_i?$). No entanto, se $\neg L_i? \in T$ (resp. $\neg L_i \in T$), pode-se remover a regra acima de $\frac{P_s}{T}$, pois pela transformação seminormal, não há regra para L_i (resp. $L_i?$) em $\frac{P_s}{T}$. Com isso, ambos esses programas concluem os mesmos literais. ◊

Lema 5.59 *Se* $\Gamma_P\Gamma_{P_s}T \neq \Gamma_{P^c}\Gamma_{P_s^c}T$, *então existe uma regra* $\alpha \leftarrow \alpha_1, \ldots, \alpha_m \in \frac{P}{\Gamma_{P_s}T}$ *tal que*

$$\alpha \in \Gamma_P\Gamma_{P_s}T \wedge \alpha \notin \Gamma_{P^c}\Gamma_{P_s^c} \wedge \forall_i[\alpha_i \in \Gamma_P\Gamma_{P_s}T \wedge \alpha_i \in \Gamma_{P^c}\Gamma_{P_s^c}T].$$

(*Prova*)

Sabe-se que $\frac{P_s^c}{\Gamma_{P_s^c}T} \subseteq \frac{P_s}{\Gamma_{P_s}T}$ e pelo lema 5.58, $\Gamma_{P_s}T = \Gamma_{P_s^c}T$. Assim, se $\Gamma_P\Gamma_{P_s}T \neq \Gamma_{P^c}\Gamma_{P_s^c}T$, então $\Gamma_{P^c}\Gamma_{P_s^c}T \subset \Gamma_P\Gamma_{P_s}T$.

Por indução sobre a construção de $\Gamma_P\Gamma_{P_s}T$ e $\Gamma_{P^c}\Gamma_{P_s^c}T$ com o operador T_P, será provado esse lema para literais que estão em $\Gamma_P\Gamma_{P_s}T$, mas não estão em $\Gamma_{P^c}\Gamma_{P_s^c}T$.

Base de indução: Se existe um fato para α em $\frac{P}{\Gamma_{P_s}T}$, ele também pertence a $\frac{P^c}{\Gamma_{P_s^c}T}$. Assim, ambas as divisões de programa têm o mesmo conjunto de fatos e portanto $T^{\uparrow 1}_{\frac{P}{\Gamma_{P_s}T}} = T^{\uparrow 1}_{\frac{P^c}{\Gamma_{P_s^c}T}}$.

Agora, considere que os literais verdadeiros na segunda aplicação do operador de van Emdem-Kowaslki. Se há um literal α que pertence a $T^{\uparrow 2}_{\frac{P}{\Gamma_{P_s}T}}$ e não pertence a $T^{\uparrow 2}_{\frac{P^c}{\Gamma_{P_s^c}T}}$, então existe uma regra $\alpha \leftarrow \alpha_1 \ldots \alpha_m \in \frac{P}{\Gamma_{P_s}T}$ tal que todo α_i, com $1 \leq i \leq m$, é um fato. Conseqüentemente, todo α_1 pertence a $T^{\uparrow 1}_{\frac{P}{\Gamma_{P_s}T}}$, que é igual a $T^{\uparrow 1}_{\frac{P^c}{\Gamma_{P_s^c}T}}$, demonstrando a base de indução.

Passo de indução: Suponha que o resultado é verificado para literais que aparecem na iésima aplicação do operador T_p com $2 \leq i \leq n$. Agora, suponha que $\alpha \in T^{\uparrow n+1}\frac{P}{\Gamma_{P_s}T}$ e $\alpha \notin T^{\uparrow n+1}_{\frac{P^c}{\Gamma_{P_s^c}T}}$.

Portanto, existe uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m \in \frac{P}{\Gamma_{P_s}T}$ com $\forall \alpha_i \in T^{\uparrow n}\frac{P}{\Gamma_{P_s}T}$.

Com isso, se $\forall i \alpha_i \in T^{\uparrow n}_{\frac{P^c}{\Gamma_{P_s^c}T}}$ o resultado é imediato; por sua vez, se $\exists i \alpha_i \notin T^{\uparrow n}_{\frac{P^c}{\Gamma_{P_s^c}T}}$, o resultado é obtido por indução. ◊

Prova do teorema 5.57: Por contradição, tem-se a seguinte prova:

($\Rightarrow$) Assume-se que $T = \Gamma_P\Gamma_{P_s}T$ e $T \neq \Gamma_{P^c}\Gamma_{P_s^c}T$. Pelo lema 5.59, sabe-se que há uma regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m \in \frac{P}{\Gamma_{P_s}T}$ tal que $\alpha \in T \wedge \alpha \notin \Gamma_{P^c}\Gamma_{P_s^c}T \wedge \forall i[\alpha_i \in T \wedge T \wedge \alpha_i \in \Gamma_{P^c}\Gamma_{P_s^c}T]$. Como $\alpha \notin \Gamma_{P^c}\Gamma_{P_s^c}T$, tem-se que $\exists j \neg L_j? \in \Gamma_{P_s^c}T$ se α_j é um literal objetivo L_j (resp. $\exists j \neg L_j \in \Gamma_{P_s^c}T$ se α_j é um literal plausível $L_j?$). No entanto, pelo lema 5.58, o literal $\neg L_j?$

(resp. $\neg L_j$) também pertence a $\Gamma_{P_s}T$ e com isso $\alpha_j \notin T$ pela transformação seminormal. Isso é uma contradição, já que por hipótese, $\forall i \alpha_i \in T$. Assim, $\Gamma_P\Gamma_{P_s}T = \Gamma_{P^c}\Gamma_{P_s^c}T$.

($\Leftarrow$) A prova da inversa é similar. $\Diamond$

Como os pontos fixos de $\Gamma\Gamma_s$ de um programa em lógica da inconsistência epistêmica são mantidos sob a tranformação para um programa canônico, a equivalência entre a definição original de $WFSX_{EI}$ e esta baseada em pontos fixos alternados é estabelecida para programas em lógica da inconsistência epistêmica arbitrários. Esse resultado é imediato.

Corolário 5.60 (XSM_{EI}, WFM_{EI} e $WFSX_{EI}$ como pontos fixos alternados) *Sejam P um programa em lógica da inconsistência epistêmica e T um ponto fixo de $\Gamma\Gamma_s$. Um modelo estável parcial estendido da inconsistência epistêmica de P ($XSM_{EI}(P)$) é igual a $T \cup not(\mathcal{B}_P - \Gamma_s T)$.*

O menor XSM_{EI} de P é chamado de modelo bem fundado da inconsistência epistêmica ($WFM_{EI}(P)$).

A semântica $WFSX_{EI}$ de P é determinada pelo conjunto de todo XSM_{EI} de P.

(*Prova*)

Segue diretamente da definição 5.36 e dos teoremas 5.43, 5.53 e 5.57. $\Diamond$

Exemplo 5.61 Seja P o programa abaixo:

$$P = \left\{ \begin{array}{ll} a? \leftarrow not\ b, not\ \neg a & c? \leftarrow b? \\ b? \leftarrow not\ a, not\ \neg b & c? \leftarrow a? \end{array} \right\}$$

Como esperado, esse programa possui os três modelos estáveis estendidos da inconsistência epistêmica apresentados pelo programa do exemplo 5.45.

$M_1 = not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?\}$;
$M_2 = \{a?, c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?\}$;
$M_3 = \{b?, c?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, \neg b?, c, \neg c, \neg c?\}$.
M_1 é o menor XSM_{EI} de P, portanto M_1 é o WFM_{EI} de P.

De acordo com a segunda condição do teorema 5.55, pode-se também utilizar o maior ponto fixo de $\Gamma_s\Gamma$ para caracterizar os modelos bem fundados da inconsistência epistêmica. Desse modo, se TU é o maior ponto fixo de $\Gamma_s\Gamma$ com relação a um programa em lógica da inconsistência epistêmica P, então $WFM_{EI}(P) = \Gamma(TU) \cup not(\mathcal{B}_P - TU)$. Alternativamente, o maior ponto fixo de $\Gamma\Gamma_s$ pode ser obtido através do menor ponto fixo de $\Gamma_s\Gamma$, fazendo os literais verdadeiros serem $\Gamma(lfp(\Gamma_s\Gamma))$ e os falsos serem $\mathcal{B}_P - lfp(\Gamma_s\Gamma)$.

Pelo teorema 5.52, a combinação de operadores $\Gamma\Gamma_s$ é monotônica, logo é garantido que existe um menor ponto fixo para cada programa em lógica da inconsistência epistêmica. Assim, $WFSX_{EI}$ pode ser recursivamente definido da seguinte maneira:

Teorema 5.62 *Para se obter uma definição iterativa construtiva de WFM_{EI}, é apresentada a seqüência transfinita de I_α abaixo:*

$$\begin{aligned} I_0 &= \{\} \\ I_{\alpha+1} &= \Gamma\Gamma_s I_\alpha \\ I_\delta &= \bigcup I_\alpha \mid \alpha < \delta \text{ para um limite ordinal } \delta \end{aligned}$$

Existe um menor ordinal λ para a seqüência acima tal que I_λ é o menor ponto fixo de $\Gamma\Gamma_s$ e $WFM_{EI}(P) = I_\lambda \cup not\ \mathcal{B}_P - \Gamma_s I_\lambda$.

Exemplo 5.63 Seja P o programa do exemplo 5.49 mostrado abaixo:

$$P = \left\{ \begin{array}{llll} d \leftarrow c & b? \leftarrow not\ d, not\ \neg b & c & \neg c? \\ d? \leftarrow c? & a? \leftarrow not\ a?, not\ \neg a & c? & \end{array} \right\}$$

$WFM_{EI}(P)$ pode ser calculado da seguinte maneira:

$$\begin{aligned} I_0 &= \{\} \\ I_1 &= \Gamma\Gamma_s I_0 = \Gamma\{a?, b?, c, c?, \neg c?, d, d?\} = \{c, c?, \neg c?, d, d?\} \\ I_2 &= \Gamma\Gamma_s I_1 = \Gamma\{a?, c?, d?\} = \{b?, c, c?, \neg c?, d, d?\} \\ I_3 &= \Gamma\Gamma_s I_2 = \Gamma\{a?, c?, d?\} = \{b?, c, c?, \neg c?, d, d?\} \end{aligned}$$

Com isso, $WFM_{EI}(P) = \{b?, c, c?, \neg c?, d, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$. Esse resultado é o mesmo apresentado nos exemplos 5.39 e 5.47.

5.3 Comparação entre $WFSX_{EI}$ e $WFSX_P$

No estudo de $WFSX_P$, foram identificados alguns resultados pouco intuitivos. De acordo com a origem, esses problemas podem ser agrupados em duas categorias.

- O primeiro grupo, designado por $\mathbf{G_1}$, é constituído por problemas decorrentes da inexistência de uma análise de consistência para conclusões obtidas de literais default;
- O segundo grupo, designado por $\mathbf{G_2}$, é formado por problemas resultantes da não obediência ao princípio da exceção primeiro em $WFSX_P$.

Baseado nos defaults *IDL-LEI*, a transformação T_1 impõe que toda regra com literais default no corpo seja do tipo $A? \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n, not\ \neg A$ com $m, n \leq 0$. Essa é uma forma de averiguar a consistência de conclusões obtidas de literais default, impedindo que problemas do tipo $\mathbf{G_1}$ ocorram. Nessa categoria, enquadra-se o paradoxo

do barbeiro na forma como é tratada em programas em lógica estendidos sob $WFSX_P$ (Exemplo 1.1). No tocante a $WFSX_{EI}$, por causa das imposições sintáticas para programas em lógica da inconsistência epistêmica garantidas por T_1, o resultado esperado de que o barbeiro não faz a sua própria barba é obtido.

Vale lembrar que o paradoxo do barbeiro pode ser compreendido a partir da seguinte declaração: *Se não há nenhuma evidência de que um indivíduo faz a sua barba, então o barbeiro faz a barba dele.* O paradoxo surge na tentativa de identificar quem faz a barba do barbeiro. Isso pode ser representado em programas em lógica da inconsistência epistêmica como segue:

Exemplo 5.64 (Paradoxo do barbeiro) Seja P o programa abaixo:

$$shave(b,x)? \leftarrow not\ shave(x,x)?, not\ \neg shave(b,x)^{16},$$

em que b é uma constante e x, uma variável. Além disso, seja P' o programa obtido de P acrescentando os fatos $\neg shave(b,b)$ e $\neg shave(b,b)?$:

$$P' = \begin{cases} (1)\ shave(b,x)? \leftarrow not\ shave(x,x)?, not\ \neg shave(b,x) \\ (2)\ \neg shave(b,b) \\ (3)\ \neg shave(b,b)? \end{cases}$$

Em $WFSX_{EI}(P')$, sabe-se da regra (1) de P', que o literal $shave(b,b)?$ é verdadeiro se $not\ shave(b,b)?$ e $not\ \neg shave(b,b)$ também forem. Da regra (2) $\neg shave(b,b)$, obtém-se $not\ shave(b,b)?$ pelo princípio da coerência. No entanto, $not\ \neg shave(b,b)$ não é verdadeiro por causa de $\neg shave(b,b)$, bloqueando a conclusão de $shave(b,b)?$ em (1). Conseqüentemente, em $WFSX_{EI}$, o resultado indesejado de que o barbeiro faz a sua própria barba em P' não é mais obtido.

O literal $not\ \neg shave(b,b)$ no corpo da regra (1) para $shave(b,b)?$ é uma imposição sintática da transformação T_1. Observe que através desse literal default é averiguada a consistência de $shave(b,b)?$ em P'. Isso quer dizer que em P', para concluir $shave(b,b)$ através de uma regra com literais defaults, $not\ \neg shave(b,b)$ deve pertencer a $WFM_{EI}(P')$ ou, o que dá no mesmo, $\neg shave(b,b)$ não deve pertencer a $WFM_{EI}(P')$. Como $\neg shave(b,b) \in WFM_{EI}(P')$, $shave(b.b)?$ não é conseqüência lógica de P'. Desse modo, os resultados pouco intuitivos obtidos pelo uso livre de literais defaults em $WFSX_P$ são eliminados.

[16] Observe que $shave(b,b)? \leftarrow not\ shave(b,b), not\ \neg shave(b,b)$ corresponde a declaração *se não há nenhuma evidência de que um indivíduo faz a sua barba, então o barbeiro faz a barba dele.*

Utilizando a definição 5.60 de $WFSX_{EI}$, tem-se que

$$\begin{aligned} I_0 &= \{\} \\ I_1 &= \Gamma\Gamma_s I_0 = \Gamma\{shave(b,b)?, \neg shave(b,b), \neg shave(b,b)?\} = \\ &\quad \{\neg shave(b,b), \neg shave(b,b)?\} \\ I_2 &= \Gamma\Gamma_s I_1 = \Gamma\{\neg shave(b,b), \neg shave(b,b)?\} = \{\neg shave(b,b), \neg shave(b,b)?\} \\ I_\delta &= I_2 \end{aligned}$$

$WFM_{EI}(P') = \{\neg shave(b,b), \neg shave(b,b)?\} \cup not\ \{shave(b,b), shave(b,b)?\}$.

Como argumentado por M. Pequeno em [65], o papel da exceção não vem sendo adequadamente tratado por muitos formalismos de raciocínio não monotônico, conduzindo a resultados que não correspondem ao esperado do senso comum.

Para solucionar esse problema, ainda em [65], é proposto que a derivação de exceções para regras não monotônicas deve ser privilegiada em relação a própria aplicação da regra, ou seja, se alguma regra não monotônica permitir a derivação de uma exceção para uma outra regra não monotônica, então a exceção deve ser derivada e conseqüentemente bloquear a aplicação da respectiva regra em vez de aplicar a regra e não derivar a exceção. Por essa razão, esse princípio é chamado de *princípio das exceções primeiro.*

Na lógica default de Reiter, a parte seminormal da justificativa de uma regra default R pode ser vista como a exceção para R. Por não estar de acordo com o princípio das exceções primeiro, em algumas situações a lógica default apresenta resultados pouco intuitivos.

Por sua vez, em *IDL-LEI*, o princípio das exceções primeiro é elegantemente inserido ao estabelecer que para bloquear um default $D = \frac{\alpha:\beta;\gamma}{\beta?}$ pela parte seminormal γ, uma contradição ($\neg\gamma?$), a exceção, é o suficiente enquanto que para bloquear D pela parte normal, uma contradição forte $\neg\beta$ é necessária. Com isso, qualquer default *IDL-LEI* D' que derive a exceção $\neg\gamma?$ bloqueia D enquanto que a conclusão de D, $\beta?$ não é o suficiente para bloquear D'; a não ser que $\beta?$ seja a exceção para D'.

Desde que qualquer programa em lógica estendido P, cujo modelo WFM_P não seja contraditório, é correto com relação a intersecção de todas as extensões da teoria default de Reiter correspondente, alguns problemas, os do tipo $\mathbf{G_2}$, são verificados. Na sequência, são elencados alguns problemas clássicos envolvendo formalismos não monotônicos que não estão de acordo com o princípio das exceções primeiro. Dentre eles, está o já conhecido problema apresentado por Morris (exemplos 1.2 e 1.3). Por ser baseado em *IDL-LEI*, $WFSX_{EI}$ apresenta soluções mais significativas do que $WFSX_P$ para esses problemas. Quando isto for conveniente nos exemplos abaixo, os literais serão simplesmente denotados em sua forma proposicional.

Exemplo 5.65 (O problema do pássaro canônico) Seja P o programa em lógica da inconsistência epistêmica para o problema apresentado por Morris [60].

$$P = \left\{ \begin{array}{lll} (1)\neg v? \leftarrow an, not\ v, not\ a? & (5)a? \leftarrow p, not\ \neg a & (9)\neg a \leftarrow \neg v \\ (2)\neg v? \leftarrow an?, not\ v, not\ a? & (6)a? \leftarrow p?, not\ \neg a & (10)\neg a? \leftarrow \neg v? \\ (3)v \leftarrow a & (7)an \leftarrow p & (11)p \\ (4)v? \leftarrow a? & (8)an? \leftarrow p? & (12)p? \end{array} \right.$$

em que a, an, p e v representam respectivamente os predicados alado, animal, pássaro e voa.

Intuitivamente, do fato que é pássaro, obtém-se que ele é animal pela regra (7). Da regra (1), que é uma imposição sintática da transformação T_1, conclui-se $\neg v?$. Em seguida, da regra (10), que é uma imposição sintática da transformação T_2 a partir da regra (9), conclui-se $\neg a?$, que não é o suficiente para bloquear a aplicação da regra (5) (seria necessário $\neg a$) e $a?$ é obtido. Com isso, $\neg v?$ não é mais veiculado pela regra (1), pois $a?$ bloqueia a aplicação dessa regra e também não mais se conclui $\neg a?$ pela regra (10). Finalmente, da regra (4), ainda se conclui $v?$. Assim, o resultado esperado que um certo pássaro é alado e voa é obtido. Em contrapartida, v e a são indefinidos em WFM_P (Exemplo 1.3).

Observe que na regra 1 de P, ser alado é uma exceção à regra geral que animais normalmente não voam. Desse modo, pelo princípio das exceções primeiro, a conclusão de $a?$ através da regra 6 de P impede a conclusão de $\neg v?$ pela regra 1. Pela definição 5.60, obtém-se os seguintes resultados:

$$\begin{array}{rcl} I_0 & = & \{\} \\ I_1 & = & \Gamma\Gamma_s I_0 = \Gamma\{p, p?, an, an?, \neg v?, a?, \neg a?, v?\} = \{p, p?, an, an?, a?, v?\} \\ I_2 & = & \Gamma\Gamma_s I_1 = \Gamma\{p, p?, an, an?, a?, v?\} = \{p, p?, an, an?, a?, v?\} \\ I_\delta & = & I_2 \end{array}$$

Assim, $WFM_{EI}(P) = \{p, p?, an, an?, a?, v?\} \cup not\ \{\neg p, \neg p?, \neg an, \neg an?, a, \neg a, \neg a?, v\ \neg v, \neg v?\}$.

Exemplo 5.66 (O problema do pássaro) Nas representações abaixo, p e v denotam respectivamente pingüim e voa.

Lógica Default de Reiter

$W = \emptyset$

$D = \left\{ \frac{:p}{p} \quad \frac{:\neg p}{\neg p} \quad \frac{:v \wedge \neg p}{v} \right\}$

Essa teoria possui duas extensões:

$E_1 = \{p\}$;

$E_2 = \{\neg p, v\}$.

IDL-LEI

$W = \emptyset$

$D = \left\{ \frac{: p}{p?} \quad \frac{: \neg p}{\neg p?} \quad \frac{: v; \neg p}{v?} \right\}$

Essa teoria possui uma única extensão: $E = \{p?, \neg p?\}$.

Programação em lógica estendida

$$P = \begin{cases} p \leftarrow not\ \neg p \\ \neg p \leftarrow not\ p \\ v \leftarrow not\ \neg v, not\ p \end{cases}$$

$WFM_P(P) = not\ \neg v$.

Programação em lógica da inconsistência epistêmica

$$P' = \begin{cases} p? \leftarrow not\ \neg p \\ \neg p? \leftarrow not\ p \\ v? \leftarrow not\ p?, not\ \neg v \end{cases}$$

$WFM_{EI}(P') = \{p?, \neg p?\} \cup not\{p, \neg p, v, \neg v, v?, \neg v?\}$.

Na lógica default de Reiter, as conclusões contraditórias são divididas em três extensões. Quanto às extensões *IDL-LEI* e a $WFSX_{EI}$, desde que estão de acordo com o princípio da exceção primeiro, é dado prioridade à conclusão de $p?$, bloqueando a conclusão da regra para $v?$. Como $p?$ não bloqueia a regra para $\neg p?$ e vice-versa, ambas essas conclusões são mantidas em *IDL-LEI* e $WFSX_{EI}$. Em contrapartida, no tocante a $WFSX_P$, cada regra para um literal bloqueia a conclusão do complemento e vice-versa, resultando num modelo com os literais p, $\neg p$ e v sendo indefinidos.

Exemplo 5.67 (O diamante de Nixon) O problema do diamante de Nixon pode ser enunciado da seguinte maneira:

Sabe-se que

– *Quakers são normalmente pacifistas.*

– *Republicanos são normalmente não pacifistas.*

– *Nixon é um quaker e é um republicano.*

Baseado nessas informações, o que se pode concluir a respeito de Nixon. Para representar esse problema, nos formalismos abaixo, serão utilizados os símbolos r, q e p para respectivamente designar que Nixon é republicano, quaker e pacifista.

Lógica Default de Reiter

$$W = \{q; r\}$$
$$D = \left\{ \frac{r : \neg p}{\neg p} \quad \frac{q : p}{p} \right\}$$

Essa teoria possui duas extensões:

$E_1 = \{q, r, p\}$;

$E_2 = \{q, r\neg p\}$.

IDL-LEI

$$W = \{q; r\}$$
$$D = \left\{ \frac{r : \neg p}{\neg p?} \quad \frac{q : p}{p?} \right\}$$

Essa teoria possui uma única extensão: $E = \{q, r, q?, r?, \neg p?, p?\}$;

Programação em lógica estendida

$$P = \begin{cases} p \leftarrow q, not\ \neg p & q \\ \neg p \leftarrow r, not\ p & r \end{cases}$$

$WFM_P(P) = \{r, q\} \cup not\ \{\neg r, \neg q\}$.

Programação em lógica da inconsistência epistêmica

$$P' = \begin{cases} p? \leftarrow q, not\ \neg p & q \\ p? \leftarrow q?, not\ \neg p & q? \\ \neg p? \leftarrow r, not\ p & r \\ \neg p? \leftarrow r?, not\ p & r? \end{cases}$$

$WFM_{EI}(P') = \{q, q?, r, r?, p?, \neg p?\} \cup not\ \{\neg q, \neg q?, \neg r, \neg r?, p, \neg p\}$.

Nesse problema, não há nenhum tipo de preferência entre as conclusões de que Nixon é pacifista e de que Nixon é não pacifista. Na lógica default de Reiter, isso é tratado, dividindo esse conflito em duas extensões e em $WFSX_P$ é atribuído o valor indefinido para p e $\neg p$. Por sua vez, em *IDL-LEI* assim como em $WFSX_{EI}$, essas conclusões são mantidas paraconsistentemente até que evidências adicionais permitam decidir qual delas deve ser retirada.

Uma outra importante categoria de aplicações envolvendo raciocínio não monotônico diz respeito ao tratamento das ações. Tipicamente, quando uma ação é executada, os aspectos inalterados são bem maiores do que os aspectos que sofreram algum tipo de mudança. Essa idéia é chamada de *frame axiom* e o problema de formalizá-la é chamado de *frame problem*. Normalmente, é assumido por default que todos os aspectos da situação corrente permanecem inalterados, exceto aqueles que são explicitamente mudados. Desse modo, os aspectos que sofrem alterações correspondem às exceções do *frame axiom*.

Em [65] é demonstrado que o princípio das exceções primeiro é particularmente adequado para lidar com o *frame problem*. Para ilustrar isso, é mostrado um dos mais famosos *frame problems*, o *Yale shoot problem* [37]. Por estarem de acordo com o princípio das exceções primeiro, $WFSX_{EI}$ e *IDL-LEI* fornecem respostas mais significativas do que respectivamente $WFSX_P$ e a lógica default de Reiter.

Exemplo 5.68 (Yale shooting problem) O problema abaixo foi apresentado em [37] e é conhecido como *Yale shooting problem*.

Numa certa situação S_0, uma pessoa está viva e a seguinte seqüência de eventos ocorre:

1. A arma é carregada;
2. Espera-se um instante;
3. Atira-se na pessoa com a arma.

Nesse problema, os únicos fatos são que a arma está carregada ou não carregada e que a pessoa está viva ou não viva. Carregar a arma, torna o fato de que a arma está carregada verdadeiro e é irrelevante para o fato da pessoa estar viva ou não. O evento de esperar um instante é irrelevante para todos os fatos. Finalmente, o evento de atirar numa pessoa com a arma carregada provoca que essa pessoa não esteja viva e é irrelevante para o fato de arma está carregada ou não.

Na representação desse problema serão utilizadas algumas convenções:

- Predicados e funções
  - O predicado $per(P, S)$ expressa que a propriedade P persiste na situação S;

  - O predicado $nor(P, E, S)$ expressa que na situação S, o evento E normalmente não afeta o valor verdade da propriedade P;
  - A função $res(E, S)$ designa a situação resultante da ocorrência do evento E na situação S.

- Eventos
  - carregar: evento de carregar a arma;
  - esperar: evento de esperar um instante;
  - atirar: evento de atirar com a arma na pessoa.

- Fatos
  - vivo: a pessoa está viva;
  - carregado: a arma está carregada.

- Situações
  - S_0: a situação inicial;
  - S_1: $= res(carregar, S_0)$;
  - S_2: $= res(esperar, S_1)$;
  - S_3: $= res(atirar, S_2)$;
  - S para representar uma situação arbitrária.

Feito isso, são apresentadas as formalizações para o *Yale shooting problem* na lógica default de Reiter, *IDL-LEI*, programação em lógica estendida utilizando $WFSX_P$ e programação em lógica da inconsistência epistêmica utilizando $WFSX_{EI}$; ressaltando que o resultado desejado é que em S_3 a pessoa não esteja viva.

Lógica Default de Reiter

(1) $per(vivo, S_0)$

(2) $per(carregado, res(carregar, S))$

(3) $per(carregado, S) \Rightarrow \neg nor(vivo, atirar, S)$

(4) $per(carregado, S) \Rightarrow \neg per(vivo, res(atirar, S))$

(5) $\dfrac{per(P, S) : per(P, res(E, S)), nor(P, E, S)}{per(P, res(E, S))}$

Essa teoria se divide em duas extensões: numa, obtém-se o resultado esperado que $\neg per(vivo, S_3)$ é verdadeiro; entretanto, noutra extensão, conclui-se que $\neg per(vivo, S_3)$ é

falso e por *modus tolens* sobre (4), $\neg per(carregado, S_2)$. Esse resultado é estranho, pois a arma está carregada em S_1 e o evento *esperar* é irrelevante para *carregado*.

IDL-LEI

(1) $per(vivo, S_0)$

(2) $per(carregado, res(carregar, S))$

(3) $per(carregado, S) \Rightarrow \neg nor(vivo, atirar, S)$

(4) $per(carregado, S) \Rightarrow \neg per(vivo, res(atirar, S))$

(5) $\dfrac{per(P, S) : per(P, res(E, S)), nor(P, E, S)}{per(P, res(E, S))?}$

Em *IDL-LEI*, de $per(vivo, S_3)?$, obtém-se $\neg per(carregado, S_2)?$ por *modus tolens* sobre (4). Entretanto, esse resultado não é suficiente para bloquear a aplicação da regra (5) (seria necessário $\neg per(carregado, S_2)$) e, assim, conclui-se que $per(carregado, S_2)?$ é verdadeiro. De (4), tem-se o resultado esperado $\neg per(vivo, S_3)?$. Esse último resultado bloqueia a conclusão de $per(vivo, S_3)?$ pela regra (5). Com isso, também não mais se conclui $\neg per(carregado, S_2)?$ de (4).

Programação em lógica estendida

$$P = \begin{cases} per(vivo, S_0) \\ per(carregado, res(carregar, S)) \\ \neg nor(vivo, atirar, S) \leftarrow per(carregado, S) \\ \neg per(vivo, res(atirar, S)) \leftarrow per(carregado, S) \\ \neg per(carregado, S) \leftarrow per(vivo, res(atirar, S)) \\ per(P, res(E, S)) \leftarrow per(P, S), not\ \neg per(P, res(E, S)), not\ \neg nor(P, E, S) \end{cases}$$

Programação em lógica estendida da inconsistência epistêmica[17]

$$P = \begin{cases} (1)\ per(vivo, S_0) \\ (2)\ per(carregado, res(carregar, S) \\ (3)\ \neg nor(vivo, atirar, S) \leftarrow per(carregado, S) \\ (4)\ \neg per(vivo, res(atirar, S)) \leftarrow per(carregado, S) \\ (5)\ \neg per(carregado, S) \leftarrow per(vivo, res(atirar, S)) \\ (6)\ per(P, res(E, S))? \leftarrow \\ \qquad\qquad per(P, S), not\ \neg per(P, res(E, S)), not\ \neg nor(P, E, S)? \end{cases}$$

[17] As regras geradas por T_2 foram omitidas.

Seguindo a seqüência de eventos pré-estabelecida, em $WFM_P(P)$, é atribuído o valor indefinido aos literais $per(carregado, S_2)$ e $\neg per(carregado, S_2)$, pois $per(carregado, S_2)$ bloqueia a conclusão de $\neg per(carregado, S_2)$ e vice-versa, resultando que $\neg per(vivo, S_3)$ é também indefinido.

Por sua vez, seguindo o raciocínio apresentado acima para *IDL-LEI*, em $WFM_{EI}(P)$, a conclusão de $\neg per(vivo, S_3)?$ pela regra (6) conduz a $\neg per(carregado, S_2)?$ através da regra gerada da transformação T_2 aplicada a regra (5). Entretanto, sabe-se que $\neg per(carregado, S_2)?$ não é o suficiente para impedir a conclusão de $per(carregado, S_2)?$ pela regra (6). Desse resultado, obtém-se $\neg per(vivo, S_3)?$ e $\neg nor(vivo, atirar, S_2)?$ através das regras resultantes da transformação T_2 aplicada respectivamente às regras (4) e (3). De $\neg nor(vivo, atirar, S_2)?$, não mais se conclui $\neg per(vivo, S_3)?$ pela regra (6). Por conseguinte, a conclusão de $\neg per(carregado, S_2)?$ é descartada. Desse modo, tanto em *IDL-LEI* quanto em $WFM_{EI}(P)$,, os resultados desejados $\neg per(vivo, S_3)?$ e $per(carregado, S_2)?$ são obtidos.

Uma semântica paraconsistente não deve somente ser capaz de resolver conflitos entre uma conclusão e outra. Ela também deve ser capaz de raciocinar significativamente diante de uma base de dados contraditória e de detectar literais dependentes de contradição. É mostrado no capítulo anterior que $WFSX_P$ e por similaridade $WFSX_{EI}$ não garantem para o caso geral a existência de modelos consistentes por default. Isso se deve ao princípio da coerência. Ademais, esses pares de literais contraditórios por default são usados para detectar os literais dependentes de contradição.

Exemplo 5.69 Considere o programa apresentado no exemplo 5.49:

$$P = \left\{ \begin{array}{llll} (1)\ d \leftarrow c & (3)\ b? \leftarrow not\ d, not\ \neg b & (5)\ c & (7)\ \neg c? \\ (2)\ d? \leftarrow c? & (4)\ a? \leftarrow not\ a?, not\ \neg a & (6)\ c? & \end{array} \right\}$$

Assim, $WFM_{EI}(P) = \{b?, c, c?, \neg c?, d, d?\} \cup not\ \{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$. Desse modelo, pode-se afirmar que

- $\{b?, c, c?, \neg c?, d.d?\}$ são verdadeiros;
- $\{a, \neg a, \neg a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?, d, \neg d, \neg d?\}$ são falsos por default;
- $a?$ é indefinido.

Observe no exemplo acima que de c (regra (5)), conclui-se d por (1). Todavia, de $\neg c?$ (regra (7)), obtém-se, pelo princípio da coerência, $not\ c$, que falsifica a única regra para d (regra (1)), ou seja, d também é falso ($not\ d \in WFM_{EI}(P)$). Um raciocínio análogo é aplicado em (4) para concluir tanto $b?$ quanto $not\ b?$. Portanto, está-se diante de uma

situação em que um literal é simultaneamente verdadeiro e falso por default. Disso, pode-se estabelecer que em $WFSX_{EI}$, um literal objetivo ou plausível α depende de informação contraditória se α e *not* α pertencem simultaneamente a WFM_{EI} e α não seje ele próprio contraditório. No exemplo 5.69, tem-se então que os literais c e $\neg c?$ são contraditórios e os literais $b?$ e d são dependentes de contradição.

Vale ainda ressaltar que o par de literais $c?$ e $\neg c?$ não caracterizam em programação em lógica da inconsistência epistêmica uma contradição. Conseqüentemente, o literal $d?$, que depende de $c?$ pela regra (2), possui valor lógico verdadeiro em WFM_{EI} e não depende de informação contraditória. Isso é caracterizado pelo literal $d?$ pertencer a $WFM_{EI}(P)$ e *not* $d?$ não pertencer a $WFM_{EI}(P)$.

5.4 Corretude de $WFSX_{EI}$ em relação a *IDL-LEI*

Estudos relacionando programas em lógica e teorias defaults têm sido bastante profícuos na revelação de nichos de pesquisa para ambos os formalismos. Através de uma teoria default apropriada, pode-se, de um lado, atribuir semântica a programas em lógica, clarificando tanto os significados da negação explícita quanto da negação default presentes nesses programas. Por outro lado, muitas dessas semânticas são definíveis a partir de operadores monotônicos. Conseqüentemente, essas semânticas possuem propriedades computacionais desejáveis tais como a existência de procedimentos construtivos (*bottom-up*) e de procedimentos derivativos (*top-down*), que podem ser utilizados no cálculo das extensões de uma teoria default.

Para estabelecer um relacionamento entre programas em lógica e teorias defaults, a idéia central é traduzir cada regra do programa numa regra default e então comparar as extensões da teoria default com a semântica do programa correspondente. Em [10], é mostrado que a semântica dos modelos estáveis é um caso especial das teorias defaults no sentido de Reiter. Em [29], esse resultado é generalizado para programas em lógica estendidos e semântica dos conjuntos resposta. Posteriormente, é mostrado que *WFSX* é correto com relação a intersecção de todas as extensões no sentido de Reiter para a teoria default correspondente quando essas extensões existirem.

De um modo similar, nesta seção, será demonstrado um resultado de corretude envolvendo $WFSX_{EI}$ e extensões *IDL-LEI*. Como já é sabido, regras de um programa em lógica da inconsistência epistêmica que possuem literais defaults podem ser traduzidas em defaults no sentido de *IDL-LEI*. Além disso, o conectivo "$\leftarrow$" não representa a implicação material, mas pode ser tratado como uma implicação default. Feitas essas considerações, é definida abaixo uma teoria default correspondente a um programa em lógica da inconsistência epistêmica:

Definição 5.70 (Teoria default correspondente a um programa) *Seja P um programa em lógica da inconsistência epistêmica. Diz-se que uma teoria default $\Delta = \langle W, D\rangle$ corresponde a P se e somente se*

- *Para cada regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ em P, com $m \geq 0$, existe um default $\frac{\alpha_1,\ldots,\alpha_m:}{\alpha}$ pertencente a W;*
- *Para cada regra $L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ b_1, \ldots, not\ b_n, not\ c_1?, \ldots, not\ c_p?, not\ \neg L$ em P, com $m, n, p \geq 0$, existe um default $\frac{\alpha_1,\ldots,\alpha_m:L,\neg b_1,\ldots,\neg b_n;\neg c_1,\ldots,\neg c_p}{L?}$ pertencente a D;*
- *Além dessas regras, nenhuma outra pertence a Δ.*

A teoria default $\Delta = \langle W, D\rangle$ correspondente a um programa em lógica da inconsistência epistêmica pode então ser vista como uma teoria default em que a base monotônica (W) é constituída por defaults (no sentido de Reiter) com a justificativa vazia e a base não monotônica (D), por defaults no sentido de *IDL-LEI*. Uma extensão para essa nova teoria pode ser calculada como segue:

Definição 5.71 (Extensão) *Sejam E um conjunto de literais objetivos ou plausíveis e $\Delta = \langle W, D\rangle$ uma teoria default correspondente a um programa em lógica da inconsistência epistêmica. Define-se*

$$
\begin{aligned}
E_0 &= \{\} \\
E_{i+1} &= E_i \cup \{\alpha \mid \frac{\alpha_1,\ldots,\alpha_m:}{\alpha} \in W \text{ e } \{\alpha_1, \ldots, \alpha_m\} \subseteq E_i\} \cup \\
&\quad \{L? \mid \frac{\alpha_1,\ldots,\alpha_m:L,\neg b_1,\ldots,\neg b_n;\neg c_1,\ldots,\neg c_p}{L?} \in D\}, \{\alpha_1, \ldots, \alpha_m\} \subseteq E_i, \\
&\quad \{\neg L, b_1, \ldots, b_n, c_1?, \ldots, c_p?\} \cap \textstyle\bigcup_{i=0}^{\infty} E_i = \{\}\}
\end{aligned}
$$

E é uma extensão de Δ sss $E = \bigcup_{i=0}^{\infty} E_i$.

Essa teoria Δ é *contraditória* se e somente se possui alguma extensão E tal que pelo menos um dos pares $\{L, \neg L\}$, $\{L?, \neg L\}$ e $\{L, \neg L?\}$ estão presentes em E.

Para provar a corretude de WFM_{EI} de um programa em lógica da inconsistência epistêmica P com relação à intersecção de todas as extensões da teoria default Δ correspondente a P, é apresentado o seguinte lema, que relaciona XSM_{EI} e as extensões de Δ.

Teorema 5.72 *Sejam P um programa em lógica da inconsistência epistêmica e $\Delta = \langle W, D\rangle$ uma teoria default correspondente a P. Se Δ possui extensão e além disso, é não contraditória, então toda extensão de Δ é igual a um modelo estável estendido da inconsistência epistêmica de P ($XSM_{EI}(P)$).*

(*Prova*)

Para provar este lema, vai-se demonstrar que se E é uma extensão de Δ, então $\Gamma(E) = E$ e $\Gamma_s(E) = E$, em que Γ e Γ_s são os operadores anti-monotônicos utilizados na definição de $WFSX_{EI}$ baseada em pontos fixos alternados (definição 5.60).

- Se E é uma extensão de Δ, $\Gamma_s(E) = E$.

  Se E é uma extensão de Δ, sabe-se que

  - Para cada default $d \in W$, d é do tipo $\frac{\alpha_1,...,\alpha_m:}{\alpha}$. Como E é uma extensão de Δ, se $\{\alpha_1 \ldots, \alpha_m\} \subseteq E$, então $\alpha \in E$. Da definição 5.70 e desde que Δ não é contraditório, sabe-se que existe em P_s uma regra do tipo $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \neg L?$ caso α seja um literal objetivo L (resp. $\alpha \leftarrow \alpha_1, \ldots, \alpha_m, not\ \neg L$ caso α seja um literal plausível $L?$), em que se $\{\alpha_1, \ldots, \alpha_n\} \subseteq E$ e $\neg L? \notin E$ (resp. $\{\alpha_1, \ldots, \alpha_n\} \subseteq E$ e $\neg L \notin E$), então $\alpha \in E$;
  - Para cada default $d \in D$, d é do tipo $\frac{\alpha_1,...,\alpha_m:\ L,\neg b_1,...,\neg b_n;\neg c_1,...,\neg c_p}{L?}$. Como E é uma extensão de Δ, se $\{\alpha_1, \ldots, \alpha_m\} \subseteq E$ e $\{\neg L, b_1, \ldots, b_n, c_1?, \ldots, c_p?\} \cap E = \{\}$, então $L? \in E$. Da definição 5.70, sabe-se que existe em P_s uma regra do tipo $L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ b_1, \ldots, not\ b_n, not\ c_1?, \ldots, not\ c_p?, not\ \neg L$, em que se $\{\alpha_1, \ldots, \alpha_m\} \subseteq E$ e $\{b_1, \ldots, b_n, c_1?, \ldots, c_p?, \neg L\} \cap E = \{\}$, então $L? \in E$.

  Desse resultado e da definição do operador Γ_s (definição 5.48), finalmente, obtém-se que $\Gamma_s(E) = E$.

- Se E é uma extensão de Δ, então $\Gamma(E) = E$.

  Esta demonstração é similar ao caso acima para Γ_s.

Desse modo, se E é uma extensão de Δ, então $\Gamma(E) = E$ e $\Gamma_s(E) = E$. Conseqüentemente, $\Gamma\Gamma_s(E) = E$, ou seja, E é um XSM_{EI} de P. ◊

Teorema 5.73 (Corretude) *Sejam P um programa em lógica da inconsistência epistêmica e Δ a teoria default correspondente a P. Se Δ possui extensão, então para todo literal objetivo ou plausível α pertencente a $WFM_{EI}(P)$, α também pertence a intersecção de todas as extensões de Δ.*

(*Prova*)

Quando Δ é uma teoria contraditória, essa demonstração é trivial. Nos demais casos, tem-se que por definição, se $\alpha \in WFM_{EI}(P)$, então α pertence a todo $XSM_{EI}(P)$. Em

particular, pelo teorema 5.72, obtém-se que α pertence a todo $XSM_{EI}(P) = E$ para toda extensão E de Δ. Então, α pertence a intersecção de todas as extensões (se existir pelo menos uma) de Δ. $\Diamond$

Na definição 5.70, cada regra $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$ de P é traduzida num default de Reiter. Em vez disso, se tais regras fossem convertidas em fórmulas *LEI* do tipo $\alpha_1 \ldots, \alpha_m \Rightarrow \alpha$, a teoria correspondente seria uma teoria *IDL-LEI*. Entretanto, por traduzir a implicação $\leftarrow$ como sendo a implicação material $\Leftarrow$, o resultado de corretude obtido pelo teorema 5.73 não seria mais válido:

Exemplo 5.74 Sejam P o programa em lógica da inconsistência epistêmica

$$P = \left\{ \begin{array}{ll} c? \leftarrow not\ \neg c, not\ \neg b & \neg c? \leftarrow b? \\ \neg c \leftarrow b & \neg a? \\ \neg a & \end{array} \right\}$$

e $\Delta = \langle W, D \rangle$ a teoria *IDL-LEI* em que

$$W = \left\{ \begin{array}{ll} b \Rightarrow \neg c & b? \Rightarrow \neg c? \\ \neg a & \neg a? \end{array} \right. \quad \text{e} \quad D = \frac{: c;b}{c?}.$$

$WFM_{EI}(P) = \{\neg a, \neg a?, c?\} \cup not\{a, a?, b, \neg b, b?, \neg b?, c, \neg c, \neg c?\}$. Pela definição A.20, a única extensão E para Δ é $E = C_D(W)$. Com isso, tem-se que $c? \in WFM_{EI}(P)$, mas $c? \notin E$, contrariando o resultado da corretude entre $WFM_{EI}(P)$ e a intersecção de todas as extensões da teoria *IDL-LEI* correspondente a P.

5.5 Resultados de complexidade

Um aspecto fundamental a ser levado em conta é o custo computacional de $WFSX_{EI}$. Em [93] é demonstrado que a complexidade do problema de decisão em *WFS* para programas sem símbolos funcionais (Datalog) é polinomial. Posteriormente, em [2] é provado que *WFSX* e conseqüentemente $WFSX_P$ também são polinomiais. Nesta sub-seção é mostrado que o mesmo pode ser dito a cerca de $WFSX_{EI}$.

Teorema 5.75 *O problema de decisão para qualquer programa em lógica da inconsistência epistêmica P sem símbolos funcionais é polinimial no tamanho da versão básica de P.*

(*Prova*)

Essa prova segue muito proxima à demonstração sobre a complexidade de *WFSX* [2] e *WFX* [93].

Deve-se mostrar que o modelo bem fundado da inconsistência epistêmica pode ser construído em tempo polinomial. Para tanto, será utilizada a definição de WFM_{EI} apresentada pelo corolário 5.60.

Seja P' um programa em lógica da inconsistência epistêmica. P' deve estar de acordo com alguns princípios estabelecidos em *IDL-LEI*. Caso não esteja, P' deve passar pelas transformações T_1 e T_2. Pela transformação T_1, cada regra de $L \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ de P' com $m \geq 0$ e $n > 0$ é substituída por $L? \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n, not\ \neg L$. Quanto à transformação T_2, para cada regra $\alpha \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ pertencente ao programa P', é acrescentada a regra $\alpha? \leftarrow \beta_1?, not\ \gamma_1, \ldots, not\ \gamma_n$ se $m = 1$; caso contrário, $\alpha? \leftarrow \beta_1, \ldots, \beta_m, not\ \gamma_1, \ldots, not\ \gamma_n$ é acrescentada a P'. Em todos esses casos, percebe-se facilmente que tanto a computação de T_1 quanto a dee T_2 podem ser feitas em tempo polinomial no tamanho de $\mathcal{B}_P$.

Assim, seja P o programa resultante e $WFM_{EI}(P) = T \cup not\ F$. Segundo o corolário 5.60, a parte positiva T é o menor ponto fixo do operador $\Gamma\Gamma_s$ enquanto que a parte negativa F é igual a $\mathcal{B}_P - \Gamma_s T$.

Até o ponto fixo ser alcançado, em qualquer estágio T_α da indução, pelo menos um elemento de $\mathcal{B}_P$ é adicionado a $T_{\alpha+1}$, logo o ponto fixo T deve ser alcançado em um número polinomial de passos. Assim, uma vez determinado T, F é também determinado em tempo polinomial. Com isso, somente resta provar que $\Gamma\Gamma_s T_\alpha$, pode ser encontrado em tempo polinomial.

Para isso, é suficiente mostrar que para qualquer conjunto C de literais objetivos e plausíveis, a computação de ΓC e $\Gamma_s C$ é polinomial. Desde que $\Gamma_s C$ é igual a ΓC aplicado a versão seminormal de um programa e claramente a versão seminormal é computável em tempo linear, basta mostrar que a computação de ΓC é polinomial:

- A computação de ΓC começa retirando todas as regras cujo corpo contenha um literal default $not\ A$ (resp. $not\ A?$) tal que $A? \in C$ (resp. $A \in C$). Claramente, essa computação é $O(|C| * |P|)$;

- Em seguida, todos os literais default restantes são retirados. Essa computação é $O(|P|)$;

- Finalmente, o operador T_P é aplicado ao programa resultante, que pode ser visto como um programa em lógica definido. É sabido que a computação de T_P para tais programas é polinomial.

Conseqüentemente, a computação de ΓC é polinomial. $\Diamond$

Capítulo 6

Semântica Operacional para Programas em Lógica Estendidos da Inconsistência Epistêmica

Na definição de um procedimento de refutação para programas em lógica da inconsistência epistêmica, aspectos concernentes ao princípio da coerência, à recursão infinita positiva e à recursão infinita através da negação default devem ser tratados. Uma abordagem desses aspectos é mostrada na seção 6.1 para programas em lógica da inconsistência epistêmica básicos e serve de motivação para a definição de SLX_{EI} na seção 6.2. Em seguida, na seção 6.3, é demonstrada que essa definição de SLX_{EI} é correta e completa com relação a $WFSX_{EI}$ para programas básicos. Finalmente, na seção 6.4, são discutidas algumas questões pertinentes ao problema da terminação em SLX_{EI}.

6.1 Introdução

Na definição de procedimentos de refutação para *WFS*, aspectos envolvendo recursões positivas infinitas e recursões infinitas através da negação por default devem ser considerados. O primeiro caso conduz ao valor verdade falso para os literais envolvidos (existe uma refutação para *not* L e nenhuma refutação para L em que L é um literal objetivo) e o segundo caso conduz ao valor verdade indefinido (não existe refutação para L nem para *not* L).

Partindo dessas ponderações, em [20, 3] é definido um procedimento de refutação para $WFSX_P$, chamado de *SLX* (*Selection rules driven Linear resolution for eXtended logic programs*). Além disso, em *SLX*, são introduzidos mecanismos que tratam do princípio da coerência. No resto, esse procedimento é similar a *SLDNF* [50], em que literais objetivos sem regras falham, $\leftarrow \Box$ é bem sucedido e literais resolvem com regras do programa mais

a regra da negação como falha.

Neste capítulo, é definido um procedimento de refutação para $WFSX_{EI}$, adaptando SLX para levar em conta os literais plausíveis e as transformações T_1 e T_2. Assim, baseado na definição de SLX, seguem as considerações abaixo:

Para programas em lógica da inconsistência epistêmica um exemplo simples de recursão positiva infinita é dado pela regra $\alpha \leftarrow \alpha$ e de recursão infinita através da negação por default, pela regra $L? \leftarrow not\ L?, not\ \neg L$. Em WFM_{EI}, no primeiro caso, obtém-se o valor falso para α e no segundo caso, obtém-se o valor indefinido para $L?$. Um procedimento de derivação para programas em lógica da inconsistência epistêmica deve ser capaz de lidar adequadamente com esses tipos de recursão para poder refletir os valores dos literais envolvidos.que haja uma refutação para $not\ L?$ (resp. $not\ L$); do mesmo modo, no segundo caso, obtém-se o valor verdade indefinido para $L?$, logo é desejável que não exista refutação para $L?$ nem para $not\ L$.

Por ser baseado em semânticas bivaloradas, em $SLDNF$, se não há refutação para L, então $not\ L$ é bem sucedido; caso haja uma refutação para L, então $not\ L$ falha. Desse modo, L e $not\ L$ não podem falhar simultaneamente. Entretanto, esse raciocínio não pode ser empregado em procedimentos de refutação para semânticas trivaloradas como a WFS, pois para essas semânticas, uma falha para provar L simplesmente significa que L não é verdadeiro, ou seja, L pode ser falso ou indefinido.

Para lidar com isso, em [80, 77], foram definidos procedimentos de derivação que consideram o *status* que é associado ao valor lógico indefinido. Literais envolvidos em recursão infinita através da negação default recebem esse *status*.

Em [3, 20], em vez de considerar esse *status* extra, são definidos dois tipos de árvores: árvores T[1], que provam a verdade em WFM_P, e árvores TU[2], que provam a não falsidade em WFM_P. Com a introdução do operador "?" em programas em lógica da inconsistência epistêmica, algumas adaptações com relação a definição de árvores T e TU devem ser feitas.

Além disso, um programa em lógica da inconsistência epistêmica P não precisa necessariamente estar de acordo com as retrições sintáticas impostas pelas transformações T_1 e T_2, pois as regras de expansão definidas abaixo para árvores T e TU simulam essas transformações (Confira lema 6.12)

Definição 6.1 (Árvores T) *Uma árvore T para um literal objetivo ou plausível α com respeito a um programa em lógica da inconsistência epistêmica básico P é uma árvore cuja raiz é rotulada com α e os os demais nós são obtidos aplicando sucessivamente as regras abaixo a partir da raiz. Na seqüência, L é um literal objetivo:*

[1]Em que T vem do inglês True (verdadeiro).

[2]Em que TU vem do inglês True (verdadeiro) ou Undefined (indefinido).

1. *Para um nó n rotulado com um literal objetivo L numa árvore T, seleciona-se a regra*

$$L \leftarrow \delta_1, \ldots, \delta_p$$

de P, em que todo δ_i, $1 \leq i \leq p$, é um literal objetivo ou plausível. Os sucessores de n são os nós rotulados respectivamente com $\delta_1, \ldots, \delta_p$. Em particular, se a regra para L é um fato, então o nó sucessor é rotulado com $\Box$;

2. *Para um nó n rotulado com um literal plausível L? numa árvore T, seleciona-se a regra*

$$L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$$

do programa P;

   (a) *Se $q = 0$, ou seja, não há literais default nessa regra, então os sucessores de n são os nós rotulados respectivamente com os literais $\delta_1, \ldots, \delta_p$. Além disso, se $p = 1$, o sucessor de n também pode ser rotulado com o literal δ_1?. Por fim, se $p, q = 0$, o nó sucessor de n é rotulado com $\Box$;*

   (b) *Se $q > 0$, ou seja, há literais default nessa regra, os sucessores de n são os nós rotulados respectivamente com $\delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots,$ not $\epsilon_q, not\ \neg L$. Se $q > 0$ e $p = 1$, os sucessores de n também podem ser rotulados respectivamente com $\delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$.*

3. *Para um nó n rotulado com um literal plausível L? numa árvore T, seleciona-se a regra*

$$L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$$

do programa P;

   (a) *Se $q = 0$ e $p \geq 2$, então os sucessores de n são rotulados respectivamente com os literais $\delta_1, \ldots, \delta_p$. Senão, se $q = 0$ e $p = 1$, o sucessor de n é o nó rotulado com o literal δ_1? e se $p, q = 0$, o nó sucessor de n é rotulado com $\Box$;*

   (b) *Se $q > 0$, os sucessores de n são os nós rotulados respectivamente com $\delta_1, \ldots,$ $\delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$. Se $q > 0$ e $p = 1$, os sucessores de n também podem ser rotulados com $\delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$.*

4. *Nos demais casos, os nós são folhas.*

Árvores TU, por sua vez, são definidas como segue:

Definição 6.2 (Árvores TU) *Similarmente à definição acima, uma árvore TU para um literal objetivo ou plausível α com respeito a um programa em lógica da inconsistência*

epistêmica básico P é uma árvore cuja raiz é rotulada com α e os demais nós são obtidos aplicando sucessivamente as regras abaixo a partir da raiz. Na seqüência, L é um literal objetivo.

1. *Para um nó n rotulado com um literal objetivo L numa árvore TU, tem-se as seguintes possibilidades:*
   (a) *Se existe uma árvore T bem sucedida para $\neg L?$, então L é o rótulo de um nó folha na árvore TU;*
   (b) *Caso contrário, utiliza-se o mesmo procedimento apresentado na condição 1 da definição 6.1.*
2. *Para um nó n rotulado com um literal objetivo L? numa árvore TU, tem-se as seguintes possibilidades:*
   (a) *Se existe uma árvore T bem sucedida para $\neg L$, então L? é o rótulo de um nó folha na árvore TU;*
   (b) *Caso contrário, utiliza-se o mesmo procedimento apresentado nas condições 2 e 3 da definição 6.1.*
3. *Nos demais casos, os nós são folhas.*

Como será justificado posteriormente no exemplo 6.7, essa diferença entre a expansão em uma árvore T e em uma árvore TU está relacionada a forma como o princípio da coerência é tratado em cada árvore. Em ambas as árvores, os nós folhas são rotulados com literais default, literais objetivos ou plausíveis sem regras para eles no programa ou ainda com a cláusula vazia $\Box$. Se um nó é rotulado com um literal objetivo ou plausível α, ele pode ser não deterministicamente expandido de várias diferentes maneiras, correspondendo às diferentes regras no programa para α. Na próxima definição, é mostrado como atribuir *status* aos nós das árvores T e TU.

Definição 6.3 (Sucesso e falha de árvores T e TU) *Cada nó numa árvore T (resp. árvore TU) tem um status associado que é falho ou bem sucedido. Todas as árvores infinitas são falhas. Uma árvore T (resp. árvore TU) finita é bem sucedida se sua raiz é bem sucedida e falha se sua raiz é falha. O status de um nó numa árvore finita é determinado de acordo com as seguintes regras:*

- *Um nó folha rotulado com $\Box$ é bem sucedido;*
- *Um nó folha rotulado com um literal objetivo ou plausível é falho;*

- *Um nó folha n numa árvore T (resp. árvore TU) rotulado com not α é bem sucedido sss todas as árvores TU (resp. árvores T) com raiz α, as árvores subsidiárias de n, são falhas; n é falho sss existe uma árvore TU (resp. árvore T) bem sucedida com raiz α;*

- *Um nó intermediário numa árvore T (resp. árvore TU) é bem sucedido sss todos os seus filhos são bem sucedidos e é falho sss pelo menos um de seus filhos é falho.*

Após aplicar essas regras, alguns nós podem ainda ter seus status indeterminados devido a recursão negativa infinita. Em árvores T é atribuído a esses nós o status falho e em árvores TU, o status bem sucedido.

Essa definição guarda muitas semelhanças com a definição de nós bem sucedidos e falhos em *SLDNF*. No entanto, uma diferença fundamental é que na definição 6.3, árvores infinitas são consideradas falhas. Com isso, a regra da negação como falha não é necessariamente finita. Esse conceito foi inicialmente introduzido na definição de *SLS* [80]. Em programação em lógica da inconsistência epistêmica, tal regra pode ser esboçada da seguinte maneira:

Definição 6.4 (Regra da negação como falha) *Seja L um literal objetivo. Um literal not α é bem sucedido sss toda derivação para α for infinita ou falha. Caso contrário not α é falho.*

Na definição de uma semântica operacional para um programa em lógica da inconsistência epistêmica P, a árvore T é utilizada para provar a verdade em WFM_{EI} e a árvore TU para provar a não falsidade em WFM_{EI}. Isso quer dizer que a árvore T deve ser bem sucedida para todos os literais que são verdadeiros em $WFM_{EI}(P)$ e deve ser falha para todos os literais que são indefinidos ou falsos em $WFM_{EI}(P)$; por sua vez, a árvore TU deve ser bem sucedida para todos os literais que são verdadeiros ou indefinidos em $WFM_{EI}(P)$ e falha para todos os literais que são falsos em $WFM_{EI}(P)$. Baseado nisso, na construção de procedimentos de prova para programas em lógica da inconsistência epistêmica sob a WFM_{EI}, devem ser consideradas as seguintes relações:

- α é verdadeiro em $WFM_{EI}(P)$ sss existe uma árvore T bem sucedida para α;

- α é falso em $WFM_{EI}(P)$ sss toda árvore TU para α é falha;

- Um literal α é indefinido sss toda árvore T para α é falha e existe uma árvore TU bem sucedida para α.

Desse modo, usando coordenadamente as árvores T e TU, é possível facilmente determinar o *status* de um nó, mesmo que ele esteja envolvido em recursão infinita positiva ou em recursão infinita através da negação por default.

Exemplo 6.5 Seja $P = \{a? \leftarrow a?\}$. Segundo a WFM_{EI}, $a?$ é falso. Para estar de acordo com a $WFM_{EI}(P)$, toda árvore TU para $a?$ deve ser falha. Como mostrado pela figura 6.1, a única árvore TU para $a?$ é infinita, portanto essa árvore é falha, obtendo, assim, o resultado desejado.

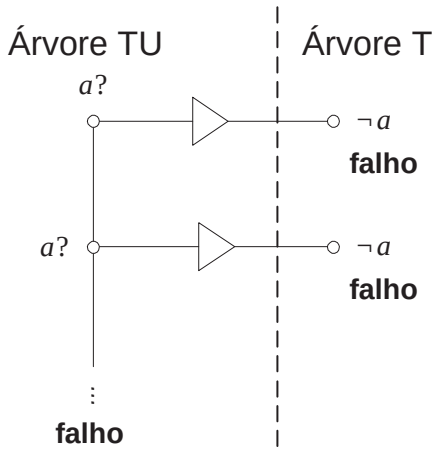

Figura 6.1: Recursão Infinita Positiva

No que tange ao problema da recursão infinita através da negação default, ele pode ser adequadamente tratado da seguinte maneira:

Exemplo 6.6 Seja $P = \{a? \leftarrow not\ a?, not\ \neg a\}$ um programa cujo literal $a?$ está envolvido em recursão infinita através da negação default ($a?$ depende de $not\ a?$). Pela $WFM_{EI}(P)$, $a?$ é indefinido, logo para estar de acordo com a $WFM_{EI}(P)$, toda árvore T para $a?$ deve ser falha e deve existir pelo menos uma árvore TU bem sucedida para $a?$. Na construção de uma árvore T para $\leftarrow a?$, inicialmente, resolve-se essa meta com a única regra de P. Como resultado, obtém-se dois filhos: $not\ a?$ e $not\ \neg a$. Para encontrar o *status* do nó rotulado com $not\ a?$, deve-se construir uma árvore TU para $a?$. Se existir uma árvore TU bem sucedida para $a?$, o literal $not\ a?$ será falho na árvore T; caso contrário, $not\ a?$ será bem sucedido.

Para construir uma árvore TU para $a?$, deve-se inicialmente averiguar o *status* das árvores T para $\neg a$. Como não existe regra para $\neg a$ em P, a árvore T para $\neg a$ é falha. Assim pela definição 6.3, o *status* da árvore TU para $a?$ é determinado resolvendo $\leftarrow a?$ com as regras de P. Desde que a única regra para $a?$ em P é $a? \leftarrow not\ a?, not\ \neg a$, obtém-se como nós filhos $not\ a?$ e $not\ \neg a$. Logo, uma recursão em $not\ a?$ através da

negação por default é encontrada (Veja figura 6.2. Como se está numa derivação que é bem sucedida para literais verdadeiros ou indefinidos, *not* $a?$ é bem sucedido na árvore TU. Desde que não há nenhuma regra para $\neg a$, não existe árvore T bem sucedida para esse literal, portanto *not* $\neg a$ é bem sucedido na árvore TU. Conseqüentemente, $a?$ é bem sucedido na árvore TU. Disso, conclui-se que *not* $a?$ é falho na árvore T, resultando que $a?$ também é falho na árvore T. Assim, como esperado, todas as árvores T para $a?$ falham e há uma árvore TU bem sucedida para $a?$.

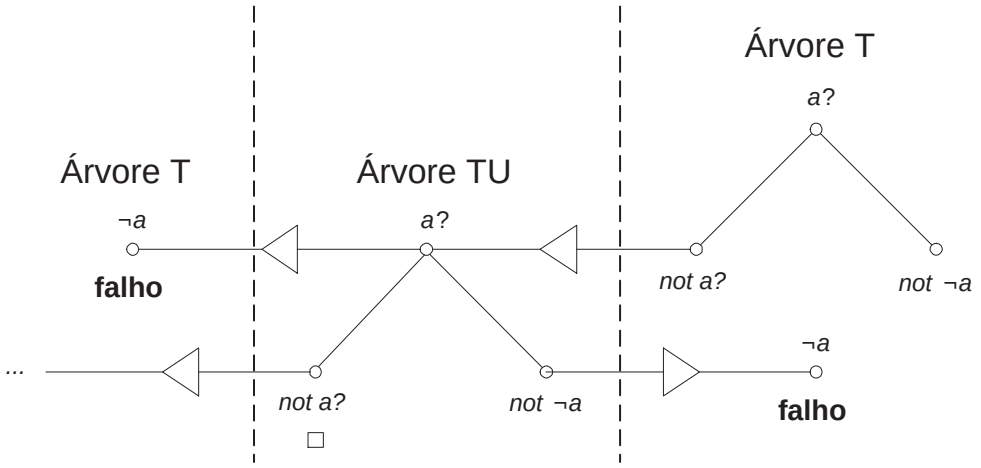

Figura 6.2: Recursão Infinita Negativa

Ainda resta fazer algumas considerações a cerca do tratamento do princípio da coerência em árvores T e em árvores TU. No exemplo abaixo, para efeito de análise, será suprimida inicialmente as condições 1a e 2a da definição 6.2.

Exemplo 6.7 Seja $P = \{b? \leftarrow not\ a?, not\ \neg b;\ \neg a, \neg a?, a?\}$ cujo WFM_{EI} é igual a $\{\neg a, a?, \neg a?, b?\} \cup not\{a, \neg a, a?, b, \neg b, b?, \neg b?\}$.

Para provar que $b?$ é verdadeiro em $WFM_{EI}(P)$, deve-se contruir uma árvore T bem sucedia para $b?$. Da única regra para $b?$, obtém-se os nós filhos *not* $a?$ e *not* $\neg b$. Para *not* $a?$ ser bem sucedido na árvore T, toda árvore TU para $a?$ deve ser falha. Todavia, uma derivação bem sucedida é obtida para $a?$ numa árvore TU, logo *not* $a?$ é falho na árvore T (Figura 6.3(a)). No entanto, como $\neg a$ é bem sucedido em $WFM_{EI}(P)$, por coerência, *not* $a?$ deve ser bem sucedido na árvore T.

Nesse sentido, para garantir a possibilidade de derivar um literal default por coerência, são introduzidas as condições 1a e 2a na definição 6.2. Como já é sabido, um nó rotulado com o literal *not* L (resp. *not* $L?$) é bem sucedido numa árvore T sss toda árvore TU

para L (resp. $L?$) é falha. Todavia, pela condição 1a (resp. condição 2a), se existe uma árvore T bem sucedida para $\neg L?$ (resp. $\neg L$), a árvore TU para L (resp. $L?$) é falha. Conseqüentemente, o nó rotulado com *not* L (resp. *not* $L?$) é bem sucedido na árvore T. Disso, pode-se concluir que a condição 1a e a condição 2a da definição 6.2 garantem a observância ao princípio da coerência nas árvores T, pois se existe uma árvore T bem sucedida para $\neg L$ (resp. $\neg L?$), então o nó rotulado com *not* $L?$ (resp. *not* L) também é bem sucedido nessa árvore. Naturalmente, as condições 1a e 2a também permitem derivar literais default por coerência nas árvores TU.

Em suma, o princípio da coerência impõe a verificação de uma precondição para a aplicação da resolução sobre literais objetivos e plausíveis. Desse modo, retornando ao exemplo 6.7, constata-se que há uma árvore T bem sucedida para $\neg a$, portanto $a?$ é falho na árvore TU e *not* $a?$ é bem sucedido na árvore T. O outro nó folha da árvore T é rotulado com o literal *not* $\neg b$. Como não há nenhuma árvore TU bem sucedida para $\neg b$, conclui-se que *not* $\neg b$ e com isso $b?$ são bem sucedidos na árvore T (Figura 6.3(b)).

Com isso, pode-se estabelecer que as duas principais diferenças entre as árvore T e TU são referentes ao trato da recursão infinita através da negação default e na forma de lidar com o princípio da coerência. Para justificar a primeira diferença, observe que um nó bem sucedido numa árvore T deve corresponder a um literal verdadeiro em WFM_{EI}. Enquanto isso, um nó bem sucedido numa árvore TU deve corresponder a um literal verdadeiro ou indefinido em WFM_{EI}. Em WFM_{EI}, como a recursão infinita através da negação default conduz a um valor indefinido para os literais envolvidos, os nós rotulados com esses literais devem ser falhos na árvore T e bem sucedidos na árvore TU.

No tocante a segunda diferença, deve-se salientar que as árvores T e TU foram concebidas para respectivamente implementarem os operadores Γ e Γ_s utilizados na definição de WFM_{EI}. Por conta do princípio da coerência, esses operadores são distintos. Como conseqüência, as árvores T e TU também devem ser distintas. Por esse motivo, o uso do *status* indefinido em vez das árvores T e TU é um elemento complicador à generalização para programas estendidos, dentre eles os programas estendidos da inconsistência epistêmica. Em contrapartida, para programas normais em lógica, o uso de árvores T e TU pode ser equivalentemente trocado pela introdução do *status* indefinido, pois a única diferença que restaria entre esses dois tipos de árvore estaria no sucesso ou falha de literais envolvidos em recursão infinita através da negação default.

6.2 Definição de SLX_{EI}

Baseado nas idéias apresentadas na seção anterior, aqui é definido um procedimento de refutação para programas em lógica da inconsistência epistêmica sob a $WFSX_{EI}$, cha-

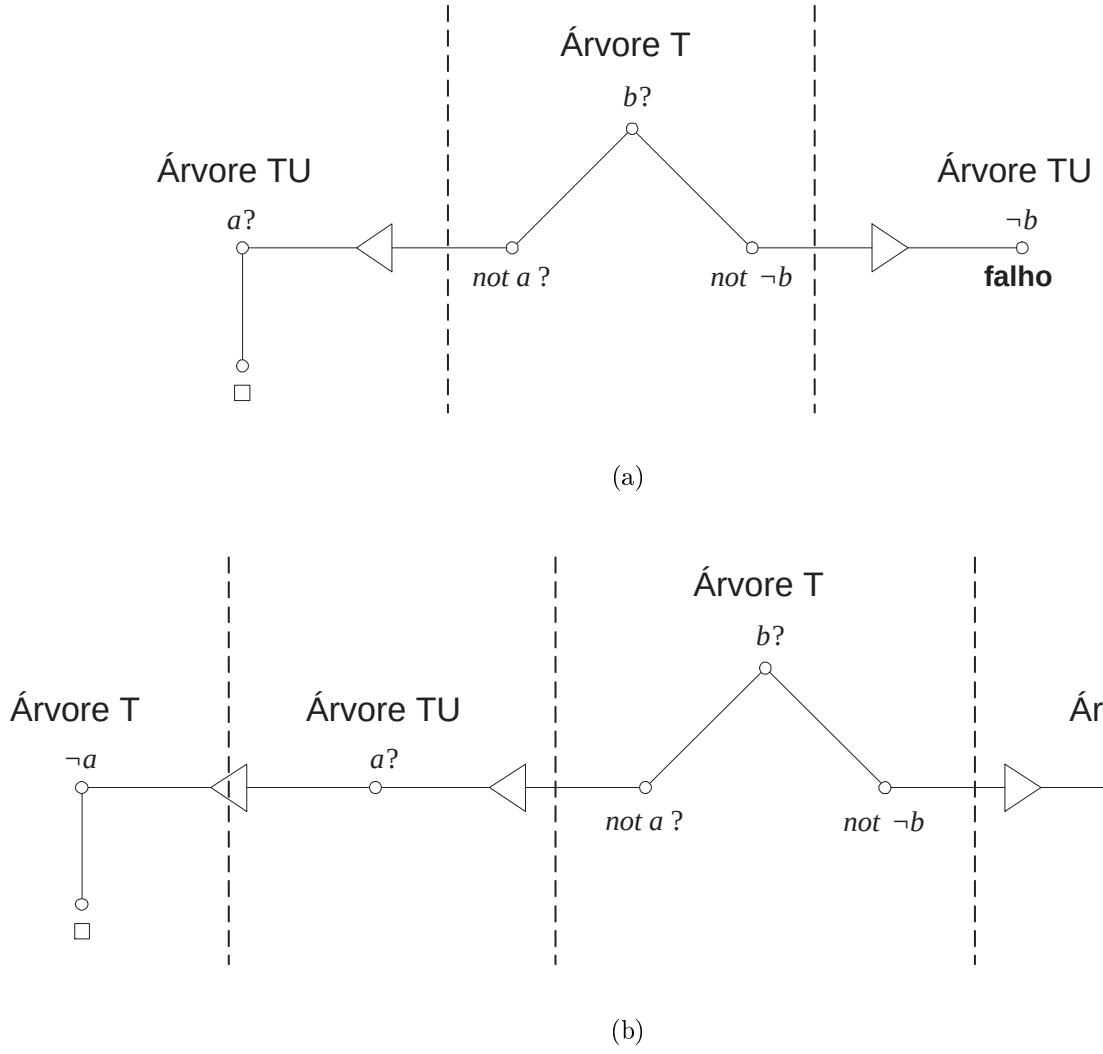

Figura 6.3: Derivações SLX_{EI} do Exemplo 6.7

mado de SLX_{EI} (*Selection rules driven Linear resolution for eXtended logic programs of Epistemic Inconsistency*).

Definição 6.8 (Refutação e falha em SLX_{EI}) *Uma refutação SLX_{EI}-T (resp. refutação SLX_{EI}-TU) para uma meta M em P é uma derivação SLX_{EI}-T finita (resp derivação SLX_{EI}-TU finita) que termina com a meta vazia $\leftarrow \square$.*

Uma derivação SLX_{EI}-T (resp. uma derivação SLX_{EI}-TU) para uma meta M em P é uma SLX_{EI}-T falha (resp. SLX_{EI}-TU falha) sss essa derivação não é uma refutação, ou seja, é infinita ou termina com uma meta diferente da meta vazia.

Definição 6.9 (Derivação SLX_{EI}-T) *Seja P um programa em lógica da inconsistência epistêmica e R uma regra computacional fixa, mas arbitrária. Uma derivação SLX_{EI}-T*

$M_0, M_1, ...$ para uma meta M em P através de R é definida como segue, em que $M_0 = M$, $M_i =\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ e suponha que R seleciona o literal α_j com $1 \leq j \leq m$. Dessa forma, tem-se que

- *Se α_j é um literal objetivo L e existe uma regra em P do tipo $L \leftarrow \delta_1, \ldots, \delta_p$, então a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$;*
- *Se α_j é um literal plausível $L?$ e há uma regra em P do tipo $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$, então*
  - *Se $q = 0$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Além disso, se $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$;*
  - *Se há pelo menos um literal default nessa regra ($q > 0$), a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $q > 0$ e $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$.*
- *Se α_j é um literal plausível $L?$ e há uma regra em P do tipo $L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$, então*
  - *Se $q = 0$ e $p \neq 1$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\,\beta_1, \ldots, not\ \beta_n$. Senão, se $q = 0$ e $p = 1$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\,\beta_1, \ldots, not\ \beta_n$;*
  - *Se $q > 0$, então a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $q > 0$ e $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$, $\alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$.*

Por sua vez, se R seleciona o literal $not\ \beta_k$ com $1 \leq k \leq n$, tem-se que

- *Se não existe nenhuma refutação SLX_{EI}-TU para β_k em P, então a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_{k-1}, not\ \beta_{k+1}, \ldots, not\ \beta_n$;*

Em qualquer outra circunstância, M_i é a última meta na derivação.

Definição 6.10 (Derivação SLX_{EI}-TU) *Seja P um programa em lógica da inconsistência epistêmica e R uma regra computacional fixa, mas arbitrária. Uma derivação SLX_{EI}-TU $M_0, M_1, \ldots$, para M em P através de R é definida como segue, em que $M_0 = M$, $M_i =\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ e suponha que R seleciona o literal α_j com $1 \leq j \leq m$. Dessa forma, tem-se que*

- *Se α_j é um literal objetivo L, então*
  - *Se existe uma refutação SLX_{EI}-T para $\neg L?$, M_i é a última meta na derivação;*
  - *Senão, se existe uma regra em P do tipo $L \leftarrow \delta_1, \ldots, \delta_p$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$;*
  - *Caso contrário, M_i é a última meta na derivação.*
- *Se α_j é um literal plausível L?, então*
  - *Se existe uma refutação SLX_{EI}-T para $\neg L$, então M_i é a última meta na derivação;*
  - *Senão, se há uma regra em P do tipo $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$, então*
    - *Se $q = 0$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Além disso, se $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$;*
    - *Se $q > 0$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $q > 0$ e $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$.*
  - *Senão, se há uma regra em P do tipo $L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$, então*
    - *Se $q = 0$ e $p \neq 1$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $q = 0$ e $p = 1$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$;*
    - *Se $q > 0$, a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $q > 0$ e $p = 1$, a meta derivada também pode ser $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$.*
  - *Caso contrário, M_i é a última meta na derivação.*

Por sua vez, se R seleciona o literal $not\ \beta_k$ com $1 \leq k \leq n$, tem-se que

- *Se existe uma refutação $SLX_{EI} - T$ para $\leftarrow \beta_k$ em P, então M_i é a última meta na derivação;*
- *Senão, se todas as derivações $SLX_{EI} - T$ para $\leftarrow \beta_k$ são falhas em $SLX_{EI} - T$, então a meta derivada é $\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_{k-1}, not\ \beta_{k+1}, \ldots, not\ \beta_n$.*
- *Caso contrário, devido a recursão infinita através da negação default, pode ocorrer que os itens anteriores não sejam suficientes para determinar a meta derivada.*

Nesse caso, por definição, a meta derivada é também $\leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_{k-1}, not\ \beta_{k+1}, \ldots, not\ \beta_n$.

Exemplo 6.11 Considere o programa

$$P = \begin{cases} a? \leftarrow not\ b?, not\ \neg a & \neg b \\ b? \leftarrow not\ \neg b & \neg b? \end{cases}$$

Uma derivação a partir de $\leftarrow_T a?$ é mostrada na figura 6.4. Como regra de computação é utilizada a que seleciona literais da esquerda para a direita. Na figura 6.4, esses literais selecionados estão em negrito.

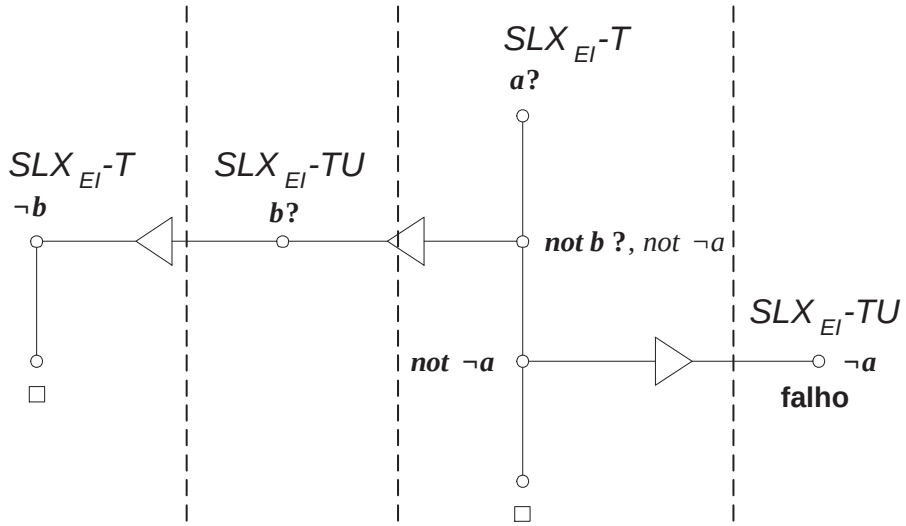

Figura 6.4: Derivação SLX_{EI} do exemplo 6.11

6.3 Corretude e completude de SLX_{EI}

Nesta seção, é provada a corretude e a completude teórica[3] do procedimento de derivação SLX_{EI}. Essa prova baseia-se nas similaridades existentes entre a construção de tal procedimento e a definição de $WFSX_{EI}$ em termos de pontos fixos alternados. Nesse sentido, será demonstrado que SLX_{EI}-T computa a parte Γ e SLX_{EI}-TU, a parte Γ_s.

No lema abaixo, é demonstrado que as derivações SLX_{EI}-T e SLX_{EI}-TU para um literal (objetivo, plausível ou default) qualquer num programa em lógica da inconsistência

[3]Na prática, semânticas bem fundadas de um modo geral não são computáveis [93]. Na teoria, a completude pode ser obtida, admitindo a possibilidade da existência de mais do que ω derivações para construir uma prova.

epistêmica P são as mesmas para esse literal num programa P_{EI}, obtido de P aplicando as transformações T_1 e T_2.

Lema 6.12 *Seja P um programa em lógica da inconsistência epistêmica e P_{EI} o programa obtido de P aplicando as transformações T_1 e T_2 e $M_i = \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ uma meta. Existe uma derivação SLX_{EI}-T (resp. SLX_{EI}-TU) de M_{i+1} a partir de M_i em P através de uma regra computacional R sss existir uma derivação SLX_{EI}-T (resp. SLX_{EI}-TU) de M_{i+1} a partir de M_i em P_{EI}.*

(*Prova*)

$\Rightarrow$

Como usual, na demonstração abaixo, ao se referir a literais, letras gregas minúsculas serão utilizadas para denotar literais objetivos ou plausíveis enquanto que letras romanas irão denotar literais objetivos somente. Além disso, os resultados serão restritos às árvores T. Para as árvores TU, a prova é similar. Assim, em M_i, suponha que R selecione o literal α_j, com $1 \leq j \leq m$. Numa derivação SLX_{EI}-T, pode-se tecer as seguintes considerações:

- Se α_j é um literal objetivo L e existe em P uma regra do tipo $L \leftarrow \delta_1, \ldots, \delta_p$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\,\beta_1, \ldots, not\,\beta_n$. Como essa regra de P também pertence a P_{EI}, M_{i+1} é derivado de M_i em P_{EI};

- Se α_j é um literal plausível $L?$ e existe em P uma regra $L? \leftarrow \delta_1, \ldots, \delta_p$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $p = 1$, M_{i+1} também pode ser igual a $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Por definição, em P_{EI}, existem as regras $L? \leftarrow \delta_1, \ldots, \delta_p$ e também $L? \leftarrow \delta_1?$ se $p = 1$. Desse modo, em ambos os casos, existe uma derivação de M_{i+1} a partir de M_i em P_{EI};

- Se α_j é um literal plausível $L?$ e existe em P uma regra $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$ com $q > 0$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $p = 1$, M_{i+1} também pode ser igual a $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Como P_{EI} contém as regras $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$ e também $L? \leftarrow \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$ se p = 1, novamente, em ambos os casos, existe uma derivação de M_{i+1} em P_{EI} a partir de M_i;

- Se α_j é um literal plausível $L?$ e existe em P uma regra $L \leftarrow \delta_1, \ldots, \delta_p$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ se $p \neq 1$ e $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ se $p = 1$. Por sua vez, P_{EI} possui a regra $L? \leftarrow \delta_1, \ldots, \delta_p$ se $p \neq 1$ e $L? \leftarrow \delta_1?$ se $p = 1$. Desse modo, M_{i+1} é derivado em P_{EI} a partir de M_i;

- Se α_j é um literal plausível $L?$ e existe em P uma regra $L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$ com $q > 0$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Se $p = 1$, M_{i+1} também pode ser igual a $\leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Como P_{EI} contém as regras $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$ e também $L? \leftarrow \delta_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$ se $p = 1$, em ambos os casos, uma derivação de M_{i+1} a partir de M_i é encontrada em P_{EI};

- Em qualquer outra circunstância, M_i é a última meta da derivação em P. Se α_j é um literal objetivo L, então não existe regra do tipo $L \leftarrow L_1, \ldots, L_p$ para L em P. Por definição, não existe regra para L em P_{EI}, logo M_i é também a última meta da derivação em P_{EI}. Por sua vez, se α_j é um literal plausível $L?$, todas as possibilidades de gerar regras para $L?$ em P_{EI} foram esgotadas. Isso implica que não há regras para $L?$ em P_{EI}, logo, mais uma vez, M_i é a última meta da derivação em P_{EI}.

Agora, suponha que R selecione o literal $not\ \beta_k$, $1 \leq k \leq n$:

- Se não existe uma refutação SLX_{EI}-TU para β_k em P, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_{k-1}, not\ \beta_{k+1}, \ldots, not\ \beta_n$. Por sua vez, dos resultados a respeito da derivação SLX_{EI}-TU, sabe-se que existe uma refutação SLX_{EI}-TU para β_k em P_{EI}. Portanto, em P_{EI}, M_{i+1} também é derivado;

- Caso contrário, se existe refutação SLX_{EI}-TU para β_k em P, M_i é a última meta da derivação. Dos resultados pertinentes à derivação SLX_{EI}-TU, M_i também é a última meta da derivação em P_{EI}.

($\Leftarrow$)

Eventualmente, uma regra de P_{EI} pode não pertencer a P. Assim, seja R uma regra de seleção. Para as regras comuns aos dois programas, obviamente, tanto em SLX_{EI}-T quanto em SLX_{EI}-TU, os literais imediatamente derivados através dessas regras a partir de M_i utilizando R serão os mesmos. Desse modo, na demonstração abaixo, é suficiente considerar as derivações que podem ter sido obtidas de regras não comuns a P e P_{EI}.

Seja $M = \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$ uma meta e suponha que numa derivação SLX_{EI}-T, R selecione o literal α_j, com $1 \leq j \leq m$. Com isso, tem-se os seguintes casos:

- Se α_j é um literal plausível $L?$ e existe em P_{EI} uma regra $L? \leftarrow \delta_1, \ldots, \delta_p$ com $p \geq 0$ e $p \neq 1$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Por definição, P contém a regra $L \leftarrow \delta_1, \ldots, \delta_p$ ou $L? \leftarrow \delta_1, \ldots, \delta_p$. Em ambos os casos, M_{i+1} é derivado de M_i em P;

- Se α_j é um literal plausível $L?$ e existe em P_{EI} uma regra $L? \leftarrow L_1?$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, L_1?, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. No tocante a P, sabe-se que P contém a regra $L \leftarrow L_1$ ou $L? \leftarrow L_1$. Em ambos os casos, M_{i+1} é derivado de M_i em P;

- Se α_j é um literal plausível $L?$ e existe em P_{EI} uma regra $L? \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$, com $p, q \geq 0$ e $p \neq 1$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not \beta_n$. Por sua vez, P possui as regras $L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$ ou $L \leftarrow \delta_1, \ldots, \delta_p, not\ \epsilon_1, \ldots, not\ \epsilon_q$, $not\ \neg L$ ou ainda variações dessas duas regras substituindo o literal L por $L?$. Em qualquer um desses casos, M_{i+1} é derivado de M_i em P;

- Se α_j é um literal plausível $L?$ e existe em P_{EI} uma regra $L? \leftarrow L_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$, com $q \geq 0$, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_{j-1}, L_1?, not\ \epsilon_1, \ldots, not\ \epsilon_q$, $not\ \neg L, \alpha_{j+1}, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n$. Entretanto, P possui a regra $L \leftarrow L_1$, $not\ \epsilon_1, \ldots, not\ \epsilon_q$ ou $L \leftarrow L_1, not\ \epsilon_1, \ldots, not\ \epsilon_q, not\ \neg L$ ou ainda variações dessas duas regras substituindo os literais L ou L_1 respectivamente por $L?$ ou $L_1?$. Em qualquer um desses casos, M_{i+1} é derivado em P a partir de M_i;

- Caso contrário, se α_j é um literal objetivo L e não existe regra para L em P_{EI}, então M_i é a última meta da derivação. Por definição, em P, não existe regra para L ou existe uma regra $L \leftarrow \delta_1, \ldots, \delta_p, \epsilon_1, \ldots, \epsilon_q$ com $q > 0$. Em ambos os casos, M_i é a última meta da derivação em P. Senão, se α_j é um literal plausível $L?$ e não existe regra para $L?$ em P_{EI}, também não existe regra para $L?$ em P. Conseqüentemente, M_i é a última meta da derivação tanto em P quanto em P_{EI}.

Agora, suponha que R selecione o literal $not\ \beta_k$, $1 \leq k \leq n$:

- Se não existe uma refutação SLX_{EI}-TU para β_k em P_{EI}, então $M_{i+1} = \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_{k-1}, not\ \beta_{k+1}, \ldots, not\ \beta_n$. Por sua vez, dos resultados referentes às derivações SLX_{EI}-TU, sabe-se que também existe uma refutação SLX_{EI}-TU para β_k em P. Conseqüentemente, M_{i+1} também é derivado em P;

- Caso contrário, se existe uma refutação SLX_{EI}-TU para β_k em P_{EI}, então M_i é a última meta da derivação. Dos resultados pertinentes à derivação SLX_{EI}-TU, M_i também é a última meta da derivação em P.

Com isso, tanto em P quanto em P_{EI}, a meta derivada M_{i+1} vai ser a mesma. $\Diamond$

Sendo assim, no restante deste capítulo, pode-se assumir sem perda de generalidade que os programas em lógica da inconsistência epistêmica já estão nesse formato sintático estabelecido por T_1 e T_2.

Além disso, para secundar esta prova de corretude e completude, serão atribuídos níveis tanto à derivação SLX_{EI}-T quanto à SLX_{EI}-TU de modo que o nível de uma derivação estabeleça qual passo da definição de WFM_{EI} está sendo computado. Como ficará evidente no transcorrer desta prova, não há necessidade de atribuir níveis a SLX_{EI}-T falhos nem a refutações SLX_{EI}-TU, pois tais derivações não contribuem para provar quais literais pertencem a WFM_{EI}.

O *nível* de uma refutação SLX_{EI}-T reflete a profundidade das chamadas às derivações subsidiárias que são necessárias na refutação. O seu valor pode ser determinado a partir dos literais selecionados de cada meta. Numa refutação SLX_{EI}-T, se esse literal é objetivo ou plausível, de imediato, não há necessidade alguma de derivação subsidiária. Conseqüentemente, o nível desse literal é igual ao de seu filho (imediato). Para uma refutação SLX_{EI}-T de um literal default, todas as derivações subsidiárias devem falhar, portanto deve-se considerar o máximo (mais precisamente, o menor limite superior para o caso infinito) das profundidades de todas as derivações TU utilizadas na prova desse literal. Desse modo, a profundidade necessária para remover todos os literais de uma meta é o máximo dos níveis associados com cada literal na meta. Finalmente, para provar $\leftarrow \Box$, nenhuma derivação subsidiária é utilizada, logo o seu nível associado é 0.

Definição 6.13 (Nível de uma refutação SLX_{EI}-T) *O nível de qualquer refutação SLX_{EI}-T é determinado pelo nível de sua raiz. Os níveis dos nós são definidos como segue, em que M_i é uma meta e M_{i+1} é o nó filho de M_i:*

- *Seja M_i uma meta numa refutação cujo próximo literal selecionado é um literal objetivo ou plausível. O nível de M_i é igual ao nível de M_{i+1};*
- *Seja M_i uma meta numa refutação cujo próximo literal selecionado é um literal default not β e seja δ o limite superior ordinal (máximo no caso finito) dos níveis de todo SLX_{EI}-TU falho para $\leftarrow \beta$. O nível de M_i é igual ao valor máximo entre δ e o nível de M_{i+1};*
- *O nível de $\leftarrow \Box$ é 0.*

Similarmente, o nível de um SLX_{EI}-TU falho reflete a profundidade das chamadas necessárias para falhar as derivações subsidiárias. Claramente, a falha de uma derivação é unicamente determinada pela sua última meta M_n. Para ser mais preciso, pelo literal selecionado de M_n. Caso seja um literal objetivo L (resp. literal plausível $L?$), duas razões podem ter ocasionado a falha na derivação SLX_{EI}-TU: não há regras para L (resp. $L?$) no programa em questão ou existe uma refutação SLX_{EI}-T para $\neg L?$ (resp. $\neg L$). No primeiro caso, não há necessidade alguma de derivação subsidiária para falhar L (resp. $L?$), logo o nível de M_n é 0. No outro caso, deve-se encontrar uma refutação

SLX_{EI}-T para $\neg L?$ (resp. $\neg L$). Várias podem existir, mas é suficiente considerar a que tem profundidade mínima. Além disso, como houve uma chamada extra para refutar esse literal, é acrescentado uma unidade ao valor obtido. Por outro lado, caso seja selecionado em M_n um literal default *not* β, uma SLX_{EI}-TU será encontrada se existir uma refutação SLX_{EI}-T para β. Assim, para determinar o nível de M_n, mais uma vez, dessas refutações SLX_{EI}-T, é suficiente considerar a que tem profundidade mínima e em seguida acrescentar a esse valor a unidade. O incremento de uma unidade no nível nesse tipo de derivação está relacionado ao incremento de índices dos $I_i s$ na seqüência para construção de WFM_{EI} somente após a aplicação dos dois operadores Γ e Γ_S.

Definição 6.14 (Nível de SLX_{EI}-TU falho) *SLX_{EI}-TU falho infinito tem nível 0. No que tange a SLX_{EI}-TU falho finito, o seu nível é o de sua última meta M_n. Seja L_k, o literal selecionado de M_n:*

- *Seja L_k um literal objetivo L (resp. literal plausível $L?$). Caso não haja nenhuma regra para L (resp. $L?$), o nível de M_n é 0; senão, o nível de M_n é $\alpha + 1$, em que α é o mínimo dos níveis de todas as refutações SLX_{EI}-T para $\leftarrow \neg L?$ (resp. $\leftarrow \neg L$).*
- *Se L_k é um literal default not β, então o nível de M_n é $\alpha + 1$, em que α é o mínimo dos níveis de todas as refutações SLX_{EI}-T para $\leftarrow \beta$.*

No lema abaixo, a existência de seqüências em que alguns literais default são removidos para aplicar os operadores Γ e Γ_s é relacionada à remoção de literais defaults nas derivações SLX_{EI}-T e SLX_{EI}-TU.

Lema 6.15 (Implementação de Γ e Γ_s por $SLX_{EI} - T$ e $SLX_{EI} - TU$) *Sejam I uma interpretação e P, um programa em lógica da inconsistência epistêmica. Considere uma derivação SLX_{EI}-T (resp. derivação SLX_{EI}-TU) para α em P, construída de acordo com definição 6.9 (resp. 6.10), exceto que os literais default selecionados not β são bem sucedidos e imediatamente removidos das árvores se $\beta \notin I$. Caso contrário, são falhos. Assim sendo, $\alpha \in \Gamma I$ (resp. $\alpha \in \Gamma_s I$) sss a derivação SLX_{EI}-T (resp. SLX_{EI}-TU) para α é finita e termina com a meta vazia.*

(*Prova*)

A demonstração desse resultado será feita somente para SLX_{EI}-TU, que é mais complexa. Para SLX_{EI}-T, a prova é similar. Assim, por indução sob a altura h da árvore de derivação SLX_{EI}-TU, será mostrado que existe uma refutação SLX_{EI}-TU de altura h para α sss $\alpha \in T^{\uparrow h}_{\frac{P_s}{I}}$. Vale ressaltar que uma derivação constituída somente pelo nó raiz possui altura 0.

Base indutiva

h = 1 Uma refutação SLX_{EI}-TU em P possui altura 1 sss pelo menos um dos seguintes itens for verificado:

1. A raiz de SLX_{EI}-TU é um literal objetivo L (resp. literal plausível $L?$) para o qual há em P uma regra da forma

$$L \text{ (resp. } L?)$$

em que todo SLX_{EI}-T para $\neg L?$ (resp. $\neg L$) é falho. Disso, obtém-se que SLX_{EI}-TU para $not\ \neg L?$ (resp. $not\ \neg L$) é bem sucedido, ou seja, $\neg L? \notin I$ (resp. $\neg L \notin I$);

2. A raiz de SLX_{EI}-TU é um literal plausível $L?$ para o qual há em P uma regra da forma

$$L? \leftarrow not\ \alpha_1, \ldots, not\ \alpha_n, not\ \neg L\ (n \geq 0),$$

em que todo SLX_{EI}-T para $\neg L$ é falho e todo literal default $not\ \alpha_1, \ldots, not\ \alpha_n$, é bem sucedido em SLX_{EI}-TU, ou seja, $\alpha_1 \notin I, \ldots, \alpha_n \notin I, \neg L \notin I$.

Por outro lado, qualquer literal objetivo L pertence a $T^{\uparrow 1}_{\frac{P_s}{I}}$ sss existe em P_s uma regra da forma $L \leftarrow not\ \neg L?$ associada à regra L em P tal que $\neg L? \notin I$. Além disso, qualquer literal plausível $L?$ pertence a $T^{\uparrow 1}_{\frac{P_s}{I}}$ sss existe em P_s uma regra da forma $L? \leftarrow not\ \neg L$ associada à regra $L?$ em P tal que $\neg L \notin I$ ou existe em P_s uma regra $L? \leftarrow not\ \alpha_1, \ldots, not\ \alpha_n, not\ \neg L$ associada à regra $L? \leftarrow not\ \alpha_1, \ldots, not\ \alpha_n, not\ \neg L$ em P tal que $\alpha_1 \notin I, \ldots, \alpha_n \notin I, \neg L \notin I$. Com isso, está demonstrado o resultado desse lema para a base indutiva.

Passo indutivo

Assume-se que existe uma refutação SLX_{EI}-TU de altura h para um literal α em P sss $\alpha \in T^{\uparrow h}_{\frac{P_s}{I}}$. Na seqüência, os resultados são limitados às refutações SLX_{EI}-TU para literais plausíveis. A demonstração envolvendo literais objetivos é similar. Desse modo, existe uma refutação SLX_{EI}-TU de altura $h+1$ para um literal plausível $L?$ em P sss há uma regra em P da forma

$$L? \leftarrow \alpha_1, \ldots, \alpha_m \qquad (m \geq 0)$$

ou da forma

$$L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n, not\ \neg L \qquad (m, n \geq 0),$$

tal que não existe refutação SLX_{EI}-T para $\neg L$ e todos os literais defaults $not\ \beta_1, \ldots,$ $not\ \beta_n, not\ \neg L$ são bem sucedidos na derivação SLX_{EI}-TU. Portanto, $\beta_1 \notin I, \ldots, \beta_n \notin I, \neg L \notin I$. Além disso, as derivações SLX_{EI}-TU para $\alpha_1, \ldots, \alpha_m$ são bem sucedidas de altura $\leq h$.

Por hipótese de indução, as derivações SLX_{EI}-TU de altura h para $\alpha_1, \ldots, \alpha_m$ são bem sucedidas sss

$$\alpha_1 \in T^{\uparrow h}_{\frac{P_s}{I}}, \ldots, \alpha_m \in T^{\uparrow h}_{\frac{P_s}{I}}.$$

Com isso, existe uma refutação SLX_{EI}-TU para $L?$ de altura $h+1$ sss existe em P_s, uma regra da forma

$$L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ \neg L$$

ou da forma

$$L? \leftarrow \alpha_1, \ldots, \alpha_m, not\ \beta_1, \ldots, not\ \beta_n, not\ \neg L \qquad (m, n \geq 0)$$

tal que $\beta_1 \notin I, \ldots, \beta_n \notin I, \neg L \notin I$ e

$$\alpha_1 \in T^{\uparrow h}_{\frac{P_s}{I}}, \ldots, \alpha_m \in T^{\uparrow h}_{\frac{P_s}{I}}$$

sss $L? \in T^{\uparrow h+1}_{\frac{P_s}{I}}$. ◊

Para demonstrar a corretude de SLX_{EI}, a computação executada em SLX_{EI}-T e SLX_{EI}-TU é relacionada com os operadores Γ e Γ_s incluindo também os literais defaults.

Lema 6.16 *Sejam P um programa em lógica da inconsistência epistêmica, α, um literal objetivo ou plausível e I_j, um elemento da seqüência de interpretações construído na definição de WFM_{EI} de P pelo teorema 5.62. Nesse caso, tem-se que:*

- *Se há uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P com nível $\leq i$, então $\alpha \in I_{i+1}$;*
- *Se todas as derivações SLX_{EI}-TU para $\leftarrow \alpha$ em P são falhas com nível $\leq i$, então $\alpha \notin \Gamma_s I_i$.*

(*Prova*)

A prova desses resultados será feita por indução transfinita sobre i:

Limites ordinais: Por simplicidade, na demonstração envolvendo os limites ordinais, será separada o caso do limite ordinal ser igual a 0 dos demais.

Seja $\delta = 0$ um limite ordinal. Se há uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P de nível 0, então todos os literais default $not\ \beta_1, \ldots, not\ \beta_n$, com $n \geq 0$, presentes nessa derivação são bem sucedidos de nível 0. Com isso, todo SLX_{EI}-TU para β_i, $1 \leq i \leq n$, é falho com

nível 0. Do resultado abaixo envolvendo SLX_{EI}-TU falho com nível = 0, conclui-se que $\beta_i \notin \Gamma_s I_0$. $1 \leq i \leq n$. Disso, resulta que há uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em $\frac{P}{\Gamma_s I_0}$. Pelo lema 6.15, $\alpha \in \Gamma\Gamma_s I_0$, ou seja, $\alpha \in I_1$.

Para outros limites ordinais δ diferentes de 0, se há uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P de nível $\leq \delta$, então todos os seus literais default $not\ \beta_1, \ldots, not\ \beta_n$, com $n \geq 0$, presentes nessa derivação são bem sucedidos com todas as derivações SLX_{EI}-TU subsidiárias para $\leftarrow \beta_i$, $1 \leq i \leq n$, sendo falhas com nível $\leq \delta$. Por hipótese, $\beta_i \notin \Gamma_s I_\delta$, $1 \leq i \leq n$. Portanto, existe uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em $\frac{P_s}{\Gamma_s I_\delta}$. Pelo lema 6.15, $\alpha \in \Gamma\Gamma_s I_\delta$, ou seja, $\alpha \in I_{\delta+1}$.

Deve-se, agora, considerar as derivações SLX_{EI}-TU falhas com nível 0. Assumir que toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ é falha com nível 0 equivale a dizer que para todo literal L_j (resp. $L_j?$) presente nessas derivações, não existe refutação SLX_{EI}-T para $\neg L_j?$ (resp. $\neg L_j$) e todos os literais default presentes em qualquer dessas derivações SLX_{EI}-TU são bem sucedidos. Além disso, toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ é infinita ou termina com uma meta cujo próximo literal selecionado é um literal objetivo ou plausível. Portanto, toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ é falha em $\frac{P_s}{I_0}$. Pelo lema 6.15, $\alpha \notin \Gamma_s I_0$.

Para outros limites ordinais δ além de 0, assume-se que todo SLX_{EI}-TU para $\leftarrow \alpha$ é falho com nível $\leq \delta$. Assim, em cada SLX_{EI}-TU para $\leftarrow \alpha$, existe pelo menos um literal L_j (resp. $L_j?$) com uma refutação SLX_{EI}-T bem sucedida para $\neg L_j?$ (resp. $\neg L_j$) de nível $\lambda < \delta$ ou existe pelo menos um literal default $not\ \beta$ com SLX_{EI}-T bem sucedido para β de nível $\lambda < \delta$. Por hipótese, no primeiro caso, obtém-se que $\neg L_j? \in I_{\lambda+1}$ (resp. $\neg L_j \in I_{\lambda+1}$) enquanto que pelo segundo caso, $\beta \in I_{\lambda+1}$. Com isso, toda SLX_{EI}-TU para $\leftarrow \alpha$ em $\frac{P_s}{\bigcup_{\lambda<\delta} I_{\lambda+1}}$ é falha. Pelo lema 6.15, $\alpha \notin \Gamma_s \bigcup_{\lambda<\delta} I_{\lambda+1}$. Disso, resulta que $\alpha \notin \Gamma_s I_\delta$.

Passo indutivo: A prova do passo indutivo é similar a exibida acima para limites ordinais diferentes de 0. Na seqüência, serão mostrados os resultados para derivações SLX_{EI}-TU. No tocante às derivações SLX_{EI}-T, a prova é similar.

Para os sucessores ordinais, assume-se que todo SLX_{EI}-TU para $\leftarrow \alpha$ é falho com nível $\leq \delta + 1$. Assim, em cada SLX_{EI}-TU para $\leftarrow \alpha$, existe pelo menos um literal L_j (resp. $L_j?$) com uma refutação SLX_{EI}-T bem sucedida para $\neg L_j?$ (resp. $\neg L_j$) de nível $\leq \delta$ ou existe pelo menos um literal default $not\ \beta$ com SLX_{EI}-T bem sucedido para β de nível $\leq \delta$. Por hipótese, no primeiro caso, obtém-se que $\neg L_j? \in I_{\delta+1}$ (resp. $\neg L_j \in I_{\delta+1}$) enquanto que pelo segundo caso, $\beta \in I_{\delta+1}$. Com isso, todo SLX_{EI}-TU para $\leftarrow \alpha$ em $\frac{P_s}{I_{\delta+1}}$ é falha. Pelo lema 6.15, $\alpha \notin \Gamma_s I_{\delta+1}$. $\Diamond$

No teorema abaixo, é provada a corretude de SLX_{EI}.

Teorema 6.17 (Corretude de SLX_{EI}) *Sejam P um programa em lógica da inconsistência epistêmica e α um literal objetivo ou plausível. Se há refutação SLX_{EI}-T para*

$\leftarrow \alpha$ em P, então $\alpha \in WFM_{EI}(P)$. Similarmente, se há uma refutação SLX_{EI}-T para $\leftarrow not\ \alpha$ em P, então $not\ \alpha \in WFM_{EI}(P)$.

(*Prova*)

Se há uma refutação SLX_{EI}-T para α em P, os resultados seguem imediatamente do item 1 do lema 6.16 e da monotonicidade de $\Gamma\Gamma_s$.

Se há uma refutação SLX_{EI}-T para *not* α com nível i, então todas as derivações SLX_{EI}-TU para $\leftarrow \alpha$ são falhas e de nível $\leq i$. Pelo item 2 do lema 6.16, tem-se que $\alpha \notin \Gamma_s I_i$.

Seja I_δ o menor ponto fixo de $\Gamma\Gamma_s$. Como $\Gamma\Gamma_s$ é monotônico, $I_i \subseteq I_\delta$, isto é, para qualquer literal objetivo ou plausível α, $\alpha \in I_i \Rightarrow \alpha \in I_\delta$. Por antimonotonicidade de Γ_s, $\alpha \in \Gamma_s I_\delta \Rightarrow \alpha \in \Gamma_s I_i$. Da contrapositiva dessa regra, obtém-se que $\alpha \notin \Gamma_s I_i \Rightarrow \alpha \notin \Gamma_s I_\delta$. Como $\alpha \notin \Gamma_s I_i$, conclui-se que $\alpha \notin \Gamma_s I_\delta$. Pelo teorema 5.62, resulta que $not\ \alpha \in WFM_{EI}(P)$. ◊

A prova da completude é similar a da corretude para SLX_{EI}. Nesse sentido, é apresentado o lema abaixo, que é a "volta" do lema 6.16.

Lema 6.18 *Sejam P um programa em lógica da inconsistência epistêmica, α, um literal objetivo ou plausível e I_i, um elemento da seqüência de intepretações construídas para definir WFM_{EI} de P. Dessa forma, existe uma regra de seleção R tal que*

1. *Se $\alpha \in I_{i+1}$, então existe uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P com nível $\leq i$.*
2. *Se $\alpha \notin \Gamma_s I_i$, então todas as derivações SLX_{EI}-TU para $\leftarrow \alpha$ em P são falhas com nível $\leq i$.*

(*Prova*)

Seja R uma regra de seleção que começa selecionando todos os literais objetivos ou plausíveis. Os literais defaults são selecionados posteriormente da seguinte maneira: *not* α é selecionado antes de *not* α' se há um j na seqüência de I_i tal que $\alpha \in I_j$ e $\alpha' \notin I_j$.

Limite ordinal: Mais uma vez, por simplicidade, será separado o caso do limite ordinal ser igual a 0 dos demais.

Seja $i = 0$, um limite ordinal. Por definição, se $\alpha \in I_1$, $\alpha \in \Gamma\Gamma_s\{\}$. Pelo lema 6.15, existe refutação SLX_{EI}-T para $\leftarrow \alpha$ em $\frac{P}{\Gamma_s\{\}}$. Isso quer dizer que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, $\beta \notin \Gamma_s\{\}$. Pelo item 2 deste lema, sabe-se que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, todo SLX_{EI}-TU

para β é falho com nível 0. Conseqüentemente, existe uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P com nível 0.

Seja $i = \delta$. Para outros limites ordinais δ diferentes de 0, se $\alpha \in I_{\delta+1}$, por definição $\alpha \in \Gamma\Gamma_s I_\delta$. Pelo lema 6.15, existe refutação SLX_{EI}-T para $\leftarrow \alpha$ em $\frac{P}{\Gamma_s I_\delta}$. Disso resulta que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, $\beta \notin \Gamma_s I_\delta$. Pelo item 2 deste lema, conclui-se que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, todo SLX_{EI}-TU para β é falho com nível $\leq \delta$. Desse modo, existe uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P com nível $\leq \delta$.

Agora, será considerado o caso das derivações SLX_{EI}-TU de nível 0. Pelo lema 6.15, se $\alpha \notin \Gamma_s I_0$, então toda derivação SLX_{EI}-TU para α em $\frac{P_s}{I_0}$ é infinita ou termina com um meta cujo literal selecionado é um literal objetivo ou plausível β. Além disso, não existe regra para β em P_s. Portanto, toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ em P é infinita ou termina com um meta cujo literal selecionado é um literal objetivo ou plausível sem regra para ele em P. Portanto, toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ em P é falha com nível 0.

Seja $i = \delta$. Para limites ordinais δ diferentes de 0, se $\alpha \notin \Gamma_s I_\delta$, então pelo lema 6.15, toda derivação SLX_{EI}-TU para α em $\frac{P_s}{I_\delta}$ é infinita ou termina com um meta cujo literal selecionado é um literal objetivo ou plausível. Isso quer dizer que toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ em P é infinita ou termina com uma meta cujo literal selecionado é um literal objetivo ou plausível ou ainda termina com uma meta cujo literal default selecionado é *not* β com $\beta \in I_\delta$.

Se a derivação SLX_{EI}-TU para α em P é infinita, ela será falha com nível 0. Por seu turno, se essa derivação termina com um literal objetivo L (resp. literal plausível $L?$), há duas possibilidades: se não existir nenhuma regra em P para L (resp. $L?$), tal derivação é falha com nível 0; caso contrário, $\neg L? \in I_\delta$ (resp. $\neg L \in I_\delta$). Então, do item 1 deste lema, obtém-se que há uma refutação SLX_{EI}-T para $\neg L?$ (resp. $\neg L$) em P com nível $\leq \delta - 1$. Com isso, a derivação SLX_{EI}-TU para α é falha com nível $\leq \delta$. Finalmente, se a derivação terminar com *not* β tal que $\beta \in I_\delta$, então, mais uma vez, do item 1, obtém-se que há uma refutação SLX_{EI}-T para $\leftarrow \beta$ com nível $\leq \delta - 1$. Desse modo, por definição, o nível dessa derivação é $\leq \delta$.

Observe que essa última condição envolvendo os literais defaults somente é possível por causa da regra de seleção R. Caso contrário, a derivação poderia terminar com uma meta cujo literal default selecionado fosse *not* β com $\beta \in I_\lambda$ e $\beta \notin I_\delta$ em que $\lambda > \delta$.

Passo indutivo: Assume-se que os itens 1 e 2 do lema são verificados para δ. Deve-se provar que os itens 1 e 2 também valem para $\delta + 1$. Essa prova é similar a anterior quando os limites ordinais são diferentes de 0. Por brevidade, a demonstração será restrita ao item 1.

Suponha que $\alpha \in I_{\delta+2}$, por definição $\alpha \in \Gamma\Gamma_s I_{\delta+1}$. Pelo lema 6.15, existe refutação SLX_{EI}-T para $\leftarrow \alpha$ em $\frac{P}{\Gamma_s I_{\delta+1}}$. Disso resulta que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, $\beta \notin \Gamma_s I_{\delta+1}$. Por hipótese de indução e pelo item 2, sabe-se que existe uma derivação SLX_{EI}-T para $\leftarrow \alpha$ em P tal que para todo literal default *not* β presente nessa derivação, SLX_{EI}-TU para β é falho com nível $\leq \delta + 1$. Conseqüentemente, existe uma refutação SLX_{EI}-T para α em P com nível $\leq \delta + 1$. $\Diamond$

Observe que na prova do item 1 nunca é utilizada a regra de seleção especial. Desse modo, para derivações SLX_{EI}-T, regras de seleção arbitrárias podem ser utilizadas. Além disso, no item 2, o único uso da regra R é o de garantir que o nível de todas derivações SLX_{EI}-TU falhas possuem realmente o nível $\leq i$. Isso é necessário para provar o lema por indução. Entretanto, é claro que se ao utilizar R, toda derivação SLX_{EI}-TU para $\leftarrow \alpha$ é falha, o mesmo ocorre com uma regra de seleção arbitrária, embora com um nível possivelmente maior. Esse é o motivo pelo qual não há necessidade de considerar essa regra de seleção especial no teorema abaixo:

Teorema 6.19 (Completude teórica de SLX_{EI}) *Seja P um programa em lógica da inconsistência epistêmica e α um literal objetivo ou plausível. Se $\alpha \in WFM_{EI}(P)$, então existe uma refutação SLX_{EI}-T para $\leftarrow \alpha$ em P. Do mesmo modo, se not $\alpha \in WFM_{EI}(P)$, então existe uma refutação SLX_{EI}-T para $\leftarrow$ not α em P.*

(*Prova*)

Se $\alpha \in WFM_{EI}(P)$, a prova segue do item 1 do lema 6.18 e da monotonicidade de $\Gamma\Gamma_s$.

Suponha que *not* $\alpha \in WFM_{EI}(P)$. Pela definição de WFM_{EI}, existe δ tal que I_δ é o menor ponto fixo de $\Gamma\Gamma_s$ com $\alpha \notin \Gamma_s I_\delta$. Pelo item 2 do lema 6.18, obtém-se que qualquer SLX_{EI}-TU para $\leftarrow \alpha$ em P é falha. Desse modo, existe uma refutação SLX_{EI}-T em P para $\leftarrow$ *not* α. $\Diamond$

Esses métodos também permitem detectar programas contraditórios.

Corolário 6.20 *Um programa em lógica da inconsistência epistêmica P é contraditório sss existe $L \in \mathcal{B}_P$ tal que há refutações SLX_{EI}-T para $\leftarrow \{L, \neg L\}$ ou $\leftarrow \{L?, \neg L\}$ ou ainda para $\leftarrow \{L, \neg L?\}$.*

(*Prova*)

A prova desse corolário segue diretamente da definição de WFM_{EI} e da corretude e completude de SLX_{EI}:

Um programa em lógica da inconsistência epistêmica P é contraditório sss existe $L \in \mathcal{B}_P$ tal que $\{L, \neg L\}$, $\{L?, \neg L\}$ ou $\{L, \neg L?\}$ pertencem a $WFM_{EI}(P)$ sss existe $L \in \mathcal{B}_P$ tal que há respectivamente refutações SLX_{EI}-T para $\leftarrow \{L, \neg L\}$ ou $\leftarrow \{L?, \neg L\}$ ou $\leftarrow \{L, \neg L?\}$. $\Diamond$

6.4 Implementação de SLX_{EI}

Apesar de ser correto e completo, o procedimento de derivação SLX_{EI} da forma como foi definido na seção 6.2 não é efetivo mesmo para programas básicos sem símbolos funcionais. Isso ocorre porque nenhum mecanismo foi implementado para detectar ciclos. Assim, a terminação desse método não é garantida.

Em [20], é apresentado um método que resolve o problema da terminação para programas em lógica estendidos básicos sem símbolos funcionais. Com esse objetivo, são introduzidas regras que podam o espaço de busca, eliminando tanto a recursão infinita através de ciclos positivos quanto a recursão infinita através da negação por default. Para detectar as recursões do primeiro tipo, são atribuídos a cada literal L de toda árvore T ou TU o conjunto de ancestrais locais de L (sem incluí-lo) naquela árvore. Por sua vez, para detectar a recursão infinita através da negação default, são atribuídos a cada literal L das árvores subsidiárias (T e TU) o conjunto de seus ancestrais globais, que levam em consideração a derivação inteira. Para a árvore principal (não subsidiária) não é necessário atribuir ancestrais globais, pois além de serem iguais aos locais, a recursão através da negação default se existir, somente poderá ser detectada nas árvores subsidiárias.

Essas regras de podas apresentadas em [20] para SLX poderão ser diretamente adaptadas para SLX_{EI}, garantido, assim, a terminação para programas em lógica da inconsistência epistêmica com base de Herbrand finita. Tais regras não serão apresentadas neste trabalho.

Embora utilizem dois tipos de derivações, em métodos como o SLX_{EI} o tamanho da prova não é duplicado. Em vez disso, cada tipo de derivação pode ser simplesmente visto com o *status* binário (bem sucedido ou falho) ligado à derivação padrão de um modo que em árvores T, o *status* dos nós rotulados com literais envolvidos na recursão infinita através da negação default são falhos e em árvores TU, esses nós são bem sucedidos. Uma implementação do meta-interpretador para SLX_{EI} é mostrada no apêndice B.

Capítulo 7

Conclusão

Neste trabalho, é definido o conceito de programação em lógica estendida da inconsistência epistêmica, unindo contribuições de duas importantes áreas de Inteligência Artificial: programação em lógica e formalização do raciocínio não monotônico. Em particular, é promovida a introdução no âmbito de programação em lógica de resultados obtidos em *IDL-LEI* enquanto que é proporcionado a *IDL-LEI* mecanismos automáticos de prova computacionalmente eficientes.

Programas em lógica estendidos da inconsistência epistêmica ampliam a visão tradicional de programas em lógica, introduzindo a negação explícita, o operador epistêmico definido em *LEI*, uma nova leitura para a regra da negação como falha e um tratamento paraconsistente das contradições. Tais programas são interpretados declarativamente por $WFSX_{EI}$ e proceduralmente por SLX_{EI}.

Reconhecidamente, $WFSX_P$ é uma das semânticas mais adequadas para programas em lógica estendidos [21]. Na definição de $WFSX_{EI}$, buscou-se preservar características estruturais de $WFSX_P$ como por exemplo a coerência. Em boa medida, pode-se afirmar que as únicas diferenças entre $WFSX_{EI}$ e $WFSX_P$ são decorrentes da introdução do operador epistêmico "?" em programas em lógica estendidos da inconsistência epistêmica.

Para ressaltar essa simetria entre ambas as semânticas, assim como na definição de $WFSX_P$ em [20], são demonstradas três definições equivalentes para $WFSX_{EI}$. No entanto, $WFSX_{EI}$ corrige os problemas verificados em $WFSX_P$, valendo-se, para tanto, do princípio da exceção primeiro, do operador epistêmico "?" e da interpretação de regras com literais defaults como defaults no sentido de *IDL-LEI*.

Como evidenciado em [66, 67, 65, 54], no tocante à interação de regras de inferência conflitantes não monotônicas, há um mal entendimento generalizado na comunidade de pesquisadores em lógica. Importantes elucidações nesse sentido foram elaboradas por Marcelino Pequeno em [65] com a definição do princípio da exceção primeiro. Por estar de acordo com esse princípio, em *IDL-LEI*, não é verificado o problema da extensão

anômala que ocorre na lógica default de Reiter.

Uma das originalidades deste trabalho incide na transcrição dessa discussão para o mundo da programação em lógica. De fato, das semânticas não monotônicas paraconsistentes pesquisadas, $WFSX_{EI}$ é a única a estar de acordo com o princípio da exceção primeiro. Como conseqüência, $WFSX_{EI}$ apresenta soluções mais significativas do que essas semânticas. A grosso modo, pode-se afirmar que $WFSX_{EI}$ está para *IDL-LEI* assim como $WFSX_P$ está para a lógica default de Reiter.

Além disso, importantes resultados tanto de cunho teórico quanto prático foram obtidos. Um desses resultados é a demonstração da corretude de $WFSX_{EI}$ com respeito à intersecção de todas as extensões (quando existirem) da teoria *IDL-LEI* correspondente. Isso quer dizer que $WFSX_{EI}$ proporciona mecanismos corretos para o cálculo da intersecção de todas as extensões de uma importante classe de teorias *IDL-LEI*. Ademais, assim como $WFSX_P$, o problema da decisão em $WFSX_{EI}$ possui complexidade linear para programas sem símbolos funcionais.

Voltando-se para aspectos mais operacionais, é definido e implementado SLX_{EI} para programas básicos a partir de *SLX* [3, 20]. Um outro importante resultado é a demonstração da corretude e completude de SLX_{EI} com relação a $WFSX_{EI}$. Desse modo, SLX_{EI} pode ser entendido como a contrapartida procedural para a apresentação declarativa de $WFSX_{EI}$.

7.1 Contextualização

Esta dissertação insere-se dentro de um sólido corpo de pesquisas em formalização do raciocínio que vem sendo desenvolvido na Laboratório de Inteligência Artificial (LIA) da Universidade Federal do Ceará (UFC), envolvendo os pesquisadores Tarcísio Pequeno, Marcelino Pequeno, Ana Teresa Martins e os colaboradores mais próximos Jean-Yves Béziau e Arthur Buchsbaum. A dissertação foi amplamente beneficiada, em sua elaboração, do ambiente profícuo proporcionado pelo projeto PROTEM-LOGIA, que envolveu sete instituições no Brasil e duas no exterior [51].

Como vem sendo destacado nos últimos anos, o grupo de Lógica do LIA possui importantes contribuições no estudo da compreensão e formalização do raciocínio do senso comum [66, 67, 65, 54]; em especial, na definição do princípio da exceção primeiro e no entendimento da paraconsistência como uma propriedade complementar à não monotonicidade.

Especificamente, no contexto das pesquisas desenvolvidas no LIA, esta dissertação insere-se como uma aplicação para programas em lógica dos resultados obtidos pelo grupo. Isso proporcionou a definição de uma ferramenta computacionalmente eficiente que não

apresenta muitos dos problemas verificados em outras aplicações similares..

7.2 Trabalhos Futuros

Dando continuidade a este trabalho, deve-se apresentar uma definição de $WFSX_{EI}$ para programas com variáveis. Com base nos resultados obtidos para $WFSX_P$, conjectura-se que ambas as versões de $WFSX_{EI}$ (com e sem variáveis) sejam equivalentes. Para alcançar uma análise meramente lógica, será definida uma teoria de modelos baseada numa lógica multivalorada para $WFSX_{EI}$. Também serão examinadas maneiras de embutir $WFSX_{EI}$ em WFS.

Em [25, 24], Dix exibiu uma série de propriedades estruturais que idealmente uma semântica de programação em lógica deveria possuir. Das semânticas paraconsistentes de programas em lógica analisadas, apenas $WFSX_P$ satisfaz todas essas propriedades. Devido a sua simetria com $WFSX_P$, conjectura-se que $WFSX_{EI}$ também esteja de acordo com todas essas propriedades. Esse é um importante resultado a ser demonstrado.

Como é sabido, semânticas paraconsistentes de programas em lógica são comumente vistas como um passo intermediário para alcançar a revisão de crença. Nesse sentido, pretende-se introduzir mecanismos de revisão de crença na definição de $WFSX_{EI}$ para restaurar a consistência quando isso for adequado.

Por estar relacionada a *IDL-LEI*, em $WFSX_{EI}$, o operador epistêmico "?" não é adjuntivo, ou seja, de $\alpha \leftarrow (\beta_1, \ldots, \beta_n)?$ não se obtém $\alpha \leftarrow \beta_1?, \ldots, \beta_n?$. Mesmo sendo desejado em alguns casos, esse caráter não adjuntivo conduz a resultados pouco intuitivos[1]. Entretanto, num trabalho recente [13], Arthur Buschsbaum, Tarcísio Pequeno e Marcelino Pequeno definiram uma nova versão para *LEI* que não mais apresenta esse problema. Um interessante trabalho futuro seria a apresentação de uma nova definição para $WFSX_{EI}$ baseada nessa nova definição de *IDL-LEI*.

No tocante a SLX_{EI}, uma imediata melhoria passa pela introdução de mecanismos para detectar a existência de laços na derivação e com isso, garantir a terminação para programas finitos. Em [20], esse problema é tratado com a introdução de regras que podam o espaço de busca das árvores de refutação, eliminando tanto as recursões cíclicas positivas quanto as recursões cíclicas através da negação default.

Além dos propósitos já mencionados, junto com a construção deste trabalho, existe o objetivo subjacente, mas nem por isso menos relevante, de ressaltar a importância do raciocínio paraconsistente e em especial, da naturalidade e intuitividade do princípio da exceção primeiro e do operador epistêmico "?". Com a $WFSX_{EI}$, espera-se também ter alcançado esse objetivo.

[1] Confira seção A.2.

Referências Bibliográficas

[1] J. F. L. Alcântara and M. Pequeno. *$WFSX_{EI}$* - uma semântica bem fundada estendida da inconsistência epistêmica. In Díbio Leandro Borges and Ariadne Carvalho, editors, *Anais do II Encontro Nacional de Inteligência Artificial (ENIA)*, pages 255–271. Edições EntreLugar, 1999.

[2] J. J. Alferes. *Semantics of Logic Programs with Explicit Negation*. PhD thesis, Univ. Nova de Lisboa, 1993.

[3] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A top-down derivation procedure for programs with explicit negation. In M. Bruynooghe, editor, *Procedure of the International Logic Programming Symposium '94*, pages 424–438. MIT Press, 1994.

[4] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.

[5] J. J. Alferes and L. M. Pereira. Contradiction: when avoidance equal removal. part i. In R. Dyckhoff, editor, *4th International Workshop on Extensions of Logic Programming*, volume 798 of *LNAI*, pages 11–23. Springer-Verlag, 1994.

[6] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 798 of *LNAI*. Springer-Verlag, 1996.

[7] K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, 1997.

[8] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. ACM*, 3(29):841–862, Julho 1982.

[9] N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern Uses of Many-valued Logic*, pages 8–37. Reidel, 1977.

[10] N. Bidoit and C. Froidevaux. General logic databases and programs: default logic semantics and stratification. *Journal of Information and Computation*, 1988.

[11] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

[12] F. Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proceedings of the Symposium on Principles of Database Systems*, pages 34–50. ACM SIGACT-SIGMOD, 1989.

[13] A. Buchsbaum, T. Pequeno, and M. Pequeno. The inconsistent logic of plausibility. (A publicar).

[14] A. R. Buchsbaum. *Lógicas das inconsistências e incompletude: semântica, axiomatização e automação.* PhD thesis, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 1995.

[15] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Inc, 1987.

[16] A. Church. An unsolvable problem of number theory. *Amer. J. Math*, 58:345–363, 1936.

[17] K. Clark. Negation as failure. In H. Gallare and J. Minker, editors, *Logic and data bases*, pages 293–322. Plenum Press, 1978.

[18] K. L. Clark. Predicate logic as a computacional formalism. Research report doc 79/59, Department of Computing, Imperial College, 1979.

[19] A. Colmerauer, H. Kanoui, R. Passero, and P. Russel. Un systeme de communication homme-machine en français. Technical report, Groupe de Recherche en Intelligence Artificielle, Universite d'Aix-Marseille, 1973.

[20] C. V. Damásio. *A paraconsistent semantics with constraints.* PhD thesis, Univ. Nova de Lisboa, 1996.

[21] C. V. Damásio and L. M. Pereira. A survey on paraconsistent semantics for extended logic programs. In Gabbay and Smets, editors, *Handbook on quantitative reasoning: negation and inconsistency*, volume 2. Oxford University Press, 1998.

[22] M. Davis. First order logic. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 31–65. Oxford Science Publications, 1993.

[23] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[24] J. Dix. A classification-theory of semantics of normal logic programs. *Fundamenta Informaticae*, XXII(3):257–288, 1985.

[25] J. Dix. A classification-theory of semantics of normal logic programs. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[26] P. M. Dung and P. Ruamviboonsuk. Well founded reasoning with classical negation. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non Monotonic Reasoning*, pages 120–132. MIT Press, 1991.

[27] M. Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 4 1985.

[28] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[29] M. Gelfond and V. Lifschitz. Logic programs with classic negation. In Waren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT press, 1990.

[30] M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In K. Apt, editor, *International Joint Conference and Symposium on Logic Programming*, pages 559–573. MIT Press, 1992.

[31] P. C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM J. Res. Develop.*, 4:28–35, 1960.

[32] M. L. Ginsberg. Multivalued logics. In *Proceedings of AAAI'86*, pages 243–247, 1986.

[33] M. L. Ginsberg. Mulivalued logic: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[34] K. Gödel. Uber die vollstandigkeit des logikkalkuls,. godel thesis, 1929. reimpresso e traduzido para o inglês como "on the completeness of the calculus of logic".

[35] K. Gödel. Die vollstandigkeit der axiome des logischen funktionenkalküls. In *Monastshefte für Mathematik und Physik*, volume 37, pages 349–360. Reimpresso e traduzido para o inglês como "the completeness of the axioms of the functional calculus of logic", 1986, pp. 102-123 edition, 1930.

[36] C. Green. Theorem proving by resolution as a basis for question - answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 183–205. American Elsevier, New York, 1969.

[37] S. Hans and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[38] P. Hayes. Computation and deduction. In *Proceedings of the Second Symposium on Mathematical Foundations of Computer Science*, pages 105–118, Czechoslovakia, 1973. Czechoslovakian Academy of Sciences.

[39] J. Herbrand. Investigations in proof theory. In J. van Heijenoort, editor, *From Frege to Gödel: a source book in mathematical logic, 1879-1931*, pages 525–581. Harvard Universty Press, Cambridge, Mass., 1967.

[40] R. Hill. Lush resolution and its completeness. Technical Report TR DCL Memo 78, Department of Artificial Intelligence, University of Edinburgh, 1974.

[41] K. Inoue. Extended logic programs with default assumptions. In Koichi Furukawa, editor, *8th International Conference on Logic Programming*, pages 490–504. MIT Press, 1991.

[42] R. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7), 1979.

[43] R. Kowalski. Problems and promises of computational logic. In John Lloyd, editor, *Computational Logic*, Basic Research, pages 1–36. Springer-Verlag, 1990.

[44] R. Kowalski and P. Hayes. Semantic trees in automatic theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 87–101. American Elsevier, New York, 1969.

[45] R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*. MIT Press, 1990.

[46] R. A. Kowalski. Predicate logic as a programming language. In *Information Processing 74*, pages 569–574, Stockholm, 1974. North Holland.

[47] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

[48] K. Kunen. Some remarks on the completed database. In R. Kowalski and K. A. Bowen, editors, *5th International Conference of Logic Programming*, pages 978–992. MIT Press, 1988.

[49] A. Leitsch. *The resolution calculus*. Springer-Verlag, 1997.

[50] J. Lloyd. *Foundation of Logic Programming*. Springer-Verlag, second edition, 1987.

[51] LOGIA. Relatório final do projeto LOGIA - Métodos formais para raciocínio e representação do conhecimento. URL: http://www.lia.ufc.br/~logi, 1996.

[52] D. W. Loveland. A linear format for resolution. In *Proc. IRIA Symposium on Automatic Demonstration*, Lecture Notes in Mathematics 125, pages 147–162. Springer, 1970.

[53] D. Luckham. Refinements in resolution theory. In *Proc. IRIA Symposium on Automatic Demonstration*, Lecture Notes in Mathematics 125, pages 163–190. Springer, 1970.

[54] A. Martins. *A syntatical and semantical uniform treatment for the IDL e LEI nonmonotonic system*. PhD thesis, Universidade Federal de Pernambuco (UFPE), 1998.

[55] J. McCarthy. Programs with commom sense. In *Proc. of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office.

[56] J. McCarthy. Circumscription: a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[57] J. McKinsey. The decision problem for some classes of sentences without quantifiers. *J. Symbolic Logic*, 8:61–76, 1943.

[58] R. C. Moore. Possible-word semantics for autoepistemic logic. In *AAAI Workshop on Non-monotonic Reasoning*, pages 396–401, New Platz, 1984.

[59] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[60] P. H. Morris. The anomalus extension problem in default reasoning. *Artificial Intelligence*, 35, 1988.

[61] D. Nelson. Constructive falsity. *JSL*, 14:16–26, 1949.

[62] D. Pearce. Answer sets and constructive logic, ii: Extended logic programs and related nonmonotonic formalisms. In L. M. Pereira and A. Nerode, editors, *2nd Workshop International on Logic Programming and Nonmonotonic Reasoning*, pages 457–475. MIT Press, 1993.

[63] D. Pearce and G. Wagner. Reasoning with negative information i: Strong negation in logic programs. In L. Haaparanta, M. Kusch, and I. Niiniluoto, editors, *Language, Knowledge and Intentionality*, pages 430–453. Acta Philosophica Fennica 49, 1990.

[64] D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, LNAI 475, pages 311–326. Springer-Verlag, 1991.

[65] M. Pequeno. *Defeasible reasoning with exceptions first.* PhD thesis, Imperial College of Science, Technology and Medicine - University of London, 1994.

[66] T. Pequeno. A logic for inconsistent nonmonotonic reasoning. Technical report, Department of Computing - Imperial College, London, 1990.

[67] T. Pequeno and A. Buchsbaum. The logic of epistemic inconsistency. In *Second International Conference on Principles of Knowledge Representation e Reasoning*, Cambridge, MA, USA, 1991.

[68] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, Viena, Austria, 1992.

[69] L. M. Pereira, J. J. Alferes, and J. N. Aparício. A practical introduction to well founded semantics. In B. Mayoh, editor, *Scandinavian Conference on Artificial Intelligence*, pages 566–577. IOS Press, 1991.

[70] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Default theory for well founded semantics with explicit negation. In D. Pearce and G. Wagner, editors, *Logics in AI. Proceedings of the European Workshop JELIA'92*, pages 339–356. Springer-Verlag, 1992.

[71] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Counterfactual reasoning based on revising assumptions. In Ueda and Saraswat, editors, *International Logic Programming Symposium*, pages 566–577. MIT Press, 1991.

[72] L. M. Pereira, J. N. Aparício, and J. J. Alferes. A derivation procedure for extended stable models. In *International Joint Conference on Artificial Intelligence*, Sydney, Austrália, 1991. Morgan Kaufmann.

[73] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Nonmonotonic reasoning with well founded semantics. In Koichi Furukawa, editor, **th International Conference on Logic Programming*, pages 475–489. MIT Press, 1991.

[74] S. G. Pimentel and W. L. Rodi. Belief revision and paraconsistency in a logic programming framework. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-monotonic Reasoning*, pages 228–242. MIT Press, 1991.

[75] T. Przmusinski. A semantics for disjunctive logic programs. In Loveland, Lobo, and Rajasekar, editors, *ILPS'91*, 1991.

[76] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techiniques in Artificial Intelligence, a Sourcebook*, pages 321–367. North Holland, 1990.

[77] H. Przymusinska, T. C. Przymusinski, and H. Seki. Soundness and completeness of partial deductions for well-founded semantics. In A. Voronkov, editor, *Proc. of the International Conference on Logic Programming and Automated Reasoning*, LNAI 624. Springer-Verlag, 1992.

[78] T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.

[79] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[80] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Proceedings of the Eighth Symposium on Principles of Database Systems*, pages 11–21. ACM SIGACT-SIGMOD, 1989.

[81] T. C. Przymusinski. Three-valued non-monotonic formalisms and logic programming. In R. Brachman, H. Leveque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 341–348. Morgan Kaufmann, 1989.

[82] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[83] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[84] J. A. Robinson. The generalized resolution principle. In D. Michie, editor, *Machine Intelligence*, volume 3, pages 77–94. American Elsevier, New York, 1968.

[85] C. Sakama. Extended well-founded for paraconsistent logic programs. In *Fifth Generation Computer System*, pages 592–599. ICOT, 1992.

[86] C. Sakama. *Studies on disjunctive logic programming.* PhD thesis, Faculty of Engineering of Kyoto University, Julho 1994.

[87] C. Sakama and K. Inoue. Paraconsistent stable semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.

[88] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[89] F. Teusink. A proof procedure for extended logic programs. In *ILPS'93*. MIT Press, 1993.

[90] A. M. Turing. On computable numbers, with aplications to the entscheindungsproblem. In *Proceedings of London Math Society*, volume 42, pages 230–265, 1936.

[91] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[92] A. van Gelder. The alternating fixpoint of logic programs with negation. In *8th Symposium on Principles of Database Systems*, pages 185–221. ACM SIGACT-SIGMOD, 1989.

[93] A. van Gelder, K. A. Rossi, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[94] G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H-D. Gerhardt, editors, *Mathematical Foundations of Database Systems*, LNCS 495, pages 357–371. Springer-Verlag, 1991.

[95] G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic and Computation*, 1(6):835–859, 1991.

[96] G. Wagner. Reasoning with inconsistency in extended deductive databases. In L. M. Pereira and A. Nerode, editors, *2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 300–315. MIT Press, 1993.

[97] G. Wagner. Vivid logic: Knowledge-based reasoning with two kinds of negation. In *Lectures Notes on Artificial Intelligence*, page 764. 1994.

Capítulo A

Lógica Default, LEI e IDL

A.1 Lógica Default de Reiter

A linguagem da lógica default no sentido de Reiter é obtida da linguagem da lógica de primeira ordem ($\mathcal{L}$) acrescida de uma nova entidade sintática, um default, que possui a seguinte forma genérica:

$$\frac{A : B}{C},$$

em que A, B e C são fórmulas da lógica de primeira ordem. A é chamado de *pré-requisito* do default; B é a *justificativa* do default e C é o seu *conseqüente* ou *conclusão*. Um default é *fechado* sss A, B e C são sentenças.

Um default *normal* é da forma

$$\frac{A : B}{B}$$

e um default *seminormal* é da forma

$$\frac{A : B \wedge D}{B},$$

em que A, B e D são fórmulas da lógica de primeira ordem.

Definição A.1 (Teoria default) *Uma teoria default Δ no sentido de Reiter é um par $\langle W, D\rangle$, em que W é um conjunto de fórmulas fechadas da lógica de primeira ordem, utilizado para a representação das informações absolutas (base monotônica) e D é um conjunto de defaults, utilizado para a representação das informações retratáveis (base não monotônica). Uma teoria default é fechada se todos os seus defaults são fechados.*

Uma teoria default é *normal* se todos os seus defaults são normais. Teorias defaults que contêm somente defaults semi-normais ou normais são chamadas de *semi-normais*. Se a teoria possui pelo menos um default que não seja normal nem seminormal, é chamada de *arbitrária*.

Intuitivamente, um default fechado pode ser compreendido como uma regra de inferência tal que se A é provado na teoria e B é consistente com esta teoria, então C é também provado na teoria. Observe que a negação da justificativa trabalha como uma premissa negativa para a inferência do conseqüente (B é consistente com a teoria sss $\neg B$ não pertence à teoria).

Na definição abaixo, é apresentado o conceito de dedutivamente fechado:

Definição A.2 *Seja* Γ *um conjunto de fórmulas fechadas de uma linguagem* $\mathcal{L}$ *e* γ *uma fórmula fechada de* $\mathcal{L}$. *Então* $Th(\Gamma) = \{\gamma \mid \Gamma \vdash \gamma\}$. Γ *é dedutivamente fechado sss* $Th(\Gamma) = \Gamma$.

O conjunto de teoremas de uma teoria default é chamado de *extensão*. Uma extensão pode ser vista como um conjunto dedutivamente fechado de crenças induzidas pela regra default da teoria como uma maneira de completar as coleções parciais W de fatos do mundo. Segundo Reiter, uma extensão E para uma teoria default $\Delta = \langle W, D\rangle$ deve satisfazer as seguintes propriedades:

- $W \subseteq E$;
- E é dedutivamente fechado ($Th(E) = E$);
- E é maximal com respeito aos defaults aplicáveis, ou seja, para um default $\frac{A\ :\ B}{C}$, se $A \in E$ e $\neg B \notin E$, então $C \in E$.

A partir dessa intuição, em teorias defaults fechadas, uma extensão pode ser definida como segue.

Definição A.3 *[82] Seja* $\Delta = \langle W, D\rangle$ *uma teoria default fechada. Para qualquer conjunto* S *de fórmulas bem formadas e fechadas, seja* $\Gamma(S)$ *o menor conjunto satisfazendo as seguintes três propriedades:*

- $W \subseteq \Gamma(S)$;
- $Th(\Gamma(S)) = \Gamma(S)$;
- *Se* $\frac{A\ :\ B}{C} \in D$, $A \in \Gamma(S)$ *e* $\neg B \notin S$, *então* $C \in \Gamma(S)$.

Um conjunto de fórmulas bem formadas e fechadas E *é uma extensão para* Δ *sss* $\Gamma(E) = E$, *ou seja,* E *é um ponto fixo do operador* Γ.

Alternativamente, uma extensão para uma teoria default pode ser definida da seguinte maneira:

Teorema A.4 *[82] Sejam $E \subseteq \mathcal{L}$ um conjunto de fórmulas bem formadas e fechadas e $\Delta = \langle W, D\rangle$ uma teoria default fechada. Define-se*

$$\begin{aligned} E_0 &= W \\ E_{i+1} &= Th(E_i) \cup \{C \mid \frac{A:B}{C}\} \in D, \text{ em que } i > 0, A \in E_i \text{ e } \neg B \notin \bigcup_{i=0}^{\infty} E_i \end{aligned}$$

E é uma extensão para Δ sss $E = \bigcup_{i=0}^{\infty} E_i$.

Exemplo A.5 Seja $\Delta = \langle W, D\rangle$ uma teoria default no sentido de Reiter com

$W = \{A, B\}$

$D = \{\frac{A:C}{C}, \frac{B:\neg C}{\neg C}\}$, em que A, B e C é a representação proposicional para fórmulas de primeira ordem.

Das definições acima, tem-se que essa teoria possui duas extensões:

$E_1 = Th(W \cup C)$

$E_2 = Th(W \cup \neg C)$

A.2 LEI

Diz-se que existe uma *inconsistência ontológica* na descrição de um estado de coisas quando uma inconsistência é intrínseca a esse estado de coisas. Por outro lado, diz-se que existe uma *inconsistência epistêmica* na descrição de um estado de coisas quando uma inconsistência é resultado da falta de conhecimento sobre o estado de coisas.

Para formalizar a noção de inconsistência epistêmica, é apresentada a *LEI*(*Logic of Epistemic Inconsistency*). Trata-se de um lógica paraconsistente que extrai conclusões significativas de contradições resultantes do raciocínio sobre o conhecimento incompleto. As definições abaixo seguem como em [54].

A.2.1 Definições

A linguagem de *LEI*, denotada por $\mathcal{L}_?$, é definida a partir da linguagem $\mathcal{L}$ da lógica clássica de primeira ordem acrescida de fórmulas sufixadas pelo operador epistêmico ?, que são utilizadas para representar as declarações plausíveis. A linguagem $\mathcal{L}$? é definida sobre um alfabeto $\mathcal{A}$, que é mostrado abaixo:

Definição A.6 (Alfabeto $\mathcal{A}$) *Seja $\mathcal{A} = \langle \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P}, \mathcal{O}, \mathcal{Q}, \{(,)\}\rangle$ em que*

- *$\mathcal{V}$ é o conjunto contável de variáveis $x, y, z, \ldots$;*

- $\mathcal{C}$ *é o conjunto contável de símbolos constantes* $a, b, c, \ldots$;
- $\mathcal{F}$ *é o conjunto contável de símbolos funcionais* $f, g, h, \ldots$;
- $\mathcal{P}$ *é o conjunto contável de símbolos predicados* $P, P_0, P_1, \ldots$;
- $\mathcal{O}$ *é o conjunto de conectivos* $\{\vee, \wedge, \neg, \Rightarrow$ [1], $?\}$;
- $\mathcal{Q}$ *é o conjunto de quantificadores* $\{\forall, \exists\}$;
- $\{(,)\}$ *é o conjunto de símbolos de pontuação.*

Termos são definidos como usualmente. As fórmulas da lógica clássica de primeira ordem são chamadas de *fórmulas ?-livre* e serão denotadas por letras romanas maiúsculas $(A, B, C, \ldots)$.

Definição A.7 (?-Fórmula: $\alpha, \beta, \gamma, \ldots$**)**

1. *Se* A *é um fórmula ?-livre, então* A *é uma ?-fórmula;*
2. *Se* α *é uma ?-fórmula, então* $\neg\alpha$ *e* $\alpha?$ *são ?-fórmulas*;
3. *Se* α *e* β *são ?-fórmulas, então* $\alpha \vee \beta, \alpha \wedge \beta$ *e* $\alpha \Rightarrow \beta$ *são fórmulas*;
4. *Se* α *é uma ?-fórmula e* x *é uma variável, então* $\forall x\alpha$ *e* $\exists x\alpha$ *são ?-fórmulas*;
5. *Nada mais é ?-fórmula.*

Numa ?-fórmula α, "?" pode ou não ocorrer. ?-fórmulas serão simplesmente chamadas de fórmulas *LEI* e serão denotadas pelas letras gregas $\alpha, \beta, \gamma, \ldots$.

Definição A.8 (Linguagem $\mathcal{L}_?$ **da LEI)** *A linguagem da LEI é o conjunto de fórmulas LEI ou equivalentemente o conjunto de ?-fórmulas.*

Além disso, é definido o conceito de fórmulas ?-fechadas:

Definição A.9 (Fórmula ?-fechada) *Uma fórmula* α *é fechada se toda fórmula atômica em* α *está sob o escopo de ?. Formalmente*

1. $\alpha?$ *é uma fórmula ?-fechada*;
2. *Se* α *e* β *são fórmulas ?-fechadas, então* $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \Rightarrow \beta$, $\forall x\alpha$ *e* $\exists x\alpha$ *são fórmulas ?-fechadas*;

[1] *Em* [67], *é utilizado o símbolo* $\rightarrow$.

Em *LEI*, são identificados dois tipos de contradição: ontológica e epistêmica.

Definição A.10 (Contradição ontológica $\perp$) $\perp = \alpha \wedge \neg A$ *ou* $\perp = A \wedge \neg\alpha$, *em que* A *é obtido de* α *retirando todas as ocorrências, se existir alguma, de* ? *em* α.

Definição A.11 (Contradição epistêmica) $\alpha \wedge \neg\alpha$, *em que há pelo menos uma ocorrência de* ? *em* α.

Na apresentação dos axiomas abaixo, contradições epistêmicas serão paraconsistentemente toleradas enquanto que a contradição ontológica irá conduzir a trivialização do cálculo axiomático.

A.2.2 Axiomas e Regras

Para a formulação de uma axiomática para a *LEI*, em [14], foram apresentados certos princípios. Nesta seção, são destacados alguns deles:

- $\neg$ deve comportar-se como a negação clássica para fórmulas ?-livre e como a negação paraconsistente para fórmulas com "?". Desde que tais fórmulas são pretendidas representar conclusões plausíveis, um conflito entre elas não é forte o suficiente para trivializar e deve ser paraconsistentemente tolerado;
- refletindo o fato que todo conhecimento irrefutável é também plausível, deve-se ter $\alpha \Rightarrow \alpha?$ como uma tese em *LEI*;
- plausibilidade não acarreta irrefutabilidade, ou seja, $\alpha? \Rightarrow \alpha$ não deve ser teorema em *LEI*;
- se α é uma fórmula ?-fechada, então $\alpha \Leftrightarrow \alpha?$ deve ser um teorema em *LEI*. Isso que dizer que ? adicionais são supérfluos.

Por convenção, será utilizado $\sim \alpha$ para representar fórmulas do tipo $\alpha \Rightarrow p \wedge \neg p$, em que α é uma fórmula *LEI* e p é uma letra proposicional arbitrária. Feitas essas considerações, é apresentada abaixo os axiomas e regras de inferência de *LEI* para o caso proposicional:

- Axiomas Proposicionais Básicos
  1. $(\Rightarrow -1) \quad \alpha \Rightarrow (\beta \Rightarrow \alpha)$
  2. $(\Rightarrow -2) \quad (\alpha \Rightarrow \beta) \Rightarrow ((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow (\alpha \Rightarrow \gamma))$
  3. (MP) $\quad \dfrac{\alpha,\ \alpha \Rightarrow \beta}{\beta}$

4. $(\wedge - 1)$ $\alpha \wedge \beta \Rightarrow \alpha$
5. $(\wedge - 2)$ $\alpha \wedge \beta \Rightarrow \beta$
6. $(\wedge - 3)$ $\alpha \Rightarrow (\beta \Rightarrow \alpha \wedge \beta)$
7. $(\vee - 1)$ $\alpha \Rightarrow (\alpha \vee \beta)$
8. $(\vee - 2)$ $\beta \Rightarrow (\alpha \vee \beta)$
9. $(\vee - 3)$ $(\alpha \Rightarrow \gamma) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\alpha \vee \beta \Rightarrow \alpha))$
10. $(\Rightarrow -3)$ $((\alpha \Rightarrow \beta) \Rightarrow \alpha) \Rightarrow \alpha$

- Axiomas Proposicionais para a Negação *LEI* Paraconsistente

1. $(\neg - 1)$ $\neg(\alpha \Rightarrow \beta) \Leftrightarrow (\alpha \wedge \neg\beta)$
2. $(\neg - 2)$ $\neg(\alpha \wedge \beta) \Leftrightarrow (\neg\alpha \vee \neg\beta)$
3. $(\neg - 3)$ $\neg(\alpha \vee \beta) \Leftrightarrow (\neg\alpha \wedge \neg\beta)$
4. $(\neg - 4)$ $\neg\neg\alpha \Leftrightarrow \alpha$

- Axiomas Proposicionais para ?

1. $(? - 1)$ $(\alpha \Rightarrow B) \Rightarrow ((\alpha \Rightarrow \neg B) \Rightarrow \neg\alpha)$
2. $(? - 2)$ $\alpha \Rightarrow \alpha?$
3. $(? - 3)$ $\alpha?? \Rightarrow \alpha?$
4. $(? - 4)$ $(\alpha? \Rightarrow \beta?)? \Rightarrow (\alpha? \Rightarrow \beta?)$
5. $(? - 5)$ $(\alpha \vee \beta)? \Rightarrow (\alpha? \vee \beta?)$
6. $(? - 6)$ $(\neg\alpha)? \Leftrightarrow \neg(\alpha?)$
7. $(? - 7)$ $\dfrac{\alpha \Rightarrow \beta}{\alpha? \Rightarrow \beta?}$
8. $(? - 8)$ $\dfrac{\alpha}{\sim((\sim\alpha)?)}$

Na definição da axiomática de *LEI*, foram mantidas quase todas os axiomas proposicionais básicos da lógica clássica. Entretanto, para garantir um tratamento paraconsistente para fórmulas com ocorrência de ?, *reductio ad absurdum* $((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \neg\beta) \Rightarrow \neg\alpha)$ é restrito às fórmulas ?-livre (axioma $? - 1$). Para recuperar parte do poder dedutivo da implicação perdido com essa restrição, é introduzida a Lei de Peirce (axioma $\Rightarrow -3$) como axioma de *LEI*. Enquanto isso, na lógica clássica, a Lei de Peirce é derivada de *reductio ad absurdum*.

Essa forma restrita de *reductio ad absurdum* também conduz ao enfraquecimento da negação paraconsistente de *LEI*. Para salvaguardar o poder dedutivo de $\neg$, são introduzidos na axiomática de *LEI*, alguns teoremas clássicos como o da dupla negação e o da distuibuição de $\neg$ sobre $\Rightarrow$, $\wedge$ e $\vee$.

Os axiomas MP, $(? - 7)$ e $(? - 8)$ são regras de inferência. Desses axiomas, apenas MP pode ser equivalentemente escrito sob a forma implicativa $(\alpha \wedge (\alpha \Rightarrow \beta) \Rightarrow \beta)$. Para ressaltar essa diferença, é utilizada uma barra dupla nas regras $(? - 7)$ e $(? - 8)$.

Casos as regras de inferência $(? - 7)$ e $(? - 8)$ fossem escritas sob a forma implicativa, $\alpha? \Rightarrow \alpha$ seria um teorema de *LEI*, ou seja, obter-se-ia o resultado indesejado que conhecimento plausível implica conhecimento irrefutável.

Em conjunção com *modus ponens* (MP), a regra $(? - 7)$ possibilita a derivação de $\beta?$ de $\alpha \Rightarrow \beta$ e α. Esse tipo de raciocínio representa o principal mecanismo de propagação de "?" através de inferências.

Algumas das principais propriedades da axiomática de *LEI* são mostradas abaixo. Suas respectivas demonstrações podem ser encontradas em [14].

Teorema A.12 *Todo teorema da lógica clássica também é um teorema para as fórmulas ?-livre em LEI.*

Corolário A.13 *A negação $\neg$ comporta-se classicamente para fórmulas ?-livre em LEI.*

1. $\vdash (A \Rightarrow B) \Rightarrow ((A \Rightarrow \neg B) \Rightarrow \neg A)$
2. $\vdash \neg\neg A \Rightarrow A$

Teorema A.14 *O símbolo $\sim$ realmente se comporta como a negação clássica.*

1. $\vdash (\alpha \Rightarrow \beta) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow \sim \alpha$
2. $\vdash \sim\sim \alpha \Rightarrow \alpha$

Teorema A.15 *Toda regra de introdução e eliminação de $\wedge$ e $\vee$ utilizada na lógica clássica também é válida em LEI.*

A.3 IDL

Em [66], é argumentado que uma conclusão resultante de um default (conclusão refutável) nunca deve ter o mesmo *status* epistêmico de uma conclusão irrefutável, obtida por dedução. Para assinalar essa diferença, é definida a teoria default *IDL* (*Inconsistent Default Logic*), em que a conclusão de todo default é sufixada com o operador epistêmico "?".

Definição A.16 *Uma teoria default IDL é um par $\langle W, D\rangle$, em que W é uma coleção de fórmulas da linguagem $\mathcal{L}_?$ de LEI e D é um conjunto de regras defaults do tipo*

$$\frac{A : B; C}{B?},$$

em que A, B, C são fórmulas de $\mathcal{L}_?$. A é chamado de pré-requisito; B é a justificativa normal, C é a justificativa seminormal e $B?$ é a conclusão ou conseqüente. Um default é fechado sss A, B, e C não possuem nenhuma ocorrência de variáveis livres[2]*. Uma teoria default IDL é fechada se todos os seus defaults são fechados.*

Em *IDL* é implementada a idéia de acomodar visões conflitantes numa mesma extensão. Assim, se $\frac{A\colon B;\ C}{B?}$ é uma regra default de uma teoria *IDL*, $\neg B?$ não bloqueia a aplicação dessa regra. Para tanto, seria necessário $\neg B$. Por seu turno, pela sua parte seminormal, $\neg C?$ é o suficiente para impedir a aplicação dessa regra. Essa caraciterística permite que *IDL* esteja de acordo com o *princípio da exceção primeiro* [65], que privilegia a derivação da exceção da regra default em detrimento da aplicação da própria regra. Caracteristicamente, a lógica default de Reiter não está de acordo com esse princípio. Uma outra diferença de *IDL* com relação à lógica default de Reiter é que *IDL* mantém conclusões contraditórias epistêmicas $\beta?$ e $\neg(\beta?)$. Essa tipo de contradição é tolerado sem trivializar, permitindo agrupar essa contradição numa única extensão.

Assim como na teoria default de Reiter, as extensões de uma teoria *IDL* podem ser calculadas a partir de uma definição de ponto fixo. Para apresentar essa definição, algumas convenções foram adotadas:

Definição A.17 (Operadores conseqüência LEI e IDL)

- *Operador conseqüência LEI:* $Cn : 2^{\mathcal{L}_?} \mapsto \mathcal{L}_?$
- *Operador conseqüência IDL:* $C_D : 2^{\mathcal{L}_?} \mapsto \mathcal{L}_?$

Desde que defaults não são fórmulas, e sim, regras, será utilizado uma forma indexada para operadores conseqüência *IDL* com o conjunto D da teoria default subjacente como um subescrito. A relação de sequentes *IDL* também será indexada por D:

Definição A.18 (Relação de seqüentes LEI e IDL)

- *Relação de seqüentes LEI:* $\vdash$: $2^{\mathcal{L}_?} \times 2^{\mathcal{L}_?}$
- *Relação de seqüentes IDL:* $\mid\!\sim_D$: $2^{\mathcal{L}_?} \times 2^{\mathcal{L}_?}$

[2] *Variáveis livres em $\mathcal{L}_?$ são definidas como na lógica clássica: confira definição 2.5.*

Os operadores conseqüência estão relacionados com as relações de seqüentes pelas seguintes equações:

Definição A.19 (Corretude e completude da relação de seqüente)

- $\alpha \in Cn(\Gamma)$ *sss* $\Gamma \vdash \alpha$
- $\alpha \in C_D(W)$ *sss* $\mid\!\sim_D \alpha$

Definição A.20 (Extensão IDL) *Sejam $\mathcal{L}_?$ uma linguagem de primeira ordem para a LEI; D, um conjunto de defaults; W, uma coleção de fatos pertencentes a $\mathcal{L}_?$; $\langle W, D\rangle$, uma teoria default IDL fechada; Cn, o operador conseqüência da LEI e C_D, o operador conseqüência IDL sobre qualquer conjunto S de fórmulas em $\mathcal{L}_?$ indexada pelo conjunto D de defaults e definida como segue: $C_D(S)$ é o conjunto minimal de fórmulas em $\mathcal{L}_?$ que satisfaz as seguintes propriedades:*

1. $W \subseteq C_D(S)$;
2. $Cn(C_D(S)) = C_D(S)$;
3. *Se* $\frac{\alpha\ :\ \beta;\ \gamma}{\beta?} \in D$, $\alpha? \in C_D(S)$, $\neg\beta$ *e* $(\neg\gamma)? \notin S$, *então* $\beta? \in C_D(S)$.

Um conjunto de fórmulas E é uma extensão para $\langle W, D\rangle$ sss $C_D(E) = E$, ou seja, E é um ponto fixo do operador C_D.

Exemplo A.21 Seja $\Delta = \langle W, D\rangle$ uma teoria *IDL*, em que

$W = \{A, B\}$

$D = \{\ \frac{A\ :\ C}{C?},\ \frac{B\ :\ \neg C}{\neg C?}\ \}$, em que A, B e C é a representação proposicional para fórmulas de $\mathcal{L}_?$.

Das definições acima, tem-se que essa teoria possui uma única extensão:

$E = Th(W \cup \{C?, \neg C?\})$

Como pode ser constatado no exemplo A.5, o caso acima quando transcrito na lógica default de Reiter possui duas extensões.

Baseado nas definições A.10 e A.11, a noção de consistência e contradição em teorias *IDL* é diferente da apresentada em outras lógicas defaults. De um lado, sabe-se que contradições epistêmicas são paraconsistentemente toleradas sem trivializar uma extensão. Em contrapartida, contradições ontológicas trivializam uma extensão, deixando-a inconsistente.

Definição A.22 (Teoria IDL inconsistente) *Uma teoria default IDL Δ é inconsistente sss ela possui uma extensão inconsistente.*

Embora em *IDL*, conclusões conflitantes sejam agrupadas numa única extensão, em *LEI*, a base monotônica de uma teoria *IDL*, essas conclusões não são misturadas. Isso quer dizer que *LEI* não é adjuntivo ao considerar fórmulas com "?", ou seja, de "$\alpha?$" e "$\beta?$" não se obtém "$(\alpha \wedge \beta)?$", embora se tenha "$\alpha? \wedge \beta?$". Devido a essa propriedade, alguns resultados não intuitivos são evitados como no exemplo abaixo [67].

Exemplo A.23 (Extensões não são misturadas) Tem-se que

$\frac{voa(x) \ : \ rápido(x)}{rápido(x)?}$

$\frac{\neg voa(x) \ : \ prudente(x)}{prudente(x)?}$

$rápido(x) \wedge prudente(x) \Rightarrow perfeito\text{-}viajante(x)$

e as seguintes conclusões defaults *"$voa(Teo)?$" e "$\neg voa(Teo)?$"*.

Utilizando a regra $(? - 7)$ da axiomática de *LEI*, da terceira declaração, obtém-se

$$(rápido(x) \wedge prudente(x))? \Rightarrow perfeito\text{-}viajante(x)?$$

Dos dois defaults acimas e de "$voa(Teo)?$" e "$\neg voa(Teo)?$", obtém-se respectivamente "$(rápido(Teo)$" e "$prudente(Teo))?$". Desse resultado, não é razoável concluir "$perfeito\text{-}viajante(Teo)?$", pois as conclusões defaults "$voa(Teo)?$" e "$\neg voa(Teo)?$" supostamente pertencem a extensões distintas na teoria default de Reiter. Em outras palavras, "$\nvdash_{LEI} \alpha? \wedge \beta? \Rightarrow (\alpha \wedge \beta)?$".

No entanto, às vezes, é desejável combinar conclusões plausíveis defaults que deveriam estar numa mesma extensão. Isso é ilustrado em [67] num outro exemplo: suponha que foram obtidas as conclusões "$Tem\text{-}Casco(Incitatus)?$" e "$Quadrúpede(Incitatus)?$". Ademais, sabe-se que

$$Tem\text{-}Casco(x) \wedge Quadrúpede(x) \Rightarrow Eqüino(x)$$

Desses resultados, é razoável concluir "$Eqüino(Incitatus)?$", mas não é possível no cálculo de *LEI* pelos motivos já explicados no exemplo anterior.

Para resolver esse problema, em trabalho recente [13], Arthur Buschsbaum, Tarcísio Pequeno e Marcelino Pequeno definiram uma nova versão para *LEI*, que apresenta um caráter adjuntivo ao considerar as fórmulas com "?" quando isso for desejável.

Capítulo B

Implementação de SLX_{EI}

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% XSB é um software gratuito que pode ser obtido pelo endereço
% eletrônico
% http://www.cs.sunysb.edu/~sbprolog/xsb-page.html
%
% SLXEI é um pré-processador de programas em lógica da
%       inconsistência epistêmica sob a semântica WFSXEI em
% programas normais equivalentes para XSB.
%
% SLXEI foi desenvolvido a partir da implementação para SLX,
% cujos autores são Jose J. Alferes  (jja@dmat.uevora.pt)  e
%       Luis Moniz Pereira(lmp@di.fct.unl.pt).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Operators definition                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(950,fy , '-' ).           % Explicit negation
:- op(950,fy , not ).           % Negation as failure Bodies
:- op(950,fy , naf ).           % Negation as failure query
:- op(950,fy , und ).           % Undefined query
:- op(950,fy , '?' ).           % Plausible query

:- dynamic toCompile/0, hasTop/1, hasRules/1, inBody/1.

%slxei_comp compila o arquivo.
```

```
slxei_comp(File) :-
     assert(toCompile),
     retractall(hasTop(_)), retractall(hasRules(_)),
     retractall(inBody(_)),
     loadFile(File),
     retract(toCompile),
     compileFileName(File,NameAux),
     retext(NameAux,Name),
     [Name].

compileFileName(File, Name) :-
     name(File,FN),
     name(Name,[116,109,112|FN]).  % tmp....

retext(NameAux,Name):-
     name(NameAux,NamePAscii),
     ((append(NameAscii,[46,112],NamePAscii));
     (append(NameAscii,[46,80],NamePAscii))),
     name(Name,NameAscii),!.

retext(NameAux,NameAux).

loadFile(File) :-
    retractall('$loaded_clause'(_,_,_)),
    see(File),
    load_clause,
    seen,
    predicates_are_loaded(File),
    load_preds,
    end_of_loading.

load_clause :-
    read(C),
    ( C = end_of_file -> true;
      ( process_clause(C), load_clause ) ).

load_preds :-
```

```
    one_template(HH),
    findall( clause(TheClause,Type),
             '$loaded_clause'(HH,TheClause,Type), Clauses),
    t_1(Clauses),
    findall( clause(TheClause,Type),
             '$loaded_clause'(HH,TheClause,Type), ClausesT_1),
    t_2(ClausesT_1),
    findall( clause(TheClause,Type),
             '$loaded_clause'(HH,TheClause,Type), ClausesT_2),
    load_predicate(HH,ClausesT_2),
    apagaTudo('$loaded_clause'(HH,_,_) ),
    fail.

load_preds :- \+ '$loaded_clause'(_,_,_), !. % no more to load

load_preds :- load_preds. % load more...

one_template(HH):- '$loaded_clause'(HH,_,_), !.


/* Before asserting */

predicates_are_loaded(File) :-
    openToCompile(File),
    write(':- import get_residual/2, table_state/2 from tables.'), nl,
    nl, write(':- auto_table.'), nl.

openToCompile(File) :- toCompile, !,
    compileFileName(File,Name),
    tell(Name).

openToCompile(_) :- auto_table.

dynamicRules([]) :- !.

dynamicRules([N/A|L]) :-
    name(N,Str),
```

```
    ((name(T,[116,95|Str]);
    name(T,[116,117,95|Str]);
    name(T,[116,95,110,101,103,95|Str]);
    name(T,[116,117,95,110,101,103,95|Str]);
    name(T,[116,95,105,110,116,95|Str]);
    name(T,[116,117,95,105,110,116,95|Str]);
    name(T,[116,95,105,110,116,95,110,101,103,95|Str]);
    name(T,[116,117,95,105,110,116,95,110,101,103,95|Str])),
    functor(HT,T,A), insertCL((HT :- fail)), fail;
    dynamicRules(L)).

/* After asserting */

end_of_loading :- toCompile, !,
    findall(X,inBody(X),D), dynamicRules(D),
    addExecutor, told.

end_of_loading.

addExecutor :-
    insertCL((und(naf G) :- !, und(G)) ),
    insertCL((und(G) :-!, tnot(tcall(G)), tnot(tcall(naf(G) )) )),
    insertCL((tcall(G) :- call(G))),
    insertCL(( confirm(H) :- table_state(H,undef), ! )),
    insertCL(( confirm(H) :- get_residual(H,[]) )),
    insertCL(( nconfirm(H) :- table_state(H,undef), ! )),
    insertCL(( nconfirm(H) :- \+ get_residual(H,_) )).

/* Preliminary preprocessing. */

process_clause( Clause ) :- !,
    preClause( Clause, H, NewClause, Type ),
    functor(H,F,N),
    insertIfNot(hasRules(F/N)),
    insert('$loaded_clause'(H,NewClause,Type)).

preClause( (H :- true), NewH, NewH, fact ) :- !,
```

```
    processIntNegLit(H,NewH),
    declareDyn(H).

preClause( (H :- B), NewH, (NewH :- NewB), rule ) :- !,
    processIntNegLit(H,NewH),
    declareDyn(H),
    preBody(B,NewB).

preClause( H, NewH, NewH, fact ) :-
    processIntNegLit(H,NewH),
    declareDyn(H).

preBody((G,Cont),(NewG,NewCont)) :- !,
    preOne(G,NewG),
    preBody(Cont,NewCont).

preBody(G,NewG) :-
    preOne(G,NewG).

preOne(not G, not NewG) :- !,
    processIntNegLit(G,NewG),
    declareDyn(G).

preOne(G, NewG) :- !,
    processIntNegLit(G,NewG),
    declareDyn(G).

processNegLit( - H,NegH) :- !,
    H =.. [Name|Args],
    name(Name,NameH),
    name(NameNeg,[110,101,103,95|NameH]), % neg_...
    NegH =.. [NameNeg|Args].

processNegLit( H, H ).

processIntNegLit( ? H, IntNegH) :- !,
    processNegLit(H,NegH),
```

```
    NegH =.. [Name|Args],
    name(Name,NameH),
    name(NameNeg,[105,110,116,95|NameH]), % int_...
    IntNegH =.. [NameNeg|Args].

processIntNegLit( H, IntNegH ):-
    processNegLit(H,IntNegH).

/* declare dynamic predicates */

declareDyn(prolog(_)) :- !.

declareDyn(- G) :-
    !, declareDyn(G).

declareDyn(? G) :-
    !, declareDyn(G).

declareDyn(G) :-
    functor(G,N,A), insertIfNot(inBody(N/A)).

/* Transformação T1

Cada regra do tipo L :- A1,...,Am,not B1,...,not Bn, (n > 0), é
substituído por
L? :- not -L, A1,...,Am,not B1,...,not Bn*/

t_1([]).

t_1([clause(TheClause, Type)|Others]):-
    t_1One(Type,TheClause),
    t_1(Others),!.

t_1One(fact,_).

t_1One(rule,(H :- Body)):-
    (\+ isNotThereLitNegBody(Body),
```

```
    addIntIfNot(H,IntH,Conf),
    preaddNot(Conf,IntH,Body,NegBody),
    makeClause(rule,IntH,NegBody,IntClause),
    retract('$loaded_clause'(H,(H :- Body ),'rule')),
    functor(IntH,F,N),
    insertIfNot(hasRules(F/N)),
    insertIfNot('$loaded_clause'(IntH,IntClause,rule)));
    (true).

isNotThereLitNegBody((G,Cont)):- !,
    \+ isNegativeDef(G),
    isNotThereLitNegBody(Cont).

isNotThereLitNegBody(G):-
    \+ isNegativeDef(G).

preaddNot(_,IntH, Body,NegBody):-
    retInt(IntH,RetH),
    compl_neg(RetH,NegH),
    \+ isThereNotIntHead(Body,NegH),
    addIntNeg(NegH,Body,NegBody).

preaddNot(no,_,Body,Body).

makeClause(rule,IntH,NegBody,(IntH :- NegBody)).

makeClause(fact,IntH,true,IntH).

isThereNotIntHead((not G,_),IntNegH):-
    compare(=,G,IntNegH).

isThereNotIntHead(not G,IntNegH):-
    compare(=,G,IntNegH).

addIntNeg(IntNegH,Body,(not IntNegH,Body)).

/* Transformação T2
```

```
Para cada regra do programa L :- A1,...,Am,not B1,...,not Bn, se m = 1,
L? :- A1?,not B1,...,not Bn é adicionado ao programa resultante; caso
contrário,
L? :- A1,...,Am,not B1,...,not Bn é adicionado.*/

t_2([]).

t_2([clause(TheClause, Type)|Others]):-
    t_2One(Type,TheClause),
    t_2(Others),!.

t_2One(fact,F):-
    \+ isInt(F),
    addInt(F,IntF),
    functor(IntF,G,N),,
    insertIfNot(hasRules(G/N)),
    insertIfNot('$loaded_clause'(IntF,IntF,fact)),!.

t_2One(fact,_).

t_2One(rule,(H :- Body)):-
    addIntIfNot(H,IntH,Conf),
    addIntBody(Body,IntBody,Conf),
    makeClause(rule,IntH,IntBody,IntClause),
    functor(IntH,G,N),
    insertIfNot(hasRules(G/N)),
    insertIfNot('$loaded_clause'(IntH,IntClause,rule)),!.

t_2One(rule,_).

addIntBody((G,Cont),(IntG,NewBody),Conf):-
    (isNegativeDef(G),
    sameLiterals(G,IntG),!,
    addIntBody(Cont,NewBody,Conf));
    (!,getLitT2((G,Cont),(IntG,NewBody),Conf)).
```

```
addIntBody(G,IntG,Conf):-
    (isNegativeDef(G),!,
    isPlausible(Conf),
    sameLiterals(G,IntG));
    (getLitT2(G,IntG,Conf)).

sameLiterals(G,G).

getLitT2((G,Cont),(IntG,NewBody),Conf):-
    ((\+ isInt(G),
    onlyDefaults(Cont),
    addInt(G,IntG));
    (isPlausible(Conf),
    sameLiterals(G, IntG))),
    sameLiterals(Cont,NewBody),!.

getLitT2(G,IntG,Conf):-
    (\+ isInt(G),
    addInt(G,IntG));
    (isPlausible(Conf),
    sameLiterals(G, IntG)).

onlyDefaults((G,Cont)):-!,
isNegativeDef(G),
onlyDefaults(Cont).

onlyDefaults( G ):-
isNegativeDef(G).

isPlausible(no).

/* Loads one predicate */

load_predicate(HH,Clauses) :-
    assertTop(HH),
    assertClauses(Clauses).
```

```
assertTop(HH) :-
    isInt(HH),!,
    compl_int(HH,AfirH),
    assertTopNeg(AfirH).

assertTop(HH) :-
    assertTopNeg(HH).

assertTopNeg(AfirH) :-
    (isNegative(AfirH),!,
    compl_neg(AfirH,PosH),
    insertTop(PosH));
    (insertTop(AfirH)).

insertTop(HH) :-
    \+ hasTop(HH), !,
    assertz( hasTop(HH) ),
    compl_neg(HH,NH),
    compl_int(HH,IH),
    compl_int(NH,INH),
    buildT(HH,TH), buildT(NH,NTH),buildT(IH,ITH), buildT(INH,INTH),
    buildTU(HH,TUH), buildTU(NH,NTUH),
    buildTU(IH,ITUH), buildTU(INH,INTUH),
    insertCL( (HH :- TH, confirm(TH)) ),
    insertCL( (- HH :- NTH, confirm(NTH) ) ),
    insertCL( ('? '(HH) :- ITH, confirm(ITH) ) ),
    insertCL( ('? '(- HH) :- INTH, confirm(INTH) ) ),
    insertCL( (naf(HH) :- INTH, confirm(INTH)) ),
    insertCL( (naf(HH) :- tnot(TUH), nconfirm(TUH)) ),
    insertCL( (naf(? HH) :- NTH, confirm(NTH) ) ),
    insertCL( (naf(? HH) :- tnot(ITUH), nconfirm(ITUH)) ),
    insertCL( (naf(- HH) :- ITH, confirm(ITH) ) ),
    insertCL( (naf(- HH) :- tnot(NTUH), nconfirm(NTUH))),
    insertCL( (naf('? '(- HH)) :- TH, confirm(TH) ) ),
    insertCL( (naf('? '(- HH)) :- tnot(INTUH), nconfirm(INTUH))),
    !.
```

```
insertTop(_).

assertClauses([]).
assertClauses([clause(TheClause,Type)|Others]) :-
    assertOne(Type,TheClause),
    assertClauses(Others), !.

/* Loads one clause */

assertOne(fact,F) :-
    buildT(F,NewT),
    buildTU(F,NewTU),
    retAddInt(F,AddIntNegF),  buildT(AddIntNegF,TNF),
    insertCL(NewT),
    insertCL((NewTU :- tnot(TNF))).

assertOne(rule, (H :- Body) ) :-
    loadBody(Body),
    getBodies(Body,NewBodyT, NewBodyTU),
    insertClause(H,Body,NewBodyT,NewBodyTU).

loadBody((not G,Cont)):-
    assertTop( G ),
    loadBody(Cont).

loadBody((G,Cont)):-
    assertTop( G ),
    loadBody(Cont).

loadBody(not G):-
    assertTop( G ).

loadBody( G ):-
    assertTop( G ).

insertClause(H,Body,NewBodyT,NewBodyTU):-
    \+ isThereLitDefBody(Body),!,
```

```
    buildT(H,NewHT),
    buildTU(H,NewHTU), retAddInt(H,AddIntNegH), buildT(AddIntNegH,TNH),
    insertCL( (NewHT  :- NewBodyT ) ),
    insertCL( (NewHTU :- tnot(TNH), NewBodyTU ) ).

insertClause(H,_,NewBodyT,NewBodyTU):-
    buildT(H,NewHT),
    buildTU(H,NewHTU),
    insertCL( (NewHT  :- NewBodyT ) ),
    insertCL( (NewHTU :- NewBodyTU ) ).

retAddInt(H,AddIntNegH):-
    (isInt(H),
    retInt(H,AfirH),
    compl_neg(AfirH,AddIntNegH));
    (compl_neg(H,AddNegH),
    addInt(AddNegH,AddIntNegH)).

isThereLitDefBody((G,_)):-
    isNegativeDef(G),!.

isThereLitDefBody(G):-
    isNegativeDef(G).

/* Processes elements in the body */

getBodies((A,B), (NewAT,NewBT), (NewATU,NewBTU) ) :- !,
    getBodiesOne(A, NewAT,NewATU),
    getBodies(B, NewBT, NewBTU).

getBodies(A, NewAT, NewATU ) :-
    getBodiesOne(A, NewAT,NewATU).


/* Processes one elements of the body */

getBodiesOne(prolog(G),G,G) :- !.
```

```
getBodiesOne(not A, NewDisjT, NewDisjTU) :- !,
    buildT(A,AT),
    buildTU(A,ATU),
    getNewNAF(ATU,AT,NewDisjT, NewDisjTU).

getBodiesOne(A, NewAT, NewPosU ) :-
    buildT(A,NewAT),
    buildTU(A,NewPosU).

getNewNAF(ATU,AT,(tnot(ATU)),(tnot(AT))).

/* Auxiliar predicates  */

compl_neg(A,NegA) :-
    A =.. [Name|Args],
    name(Name,[110,101,103,95|Pos]), !, % neg_...
    name(NamePos,Pos),
    NegA =.. [NamePos|Args].

compl_neg(A,NegA) :-
    A =.. [Name|Args],
    name(Name,NameH),
    name(NameNeg,[110,101,103,95|NameH]), % neg_...
    NegA =.. [NameNeg|Args].

compl_int(A,AfirA) :-
    A =.. [Name|Args],
    name(Name,[105,110,116,95|Afir]), !, % int_...
    name(NameAfir,Afir),
    AfirA =.. [NameAfir|Args].

compl_int(A,IntA) :-
    A =.. [Name|Args],
    name(Name,NameH),
    name(NameInt,[105,110,116,95|NameH]), % int_...
    IntA =.. [NameInt|Args].
```

```
buildT(G,NewG) :-
    G =.. [Name|Args],
    name(Name,Str),
    name(NN,[116,95|Str]),% t_ ...
    NewG =.. [NN|Args].

buildTU(G,NewG) :-
    G =.. [Name|Args],
    name(Name,Str),
    name(NN,[116,117,95|Str]),% tu_ ...
    NewG =.. [NN|Args].

addInt(H,IntH):-
    H =.. [Name|Args],
    name(Name,NameH),
    name(NameInt,[105,110,116,95|NameH]), % int_...
    IntH =.. [NameInt|Args].

addIntIfNot(H,H,yes):-
    isInt(H),!.

addIntIfNot(H,IntH,no):-
    addInt(H,IntH).

retInt(IntH,RetH):-
    IntH =.. [Name|Args],
    name(Name,[105,110,116,95|Afir]), !, % int_...
    name(NameAfir,Afir),
    RetH =.. [NameAfir|Args].

isNegative( G ) :-
    G =.. [Name|_],
    name(Name,[110,101,103,95|_]), !. % neg_...

isNegativeDef( G ) :-
    G =.. [Name|_],
```

```
    name(Name,[110,111,116|_]), !. % not ...

isInt( G ) :-
    G =.. [Name|_],
    name(Name,[105,110,116,95|_]), !. % int_...

append([],L,L).

append([H|T],L,[H|New]) :-
    append(T,L,New).

insertIfNot(A) :- A, !.

insertIfNot(A) :- insert(A).

insert( Clause ) :-
    assertz(Clause).

insertCL( Clause ) :-
    toCompile, !,
    nl,
    write(Clause), write('.').

insertCL( Clause ) :- insert(Clause).

apagaTudo(G) :-
    clause(G,_),
    retract(G), fail.

apagaTudo(_).
```