UMA ANÁLISE DOS OPERADORES DE REVISÃO DE TEORIAS DO SISTEMA DE REVISÃO FORTE_MBC

Ana Luísa de Cerqueira Leite Duboc

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Gerson Zaverucha

Rio de Janeiro
Junho de 2014

UMA ANÁLISE DOS OPERADORES DE REVISÃO DE TEORIAS DO SISTEMA DE REVISÃO FORTE_MBC

Ana Luísa de Cerqueira Leite Duboc

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Gerson Zaverucha, Ph.D.

Prof. Mário Roberto Folhadela Benevides, Ph.D.

Prof[a]. Aline Marins Paes Carvalho, D.Sc.

Prof[a]. Sheila Regina Murgel Veloso, D.Sc.

Prof. Luís da Cunha Lamb, Ph.D.

Prof. Marcelo Finger, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2014

Duboc, Ana Luísa de Cerqueira Leite

Uma Análise dos Operadores de Revisão de Teorias do Sistema de Revisão FORTE_MBC/Ana Luísa de Cerqueira Leite Duboc. – Rio de Janeiro: UFRJ/COPPE, 2014.

X, 110 p.: il.; 29, 7cm.

Orientador: Gerson Zaverucha

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 106 – 110.

1. Operadores de Refinamento. 2. Revisão de Teoria. 3. Cláusula mais específica. 4. ILP. 5. Aprendizado de Máquina. 6. Lógica de Primeira Ordem. I. Zaverucha, Gerson. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Toda história tem seu fim, mas
na vida cada final é um novo
começo

Agradecimentos

Aos meus pais, Maria Lúcia e Gilson, à minha vó Maria Sylvia, e à minha irmã Letícia, pelo apoio, amor e suporte dados para que eu chegasse até aqui.

Ao meu orientador, Professor Doutor Gerson Zaverucha, pelos conhecimentos transmitidos, por me guiar na elaboração deste trabalho e por acreditar em mim.

A minha amiga, Aline Paes, pelo apoio e carinho de sempre.

Ao meu namorado, Caio, por todo apoio que me deu, principalmente na finalização da Tese.

A CAPES pelo suporte financeiro.

A todos que de alguma forma contribuíram para a escrita de mais um capítulo da minha história.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA ANÁLISE DOS OPERADORES DE REVISÃO DE TEORIAS DO SISTEMA DE REVISÃO FORTE_MBC

Ana Luísa de Cerqueira Leite Duboc

Junho/2014

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

FORTE_MBC é um Sistema de Revisão de Teorias de Primeira-Ordem, construído a partir do FORTE mas com um importante complemento: ele usa a cláusula mais específica (*Bottom Clause*) e declaração de modos para definir o espaço de busca de literais. A introdução da *Bottom Clause* foi essencial para a redução do tempo de execução do processo de revisão (em média 55 vezes mais rápido), já que revisar teorias normalmente considera espaços de busca extensos, visto que refina teorias inteiras, ao invés de cláusulas individuais, uma por vez. Porém, a eficácia e a eficiência do processo de refinamento depende fortemente de outros fatores, como da ordem de generalidade que induz um modelo de generalização para o espaço de busca e, consequentemente, dos operadores de refinamento. Estes fatores nunca foram, até então, formalmente analisados no FORTE(_MBC). Nesta Tese contribuímos com (1) uma modificação dos modelos teóricos existentes para caracterizar operadores de refinamento, de forma a caracterizar o espaço de busca e os operadores de refinamento do FORTE_MBC; (2) uma melhoria nos operadores de refinamento de teorias do FORTE_MBC para torná-los ideais para o espaço definido. Por fim, a eficiência da revisão também é influenciada pelo total de átomos básicos vindos do Conhecimento Preliminar (BK). A *Bottom Clause* gerada no FORTE_MBC inclui todos os literais gerados a partir do BK que são relevantes para um exemplo dados certos parâmetros, podendo gerar espaços exponenciais. Porém, muitos destes literais não obedecem aos modos quando adicionados à cláusula sob revisão. Para amenizar este problema, propomos o sistema FORTE_BETH, que usa o algoritmo do sistema BETH para gerar uma *Bottom Clause* linear que contém apenas os literais que poderiam ser adicionados a uma cláusula de acordo com os modos definidos, o que tem o potencial de tornar o processo de revisão ainda mais eficiente.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

AN ANALYSIS OF FORTE_MBC THEORY REVISION OPERATORS

Ana Luísa de Cerqueira Leite Duboc

June/2014

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

FORTE_MBC is a First-Order Logic Theory Revision System, built upon FORTE but with an important supplement: it makes use of the Bottom Clause and modes declaration to define the search space of literals. Introducing the Bottom Clause was essential to reduce the runtime of the revision process (an average speed-up of 55), since revising theories usually considers large search spaces, mainly because it refines whole theories instead of stepwise individual clauses. However, the effectiveness and efficiency of the refinement process heavily relies on other factors, such as the generality relation that induces a generalization model for the search space, and, as a consequence, the refinement operators. These factors have never been formally analyzed in FORTE(_MBC) yet. In this work, we contribute with (1) a modification of the existing theoretical frameworks for characterizing refinement operators in order to characterize the search space and refinement operators of FORTE_MBC; (2) an improvement of the theory refinement operators of FORTE_MBC to make them ideal for the defined space. Finally, the efficiency of the review process is also influenced by the number of ground atoms from Background Knowledge (BK). The bottom clause generated in FORTE_MBC includes all literals created from BK that are relevant for some example, given some parameters, which can generate exponential spaces. However, many of those literals do not obey to the modes when are added to the clause under revision. To alleviate this problem, we propose the FORTE_BETH system, which use the BETH algorithm to generate a bottom clause of linear complexity that contains only the literals that could be included in a clause according to the modes defined, what can lead to a more efficient review process.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A lógica de primeira-ordem é um sistema robusto de representação de conhecimento, pois consegue representar situações complexas envolvendo vários objetos e relações entre eles. Dentro da área de Programação em Lógica Indutiva (ILP) [16, 22, 23, 25, 26], definida como a interseção entre o aprendizado indutivo e a programação lógica, algoritmos de aprendizado têm sido desenvolvidos para aprender teorias em lógica de primeira-ordem. A partir de uma teoria inicial assumida como correta e que portanto não pode ser modificada, o que chamamos de conhecimento preliminar, e de um conjunto de exemplos positivos e negativos, sistemas ILP acham um conjunto de novas cláusulas que impliquem logicamente os exemplos positivos, não impliquem os exemplos negativos e tenham boa capacidade de fazer generalizações (lidam corretamente com os exemplos de teste).

Enquanto o aprendizado indutivo lida com aprender teorias a partir do zero, já que não considera teorias iniciais dadas que fazem parte do conhecimento preliminar, a área de ILP também engloba a tarefa de revisar teorias [47], onde, além do conhecimento preliminar, partimos de uma teoria inicial que pode esta incorreta ou imcompleta, e portanto pode ser modificada. Portanto o objetivo é achar uma teoria revisada que passe a implicar logicamente os exemplos positivos e não implique logicamente os exemplos negativos. A única diferença em relação aos sistemas ILP é que uma teoria inicial incorreta e modificável é dada. Os sistemas de revisão de teorias tem como objetivo gerar teorias revisadas mais acuradas do que a inicial.

Sistemas de revisão de teorias [1, 10, 36, 37, 40, 46, 48], partem da teoria inicial fornecida e a modifica nos pontos onde é verificada uma classificação incorreta dos exemplos. A melhor modificação entre todas as revisões propostas é escolhida. As modificações são feitas utilizando-se operadores de revisão, que se dividem em operadores de especialização e generalização. Pontos na teoria onde a prova de exemplos positivos falha são lugares onde a teoria precisa ser generalizada, e cláusulas usadas em provas de exemplos negativos são pontos onde a teoria precisa ser especializada.

Um conhecido sistema de revisão de teorias é o FORTE [37], que automatica-

mente revisa teorias Horn de primeira-ordem livres de função. FORTE executa uma busca *Hill Climbing* através do espaço de operadores de especialização e generalização, numa tentativa de achar uma revisão mínima que torne a teoria consistente com o conjunto de exemplos de treinamento e que tenha uma boa capacidade de generalização. Ao contrário da maioria dos algoritmos de aprendizado relacional, tais como os implementados nos sistemas Foil [34] e Progol [24], que adotam estratégias de cobertura iterativas baseadas em refinamento de cláusulas, sistemas de revisão de teorias lidam com o problema de refinar teorias inteiras (como nos sistemas MPL [8] e HYPER [5]) ao invés de cláusulas individuais, uma de cada vez, o que é considerado um problema difícil [5]. Apesar disso, refinar teorias é geralmente baseado em operadores que refinam cláusulas. Por exemplo, uma das possíveis modificações que o FORTE pode fazer em uma teoria é adicionar novas regras, através do operador de teorias de Adição de Regras. Este operador realiza modificações no nível de teoria, mas para isso ele refina cláusulas individuais, através do operador de cláusulas de exclusão de Antecedentes seguido da Adição de Antecedentes.

Considerar um espaço de teorias tem a vantagem de produzir hipóteses finais mais compreensíveis e resultados mais acurados, já que a teoria é avaliada como um todo, e também de permitir que se lide com aprendizado de múltiplos predicados simultaneamente. Porém, refinar teorias acarreta em espaços de busca maiores do que se refinar cláusulas individuais, o que pode consumir muito tempo e espaço de armazenamento. Podemos esperar também que a busca seja mais custosa se a teoria tiver muitas falhas. Para lidar com estes problemas foram desenvolvidos os sistemas YAV_FORTE [33] e FORTE_MBC [9, 10], que surgiram a partir do FORTE e usam a cláusula mais específica, também chamada de *Bottom Clause*, como espaço de busca de literais a serem adicionados a uma cláusula. Estes sistemas também usam um conjunto de declarações de modos para validar as cláusulas geradas pela revisão, sendo que os modos também são usados na construção da *Bottom Clause*. As declarações de modos são fornecidas pelo especialista do domínio e descrevem as relações (predicados) entre objetos de dados e tipos desses dados. Estas declarações permitem também informar se a relação pode ser utilizada na cabeça ou no corpo das cláusulas geradas. Na declaração de modos também são descritos o tipo dos argumentos e o número máximo de instanciações de cada predicado.

Ao introduzir a *Bottom Clause* [24] e o conjunto de declaração de modos, o FORTE_MBC reduziu consideravelmente o espaço de busca de cláusulas, que por sua vez reduziu o espaço de busca de teorias, o que se mostrou essencial para reduzir o tempo de execução do processo de revisão em uma média de 55 vezes [10]. Além disso, ele gera teorias mais compreensíveis e não diminui significativamente a acurácia obtida pelo processo de revisão original, mantendo as acurácias mais elevadas quando comparado ao aprendizado do zero, assim como o sistema FORTE original.

No entanto, a eficácia e a eficiência do aprendizado como um processo de refinamento depende fortemente de outros fatores, tais como as propriedades do espaço de busca e, como consequência, dos operadores de refinamento [13]. Tradicionalmente, em ILP, os sistemas são definidos através de uma caracterização formal de seu espaço de busca e dos seus operadores de refinamento, assim algoritmos e operadores mais eficientes podem ser projetados [4, 11, 13, 20, 42]. O FORTE_MBC já melhorou a eficiência do FORTE, mas seu espaço de busca e operadores de refinamento não foram devidamente caracterizados.

Operadores de refinamento determinam as características dos passos de refinamento executados através do espaço de busca. A definição de um operador é afetada pela escolha da relação de generalidade, usualmente implicação lógica ou θ-*subsumption*, que induz um modelo de generalização para este espaço. Idealmente, operadores de refinamento devem ser ideais ou ótimos. Porém, não é possível ter tais tipos de operadores baseados em θ-*subsumption* [45], e portanto muitos trabalhos propõem modelos mais restritos de operadores de refinamento baseados em θ-*subsumption* [4, 11, 13, 20, 31, 35, 42].

Buscar num espaço de hipóteses limitado por uma cláusula mais específica é a base de sistemas conhecidos de ILP, como Progol [24] e Aleph [41]. Em [42], tendo o sistema Progol como instância, a estrutura e as propriedades de tal espaço de hipóteses foram corretamente caracterizados pela introdução da *subsumption order* relativa a uma *Bottom Clause*. Além disso também foi mostrada a existência de operadores de refinamento ideais para esta ordem de generalidade. Porém, tal ordem de generalidade não caracteriza propriamente o espaço de busca do FORTE_MBC, pois o Progol considera que os literais das cláusulas criadas são gerados a partir de substituições θ nos literais da *Bottom Clause*, o que não acontece no FORTE_MBC, onde o espaço de cláusulas é proposicional. Além disso, o Progol se restringe a um espaço de cláusulas, e lida apenas com aprendizado do zero, executando uma busca *top-down*, de forma que apenas operadores de refinamento de especialização de cláusulas são considerados.

Por outro lado, Midelfart [20] introduz uma extensão de *subsumption* em teorias (conceito este introsuzido por Arimura [2]) que leva em consideração uma teoria mais específica, chamada de Teoria *Bottom*, que é uma teoria formada por *Bottom Clauses*. Ao introduzir esta teoria, Midelfart [20] propôs um espaço de busca mais restrito, que também é ordenado pela θ-*subsumption*, mas como uma linguagem mais restrita, ou seja, a linguagem de teorias subsumindo uma determinada Teoria *Bottom*. Porém, os resultados teóricos apresentados em [20] não se aplicam ao espaço de teorias do FORTE_MBC, visto que este não é limitado por uma Teoria *Bottom* e, como em [42], também só consideram um espaço de busca de especialização.

Como primeira contribuição, nesta Tese nós adaptamos o modelo teórico apre-

sentado em [42] para caracterizar o espaço de busca e operadores de refinamento de cláusulas do FORTE_MBC. Em seguida nós estendemos este modelo para um espaço de teorias, de forma a caracterizar o espaço de busca e operadores de refinamento de teorias do FORTE_MBC. Em particular, estamos procurando por operadores de refinamento completos e de preferência ideais, já que quando a busca começa de um ponto qualquer do espaço de busca, o que é o caso do FORTE_-MBC, é melhor que os operadores sejam completos, mesmo que isto acarrete em redundância [42].

Os resultados por nós apresentados são importantes para a melhor compreensão deste espaço restrito de teorias de sistemas de revisão de teorias de primeira ordem, que usam uma teoria inicial, um conjunto de *Bottom Clauses* e declaração de modos para restringir o espaço de busca, como é o caso do FORTE_MBC. Tais resultados podem ser aplicados a qualquer sistema de revisão de teorias de primeira ordem que possuam tais características. Adicionalmente, caracterizar tal espaço de refinamento é essencial para que se possa propor operadores de refinamento ideais para o mesmo. No FORTE_MBC, por exemplo, nós mostramos que os operadores de refinamento de teorias não são ideais para o espaço de busca definido, já que eles não são próprios, e já que o operador de generalização de teorias não é completo. Então, como outra contribuição, propomos operadores de refinamento ideais para o FORTE_MBC.

Ao analisar a *Bottom Clause* como espaço de busca de literais e limitante do reticulado de cláusulas no FORTE_MBC, temos que, apesar da *Bottom Clause* reduzir o espaço de busca, quando comparado a um espaço ordenado por θ-*subsumption*, por exemplo, ela também pode resultar num espaço de busca exponencial, dependendo do conjunto de átomos básicos usados para descrever os exemplos e dos parâmetros considerados na sua construção.

Em problemas aonde os exemplos são muito grandes, o conhecimento preliminar (BK) pode conter muitos átomo básicos ou gerar muitos átomos básicos a partir de suas regras. Quando uma *Bottom Clause* é criada nestes sistemas, ela inclui todos os literais gerados a partir dos átomos básicos relevantes para um determinado exemplo (de acordo com os parâmetros impostos, tais como declarações de modos e limite de profundidade de variável), o que pode levar a *Bottom Clauses* muito grandes que geram espaços exponenciais quando aprendendo ou revisando uma cláusula. Porém, nem todos estes literais serão de fato avaliados como possíveis candidatos a serem adicionados na cláusula sendo revisada, devido à restrição de modos desta cláusula.

Para tratar este problema, em [44] foi proposta uma abordagem chamada BETH que gera uma *Bottom Clause* que contém apenas os literais relevantes para o processo de construção de uma cláusula. A complexidade da *Bottom Clause* deixou de ser exponencial e passou a ser linear com respeito ao tamanho máximo permitido para uma cláusula.

O BETH tira vantagem dos pontos fortes destas duas abordagens, combinando-as em uma só. Nesta nova abordagem, a *bottom clause* não mais é construída, a partir de um exemplo positivo, antes de se começar a busca por uma boa cláusula. Ao invés disso, depois que o exemplo positivo é escolhido, os literais são gerados de forma *top-down*, mas ainda a partir de um exemplo, ou seja, guiados pela busca heurística, com a diferença de que os literais gerados são restritos àqueles que cobrem o exemplo positivo escolhido.

Motivados por este trabalho, finalizamos esta Tese propondo uma alteração no algoritmo de adição de antecedentes do FORTE_MBC, através da implementação da abordagem proposta pelo BETH, de forma que a *bottom clause* não seja mais construída a priori, mas sim descoberta durante a especialização de uma cláusula. Pretendemos com isso tornar o processo de revisão do FORTE_MBC ainda mais eficiente, visto que a complexidade do tamanho da *bottom clause* deixará de ser exponencial e passará a ser linear, resultando numa busca mais eficiente no espaço limitado por esta. A introdução do novo algoritmo dá origem ao sistema FORTE_-BETH.

A presente Tese está organizada da seguinte forma: no Capítulo 2 serão descritos conceitos relacionados à lógica de primeira-ordem, Programação em Lógica Indutiva, construção da *Bottom Clause* e operadores de refinamento, que são a base para o entendimento desta Tese. No Capítulo 3.1 discutiremos revisão de teorias de primeira-ordem e o refinamento de teorias no sistema FORTE_MBC, introduzindo seus operadores de refinamento e que tipo de teorias o sistema é capaz de gerar. No Capítulo 4 formalmente caracterizamos o espaço de busca e operadores de refinamento do FORTE_MBC, propondo operadores de refinamento ideais para este. Ainda neste capítulo, discutimos alguns trabalhos relacionados citados anteriormente, mostrando porque o espaço de busca do FORTE_MBC não pode ser caracterizado pelas ordens de generalidade introduzidas por estes trabalhos. No Capítulo 5 introduzimos o algoritmo FORTE_BETH. Finalmente no Capítulo 6 apresentamos nossas conclusões e trabalhos futuros.

Capítulo 2

Conceitos Fundamentais

Neste capítulo apresentaremos os conceitos nos quais baseamos nossa pesquisa e que serão importantes para o entendimento do restante da Tese. Os leitores já familiarizados com algum desses assuntos podem encaminhar-se para as outras seções.

Este trabalho compreende temas como sistemas de representação em lógica de primeira-ordem, programação em lógica indutiva (ILP), aprendizado e revisão de teorias utilizando-se a cláusula mais específica, conhecida como *Bottom Clause*, sistemas de revisão de teorias, mais especificamente o sistema FORTE_MBC [9] e definição e análise de operadores de refinamento em ILP. Em um primeiro momento serão apresentados conceitos relacionados a lógica. Na Seção 2.1 é definida a terminologia utilizada em lógica de primeira-ordem [6, 18, 49] e na Seção 2.2 definimos Programação em Lógica Indutiva (ILP) [16, 22, 23, 25, 26]. A seguir, na Seção 2.3 descrevemos como se dá a construção da cláusula mais específica introduzida pelos sistemas Progol [24] e Aleph[19, 41]. Por fim, na Seção 2.4 conceitos relacionados a operadores de refinamentos, ordens de generalidade e reticulados são revistos. No próximo capítulo descrevemos revisão de teorias de primeira-ordem [47], especificamente o sistema de Revisão FORTE_MBC analisado nesta Tese.

2.1 Lógica de primeira-ordem

A lógica de primeira-ordem, conhecida também como cálculo de predicados, é um sistema lógico que estende a lógica proposicional. A lógica de primeira-ordem é capaz de representar relações entre objetos, concluir particularizações de uma propriedade geral dos indivíduos de um universo de discurso, assim como derivar generalizações a partir de fatos que valem para um indivíduo arbitrário do universo de discurso. Para ter tal poder de expressão, a linguagem de primeira-ordem usa símbolos mais sofisticados do que os da linguagem proposicional. Para expressar propriedades que valem para todos os indivíduos ou apenas para alguns, são utilizados os quantificadores $\forall$(universal) e $\exists$(existencial), respectivamente. Daremos a seguir alguns conceitos e

terminologias referentes à lógica de primeira-ordem.

Um *alfabeto de primeira-ordem* consiste de:

1. Símbolos lógicos:
   - Pontuação: "(",",",",")"
   - Conectivos: $\neg$(negação),$\wedge$(conjunção), $\vee$(disjunção), $\leftarrow$(implicação) e $\leftrightarrow$(bi-condicional)
   - Quantificadores: $\forall$(universal) e $\exists$(existencial)
   - Variáveis, que se iniciam com letras maiúsculas. Exemplos: A, Var
   - Símbolo de igualdade (opcional): $=$
2. Símbolos não-lógicos: funções (que se iniciam por letras minúsculas), constantes (símbolos funcionais de aridade 0) e predicados.

O conjunto de *termos de primeira-ordem* é o conjunto formado pelas variáveis, constantes, e funções aplicadas a termos, como por exemplo $f(t_1, ..., t_n)$, onde $t_1, ..., t_n$ são termos. Se $t_1, ..., t_n$ são termos e P é um símbolo predicativo n-ário, então $P(t_1, ..., t_n)$ é chamado de fórmula atômica ou átomo. Exemplo: *tio(joão, X)*. Um átomo é um literal positivo, e a negação de um átomo é um literal negativo.

Se A e B são fórmulas, então ($\neg$A), (A$\wedge$B), (A$\vee$B), (A$\leftarrow$B) e (A$\leftrightarrow$B) são fórmulas. Se B é uma fórmula e X é uma variável, então $\forall_X$(B) e $\exists_X$(B) também são fórmulas.

Uma *cláusula* é uma disjunção de literais precedida por um prefixo de quantificadores universais, um para cada variável que aparece na disjunção, na seguinte forma:

$$\forall X_1 \forall X_2 ... \forall X_s (L_1 \vee L_2 \vee ... \vee L_m)$$

onde cada L_i é um literal e $X_1, X_2, ..., X_s$ são todas as variáveis ocorrendo em $(L_1 \vee L_2 \vee ... \vee L_m)$. O conjunto $\{A_1, A_2, ..., A_h, \neg B_1, \neg B_2, ..., \neg B_b\}$, onde A_i e B_i são átomos, indica a cláusula:

$$\forall A_1 \forall A_2 ... \forall A_h \forall B_1 \forall B_2 ... \forall B_b (A_1 \vee ... A_h \vee \neg B_1 \vee ... \vee \neg B_b)$$

que é equivalentemente representado por $\{A_1 \vee ... \vee A_h \leftarrow B_1 \wedge ... \wedge B_b\}$, e pode ser escrita como $A_1, ..., A_h \leftarrow B_1, ..., B_b$, onde $A_1, ..., A_h$ é a cabeça e $B_1, ..., B_b$ é o corpo da cláusula. Uma *teoria* é um conjunto de cláusulas, representando a conjunção dessas cláusulas.

Visto que uma cláusula pode ser representada como um conjunto de literais, e uma teoria como um conjunto de cláusulas, temos que:

- Dadas duas cláusulas C e D, e um literal l:
  - $C \cup \{l\}$ representa a inclusão do literal l na cláusula C, tal que l se torna o literal mais à direita de C.
  - $C \setminus \{l\}$ ou $C - \{l\}$ representam a exclusão do literal l da cláusula C.
  - $C \cap D$ representa a interseção das cláusulas C e D, ou seja, os literais em comum entre as duas cláusulas.
  - $C \cup D$ representa a adição dos literais da cláusula D à cláusula C, tal que os literais de D são colocados mais à direita em C.
  - $C \setminus D$ ou $C - D$ representam a diferença entre as cláusulas C e D, tal que os literais de D são excluídos de C.
- Dadas duas teorias T_1 e T_2, e uma cláusula C:
  - $T_1 \cup \{C\}$ representa a adição da cláusula C à teoria T_1.
  - $T_1 \setminus \{C\}$ ou $T_1 - \{C\}$ representa a exclusão da cláusula C da teoria T_1.
  - $T_1 \cap T_2$ representa a interseção das teorias T_1 e T_2, ou seja, as cláusulas em comum entre as duas teorias.
  - $T_1 \cup T_2$ representa a união das teorias T_1 e T_2.
  - $T_1 \setminus T_2$ ou $T_1 - T_2$ representam a diferença entre as teorias T_1 e T_2, tal que as cláusulas de T_2 são excluídas de T_1.

Uma *cláusula de Horn* é uma cláusula que contém no máximo um literal positivo, ou seja, um literal na cabeça. Cláusulas com exatamente um literal na cabeça são chamadas de *cláusulas definidas*. Um *fato* é uma cláusula definida com o corpo vazio, representado por $A \leftarrow$ e denotado por A. Um conjunto de cláusulas definidas é chamado de *programa lógico definido*. A figura 2.1 mostra um exemplo de um programa lógico definido.

$progenitor(bob, frank).$ $progenitor(cindy, frank).$
$progenitor(alice, john).$ $progenitor(tom, john).$
$irmao(tom, cindy).$ $irma(cindy, tom).$
$marido(tom, alice).$ $marido(bob, cindy).$

$tio(X, Y) \leftarrow irmao(X, Z), progenitor(Z, Y).$
$tio(X, Y) \leftarrow marido(X, Z), irma(Z, W), progenitor(W, Y).$

Figura 2.1: Exemplo de um programa lógico definido [21]

Uma *substituição* $\theta = \{V_1/t1, ..., V_n/t_n\}$ é uma associação de termos t_i para variáveis V_i. Aplicar uma substituição θ para um termo, átomo ou cláusula F

produz o termo, átomo ou cláusula instanciado $F\theta$, onde todas as ocorrências de variáveis V_i são simultaneamente substituídas pelo termo t_i. Um termo, átomo ou cláusula é básica quando não existe nenhuma variável ocorrendo nele. Uma cláusula é livre de funções se ela não contém símbolos funcionais. Seja $\sum = \{E_1, ..., E_n\}$ um conjunto de fórmulas e β uma substituição. β é um unificador de $\sum$ se, e somente se, $E_1\beta = ... = E_n\beta$.

Uma cláusula C θ-subsume ($\succeq$) a cláusula D se existe uma substituição θ tal que $C\theta \subseteq D$. Por exemplo, a cláusula $C : f(A, B) \leftarrow p(B, G), q(G, A)$ θ-subsume a cláusula $D : f(a, b) \leftarrow p(b, g), q(g, a), t(a, d)$ através da substituição $\{A/a, B/b, G/g\}$. Portanto, $C \succeq D$.

Interpretação (ou atribuição de valor-verdade) é um processo que mapeia uma sentença em alguma declaração em relação a um determinado domínio. Se a interpretação der o valor verdadeiro para uma sentença, então ela satisfaz esta sentença e tal interpretação é chamada um *modelo* da sentença. A noção de modelo é aplicada extensionalmente para uma teoria clausal: uma interpretação é um modelo para um conjunto se ela é um modelo para cada um dos membros do conjunto. Uma teoria H implica logicamente uma cláusula C ($H \vDash C$), ou seja, C é *consequência lógica* de H, se e somente se todo modelo de H também for modelo de C.

Um *sistema dedutivo SD = (L, AX, R)* é um sistema formado por uma linguagem L, um conjunto de axiomas lógicos AX e um conjunto de regras de inferência R. Uma fórmula C é um teorema (ou C tem uma *dedução* ou *prova* a partir) de um conjunto de fórmulas H, no sistema dedutivo SD (denotado por $H \vdash C$), se e somente se, existir uma sequência finita de fórmulas $D_1, ..., D_n$ tal que:

- $D_n = C$
- Para todo $i \in [1, n]$, uma das condições abaixo é satisfeita:
  - D_i é uma instância de um axioma lógico de AX
  - $D_i \in H$
  - D_i é conclusão de uma das regras de inferência de R onde as premissas já estão na sequência.

2.2 Programação em Lógica Indutiva

Tendo sido definida a terminologia utilizada em lógica de primeira-ordem e em programação lógica, podemos definir brevemente o que vem a ser Programação em Lógica Indutiva (ILP − *Inductive Logic Programming*). Para mais informações sobre o assunto veja [16, 22, 23, 25, 26].

Programação em Lógica Indutiva (ILP) é uma área da inteligência artificial que investiga a construção indutiva de teorias de cláusulas de Horn de primeira-ordem a partir de exemplos e de um conhecimento preliminar. ILP é definido como sendo a interseção entre aprendizado indutivo e programação lógica. Do aprendizado indutivo, ILP herda o seu objetivo: construção de ferramentas e técnicas para induzir hipóteses a partir de observações (exemplos) e sintetizar novo conhecimento a partir da experiência. Da Programação em Lógica [18], ILP herda o formalismo representacional, como a representação de teorias, hipóteses, exemplos e conhecimento preliminar, técnicas bem estabelecidas e uma profunda base teórica.

ILP pode ser vista como o estudo de métodos de aprendizado para dados e regras que são representados como predicados lógicos de primeira-ordem. Predicados lógicos permitem variáveis quantificadas e relações, e podem representar conceitos que não podem ser expressos usando uma linguagem de descrição proposicional. Uma base de dados relacional pode ser facilmente traduzida em lógica de primeira-ordem e ser usada como fonte de dados para ILP. Como exemplo, considere as seguintes regras escritas na sintaxe Prolog, as quais definem a relação tio/2:

$tio(X, Y) \leftarrow irmao(X, Z), progenitor(Z, Y).$
$tio(X, Y) \leftarrow marido(X, Z), irma(Z, W), progenitor(W, Y).$

O objetivo de ILP é inferir regras, como as apresentadas acima, dada uma base de dados com fatos e definições lógicas de outras relações. Por exemplo, um sistema de ILP pode aprender as regras para tio/2, chamado de predicado alvo, dado um conjunto de exemplos positivos e negativos das relações tio/2 e um conjunto de fatos para as relações progenitor/2, irmao/2, irma/2 e marido/2, chamadas de predicados de conhecimento preliminar para os membros de uma família [21].

A hipótese fundamental do aprendizado indutivo é que qualquer hipótese descoberta que aproxima bem uma determinada função alvo usando um conjunto suficientemente grande de exemplos de treinamento, também aproximaria bem a função alvo sobre outros exemplos não observados (generalização). A partir dos exemplos de treinamento, o sistema de ILP induz um programa lógico correspondente à visão que define a relação alvo, em termos de outras relações que são dadas como conhecimento preliminar. Em sistemas ILP, os exemplos de treinamento, o conhecimento preliminar e as hipóteses induzidas são todos expressos na forma de programas lógicos, sendo que os exemplos de treinamento são representados como fatos da relação alvo a ser aprendida. O conjunto de conhecimento preliminar é definido como extensional se ele é representado como um conjunto de literais básicos, tais como $pai(pedro, ana)$, ou definido como intensional, se é representado como uma teoria de cláusulas de Horn. Por exemplo, $avo(X, Y) \leftarrow pai(X, Z), pai(Z, Y)$ e um conjunto

de fatos básicos definindo a relação *pai* é considerado um conhecimento preliminar intensional. Podemos definir a tarefa básica de ILP da seguinte forma [16]:

Definição 1. *Dados*:

- Um conhecimento preliminar invariante BK
- Um conjunto de exemplos positivos E^+ e negativos E^-

Achar:

- Uma teoria H que, junto com o conhecimento preliminar BK, implique logicamente todos os exemplos positivos (completo), $BK \cup H \models E^+$ e nenhum dos exemplos negativos (consistência), $\forall e^- \in E^- : BK \cup H \not\models e^-$, e que esteja de acordo com o critério de qualidade, tal como o de minimalidade do tamanho da teoria.

Para formalizar esta definição vamos definir o que vem a ser a relação de cobertura em ILP: dados BK, H e E definidos como anteriormente, a hipótese H cobre um exemplo $e \in E$ em relação a BK se $BK \wedge H \models e$.

As definições acima requerem que a relação alvo c e a teoria H concordem em todos os exemplos de E. Porém não há garantias de que H irá corresponder a c em exemplos não vistos. A acurácia de classificar exemplos não vistos é o principal critério de sucesso do sistema de aprendizado.

Definição 2. Dada uma teoria de primeira-ordem H, a acurácia de classificação de H é medida como sendo a porcentagem de exemplos corretamente classificados pela teoria.

A transparência de H denota o quanto ela é compreensível por humanos. Uma medida utilizada é o tamanho da hipótese, expressa pelo total de condições no conjunto de regras que formam H.

Sistemas ILP podem aprender um único conceito ou múltiplos conceitos (predicados). O aprendiz pode tentar aprender um conceito do zero ou pode aceitar uma teoria inicial que pode ser revisada no processo de aprendizado.

Um sistema ILP como o FOIL [34] e o Progol [24] acham cláusulas iterativamente adicionando uma cláusula de cada vez à teoria resultante. Uma característica comum destas cláusulas é que elas cobrem, pelo menos, um exemplo não coberto anteriormente, dado o conhecimento preliminar e as cláusulas já descobertas. Não é necessário que uma cláusula cubra exemplos diretamente. A cláusula pode implicar uma instância básica do predicado alvo, que então implica um exemplo através de outra cláusula (recursiva), dado que essa cláusula já foi achada. Neste caso, a cláusula cobre exemplos indiretamente [20]. A partir disso podemos definir o conceito de teoria relevante:

Definição 3 (Teoria relevante)**.** [20] Uma teoria é relevante se todas as cláusulas desta teoria cobrem algum exemplo diretamente.

2.3 Progol/Aleph e a construção da *Bottom Clause*

Em ILP, a busca pela hipótese exige uma heurística de busca no espaço de hipóteses. Geralmente existem duas abordagens: *top-down* e *bottom-up*. As duas abordagens podem ser vistas como um tipo de algoritmo de cobertura (*covering*). Porém elas diferem na forma como a cláusula é construída. Na abordagem *top-down* [34], a cláusula é construída do mais geral para o específico, sendo que a busca geralmente começa da cláusula mais geral e sucessivamente a especializa com predicados do conhecimento preliminar, de acordo com alguma heurística de busca. Na abordagem *bottom-up* [23], a busca começa com a hipótese mais específica, com o conjunto de exemplos, e constrói as cláusulas do específico para o geral através da generalização de cláusulas mais específicas. A Abordagem *top-down* mais popular em ILP é o FOIL[34], enquanto que a abordagem *bottom-up* mais popular em ILP é o GOLEM[27]. Progol [24] e Aleph[41] são sistemas que combinam *top-down* e *bottom-up*. Ele combinam o FOIL com implicação inversa (*inverse entailment*), originada a partir da abordagem *bottom-up*. Para mais informações sobre implicação inversa consulte [23].

Progol e Aleph usam um simples algoritmo de cobertura, como o do FOIL, no qual cada iteração realiza os seguintes passos:

1. Seleciona aleatoriamente um exemplo positivo para ser generalizado. Este exemplo deve pertencer ao conjunto de exemplos de treinamento positivos que ainda não foram cobertos. Se não existirem mais exemplos, para;

2. **Saturação**: constrói a cláusula mais específica, variabilizada, que implique o exemplo selecionado no passo anterior. A cláusula minimamente generalizada é chamada de *Bottom Clause*.

3. **Redução**: realiza uma busca por uma "boa" cláusula entre a cláusula maximamente geral, a cláusula de corpo vazio e a *Bottom Clause* maximamente específica. A "boa qualidade"de cada cláusula encontrada ao longo do caminho de busca é medida utilizando uma função de avaliação;

4. **Remoção da cobertura**: adiciona a cláusula de melhor qualidade à teoria e remove todos os exemplos positivos cobertos por ela da base de exemplos;

5. Repete até que todos os exemplos positivos sejam cobertos.

O Progol e o Aleph, assim como a maioria dos sistemas de aprendizado de teorias, aprendem um conceito por vez. A seguir explicaremos o que vem a ser Modos e *Determinations*, que são conceitos necessários para se entender como que a *Bottom Clause* é construída nestes sistemas.

2.3.1 Declaração de Modos [19]

Já que a *Bottom Clause* pode ter um número infinito de literais no corpo, ou pode ser exponencialmente grande dependendo do número de átomos básicos usados para descrever os exemplos, Progol e Aleph usam uma técnica chamada "declaração de modos"juntamente com o limite de profundidade de variável para restringir o tamanho da *Bottom Clause*. O limite de profundidade de uma variável em uma determinada cláusula é a profundidade máxima que a árvore de construção da *Bottom Clause* pode atingir, onde em cada nível da árvore novas variáveis são criadas a partir das variáveis do nível anterior. Este conceito de profundidade será melhor explicado no exemplo de construção da *Bottom Clause* dado mais adiante.

As declarações de modos são fornecidas pelo especialista do domínio e descrevem as relações (predicados) entre objetos de dados e tipos desses dados. Estas declarações permitem também informar se a relação pode ser utilizada na cabeça (declarações *modeh*) ou no corpo (declarações *modeb*) das regras geradas. Na declaração de modos também são descritos o tipo dos argumentos e o número máximo de instanciações de cada predicado. As declarações tem o formato $mode(Numero\_Chamadas, Modo)$. O $Numero\_Chamadas$, ou como é mais conhecido, o $recall\_number$, determina o limite do número de instanciações alternativas de um predicado. O $recall\_number$ pode ser qualquer número positivo $n \geq 1$ ou '*'. Se é conhecido que há somente um certo número de instanciações possíveis para um determinado predicado, é possível informar isto pelo $recall\_number$. O $recall\_number$ '*' é utilizado quando não há limite para o número de instanciações de um predicado.

A segunda parte de declaração de *mode/2*, denominada de *Modo*, indica o formato do predicado que será utilizado da seguinte forma:

$$predicado(TipoArgumento\_1, TipoArgumento\_2, ..., TipoArgumento\_n).$$

Para o aprendizado da relação $sogra\_de(Sogra, Genro)$ com conhecimento preliminar descrito, por exemplo, pelas relações $progenitor\_de(Mae, Filha)$ e $esposa\_de(Esposa, Marido)$ as declarações de modo podem ser:

$$:-modeh(1, sogra\_de(+mulher, +homem)).$$
$$:-modeb(*, progenitor\_de(+mulher, -mulher)).$$
$$:-modeb(1, esposa\_de(+mulher, +homem)).$$

onde os símbolos + e − serão explicados a seguir. A declaração *modeh/2* indica o predicado que irá compor a cabeça das regras. O valor do *recall_number* é 1 para esse predicado, pois o predicado pode ter uma única resposta (sim ou não), dados os dois argumentos. No exemplo, *modeh/2* informa que a cabeça das regras deve ser $sogra\_de(Sogra, Genro)$, onde *Sogra* é do tipo *mulher* e *Genro* é do tipo *homem*. As declarações *modeb/2* indicam que as regras geradas podem ter no corpo os predicados $progenitor\_de(Mae, Filha)$ e $esposa\_de(Esposa, Marido)$, nos quais *Mae*, *Filha* e *Esposa* são do tipo *mulher* e *Marido* é do tipo *homem*. O valor do *recall_number* para o predicado $progenitor\_de(Mae, Filha)$ é igual a '*', pois não se sabe quantas filhas uma mãe pode ter. Para o predicado $esposa\_de(Esposa, Marido)$ esse parâmetro tem valor '1', pois uma esposa só possui um único marido, e vice-versa.

Os tipos de argumentos podem ser +, − ou #. O símbolo '+' indica que o argumento do predicado é uma variável de entrada. O símbolo '−' indica uma variável de saída e o símbolo '#' indica que esse argumento é uma constante. A utilização desses tipos de variáveis devem seguir as regras listadas a seguir:

- **Variável de Entrada(+):** Qualquer variável de entrada do tipo T em um literal do corpo B (modeb) deve aparecer ou como uma variável de saída do tipo T em um literal do corpo antes de B, ou como uma variável de entrada do tipo T na cabeça, como mostra o exemplo a seguir:

  $:-modeh(1, sogra\_de(+mulher, +homem)).$
  $:-modeb(*, progenitor\_de(+mulher, -mulher)).$
  $:-modeb(1, esposa\_de(+mulher, +homem)).$

  A variável de entrada do tipo *homem* do predicado *esposa_de/2* vem da variável de mesmo tipo da cabeça. A variável de entrada do tipo *mulher* do predicado *esposa_de/2* vem ou da variável de saída de mesmo tipo do predicado *progenitor_de/2* ou da variável de entrada de mesmo tipo da cabeça.

- **Variável de Saída (-):** Qualquer variável de saída do tipo T na cabeça (*modeh*) deve aparecer como uma variável de saída do tipo T em um literal do corpo B. Qualquer literal que possua variável de saída deve ter, pelo menos, uma variável de entrada. Como mostra o exemplo a seguir:

  $:-modeh(1, sogra\_de(+mulher, -homem)).$
  $:-modeb(*, progenitor\_de(+mulher, -mulher)).$
  $:-modeb(1, esposa\_de(+mulher, -homem)).$

A variável de saída do tipo *homem* da cabeça vem da variável de saída de mesmo tipo do predicado $esposa\_de/2$.

- **Constante:** Qualquer argumento denotado por $\#T$ só pode ser substituído por constantes.

Os tipos devem ser especificados para cada argumento dos predicados utilizados na construção de uma hipótese. Para o Progol, os tipos são predicados, e a declaração de tipos nada mais é que um conjunto de fatos. Progol reconhece argumentos de tipos diferentes e os trata distintamente. Por exemplo, a descrição dos objetos dos tipos *homem* e *mulher* poderia ser:

$mulher(jane)$.
$mulher(sueli)$.
$homem(henrique)$.
$homem(junior)$.
...

2.3.2 Declaração dos *Determinations* [19]

O $determination/2$ serve para declarar os predicados que podem ser usados no corpo de um predicado alvo. Essa declaração tem o formato:

$determination(PredAlvo/Aridade\_a, PredCorpo/Aridade\_c)$.

O primeiro argumento é o nome do predicado alvo e sua aridade, isto é, o predicado que irá aparecer na cabeça da regra induzida. O segundo argumento consiste do nome e aridade de um predicado que pode aparecer no corpo da cláusula. Por exemplo, para o aprendizado da relação $sogra\_de(Sogra, Genro)$, uma declaração seria:

$:-determination(sogra\_de/2, progenitor\_de/2)$.
$:-determination(sogra\_de/2, esposa\_de/2)$.

Para que um predicado seja utilizado na construção de uma hipótese, é necessário que ele tenha declarações tanto de $modeb/2$ quanto de $determination/2$. Caso uma das declarações esteja ausente, o predicado não será utilizado na construção da hipótese.

Algoritmo 2.1 Algoritmo Progol/Aleph para construção da *Bottom Clause*

Seja e um exemplo positivo não provado.
$hash : Terms \rightarrow N$ é uma função que univocamente mapeia termos em variáveis distintas.

1: Adicione e ao conhecimento preliminar.
2: $InTerms = \emptyset$, $\overrightarrow{\perp} = \emptyset$.
3: Ache a primeira declaração mode de cabeça h tal que h subsume e com substituição θ
4: **para cada** v/t em θ **faça**
5: **se** Se v corresponde a um tipo # **então**
6: substitua v em h por t.
7: **se** v corresponde a um tipo + ou − **então**
8: substitua v em h por v_k onde v_k é uma variável tal que $k = hash(t)$.
9: **se** v corresponde a um tipo + **então**
10: adicione t ao conjunto $InTerms$.
11: Adicione h a $\perp$.
12: **para cada** declaração *modeb* de corpo b **faça**
13: **para toda** substituição possível θ de argumentos correspondendo a tipo + por termos no conjunto $InTerms$ **faça**
14: **repita**
15: **se** Prolog tem sucesso no objetivo b com substituição θ' **então**
16: **para cada** v/t em θ e θ' **faça**
17: **se** v corresponde a tipo # **então**
18: substitua v em b por t
19: **senão**
20: substitua v em b por v_k onde $k = hash(t)$
21: **se** v corresponde a tipo − **então**
22: adicione t ao conjunto $InTerms$
23: Adicione b a $\overrightarrow{\perp}$
24: **até que** se tenha executado o loop *Recall* vezes
25: Incremente a profundidade de variável
26: Vá para o passo 12 se a profundidade máxima de variável não foi alcançada
27: Retorne $\overrightarrow{\perp}$ variabilizada usando-se a função $hash$.

2.3.3 Construção da *Bottom Clause* [19]

O Algoritmo 2.1 corresponde ao passo de saturação do Progol/Aleph responsável pela geração da *Bottom Clause* a partir de um exemplo positivo não provado.

A lista *InTerms* guarda os termos que instanciam argumentos de entrada no predicado da cabeça, e os termos que instanciam argumentos de saída nos predicados do corpo. A função *hash* associa a cada termo uma variável diferente. O *recall* é usado para determinar quantas vezes chamar o interpretador Prolog para cada instanciação da cláusula no passo 12. A profundidade máxima de variável determina quantas vezes o passo 12 será executado. Podemos considerar a construção da

Bottom Clause como uma árvore, onde cada vez que o passo 12 é executado, um nível da árvore é contruído.

Para ilustrar o passo de construção de *Bottom Clause* considere a base de dados a seguir [19]:

$: -modeh(1, sogra\_de(+mulher, -homem)).$
$: -modeb(*, progenitor\_de(+mulher, -mulher)).$
$: -modeb(1, esposa\_de(+mulher, -homem)).$
$: -determination(sogra\_de/2, progenitor\_de/2).$
$: -determination(sogra\_de/2, esposa\_de/2).$

/* declaração dos tipos */
$homem(pai1).$
$homem(marido1).$
$homem(marido2).$
$mulher(mae1).$
$mulher(filha11).$
$mulher(filha12).$
$mulher(neta11).$

/* Conhecimento preliminar */
$progenitor\_de(mae1, filha11).$
$progenitor\_de(pai1, filha11).$
$progenitor\_de(mae1, filha12).$
$progenitor\_de(pai1, filha12).$
$progenitor\_de(filha11, neta11).$
$esposa\_de(mae1, pai1).$
$esposa\_de(filha11, marido1).$
$esposa\_de(filha12, marido2).$

/* Exemplos positivos */
$sogra\_de(mae1, marido1).$
$sogra\_de(mae1, marido2).$

/* Exemplos negativos */
$sogra\_de(mae1, pai1).$
$sogra\_de(filha11, marido2).$
$sogra\_de(filha11, marido1).$

A construção da *Bottom Clause* considerando o exemplo positivo $sogra\_de(mae1, marido1)$ segue os passos abaixo:

No início do Algoritmo 2.1, a lista *InTerms* e a *Bottom Clause* estão vazias. O primeiro passo do algoritmo é achar uma declaração *modeh* tal que h subsume o exemplo e com uma determinada substituição θ. A primeira e única declaração *modeh* para este exemplo é:

$modeh(1, sogra\_de(+mulher, -homem))$.

A substituição θ é dada por:

$\theta = \{+mulher/mae1, -homem/marido1\}$.

Como os argumentos de *sogra_ de* correspondem a tipos + e −, os argumentos serão substituídos por duas variáveis novas A e B, retornando o predicado $sogra\_de(A, B)$, onde o A corresponderá a *mae1* e o B a *marido1*. O termo *mae1* é colocado na lista *InTerms* pois o argumento substituído por *mae1* é do tipo +. O predicado $sogra\_de(A, B)$ é colocado na *Bottom Clause*. Portanto temos:

$InTerms = \{mae1\}$, $\perp = \{sogra\_de(A, B)\}$

Neste momento temos o nível 0 da árvore, mostrado na Figura 2.2.

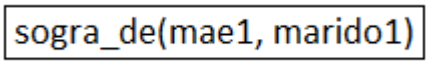

Figura 2.2: árvore de construção da *Bottom Clause* - nível 0

Os próximos passos são feitos para cada declaração *modeb*. A primeira declaração é:

$modeb(*, progenitor\_de(+mulher, -mulher))$

As possíveis substituições θ de argumentos correspondentes a tipo + por termos no conjunto *InTerms* são dadas por:

$\theta = \{+mulher/mae1\}$

Como o *recall* de *progenitor_ de* é $*$, será repetido o passo a seguir para cada fato do predicado *progenitor_ de* que tenha o termo *mae1* como primeiro argumento. O primeiro fato encontrado é $progenitor\_de(mae1, filha11)$, com substituição $\theta' = \{-mulher/filha11\}$.

Os argumentos de *progenitor_ de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável A, e a variável C é introduzida para o termo *filha11*. O predicado *progenitor_ de(A,C)* é colocado na *Bottom Clause*. Como o termo *filha11* instancia um argumento do tipo −, ele é incluído na lista *InTerms*. Portanto temos:

$$InTerms = \{mae1, filha11\},\ \perp = \{sogra\_de(A, B), progenitor\_de(A, C)\}$$

O segundo fato encontrado é $progenitor\_de(mae1, filha12)$, com substituição $\theta' = \{-mulher/filha12\}$.

Os argumentos de *progenitor_ de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável A, e a variável D é introduzida para o termo *filha12*. O predicado *progenitor_ de(A,D)* é colocado na *Bottom Clause*. Como o termo *filha12* instancia um argumento do tipo $-$, ele é incluído na lista *InTerms*. Portanto temos:

$$InTerms = \{mae1, filha11, filha12\},$$
$$\perp = \{sogra\_de(A, B), progenitor\_de(A, C), progenitor\_de(A, D)\}$$

Não há mais fatos para *progenitor_ de* tendo *mae1* como primeiro argumento. Portanto passamos para a próxima declaração *modeb*, dada por:

$$modeb(1, esposa\_de(+mulher, -homem))$$

Consideramos a substituição $+mulher/mae1$. Como o *recall* de *esposa_ de* é 1 será repetido o passo a seguir apenas uma vez. É procurado um fato com o predicado *esposa_ de* que tenha *mae1* como primeiro argumento. O fato encontrado é $esposa\_de(mae1, pai1)$, com substituição $\theta' = \{-homem/pai1\}$.

Os argumentos de *esposa_ de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável A, e a variável E é introduzida para o termo *pai1*. O predicado *esposa_ de(A,E)* é colocado na *Bottom Clause*. Como o termo *pai1* instancia um argumento do tipo $-$, ele é incluído na lista *InTerms*. Portanto temos:

$$InTerms = \{mae1, filha11, filha12, pai1\},$$
$$\perp = \{\ sogra\_de(A, B), progenitor\_de(A, C), progenitor\_de(A, D),$$
$$esposa\_de(A, E)\}$$

Neste momento formamos o nível 1 da árvore de construção da *Bottom Clause*, como mostrado na Figura 2.3.

Pela Figura 2.3 podemos ver que o nível 1 da árvore é formado por todos os fatos envolvendo os predicados do corpo *progenitor_ de* e *esposa_ de* que contenham como argumento de entrada o termo *mae1*, que é justamente o termo de entrada vindo da cabeça da regra dada por $sogra\_de(mae1, marido1)$.

De acordo com a própria definição de variável de entrada, temos que uma variável de entrada em um literal do corpo deve ter aparecido antes como variável de saída em um outro literal do corpo, ou como variável de entrada no literal da cabeça. Como neste momento do algoritmo estamos formando os primeiros literais do corpo, são gerados literais cujas variáveis de entrada destes sejam variáveis de entrada do

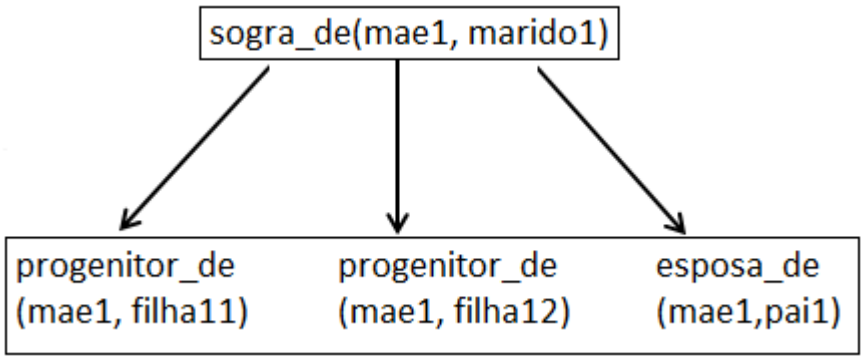

Figura 2.3: Árvore de construção da *Bottom Clause* - nível 1

literal da cabeça, e até agora o único termo que representa uma variável de entrada no literal da cabeça é *mae1*.

Formado o nível 1 da árvore, passamos para o passo 25 do Algoritmo 2.1 que corresponde a incrementar a profundidade de variável. Podemos entender claramente agora o que significa esta profundidade. A profundidade máxima de variável é um parâmetro que irá indicar quantos níveis poderemos ter, no máximo, na árvore de construção da *Bottom Clause*, começando-se pelo nível 0, e este parâmetro deverá ser definido no início do algoritmo. Em cada nível, novas variáveis serão introduzidas a partir dos termos do nível anterior. Formalmente, a profundidade de variável, a qual representaremos pela letra i, é definida como:

Definição 4. (Profundidade de Variável i) [42] Seja C uma cláusula definida e v uma variável em C. A profundidade i de v é definida como:

$$d(v) = \begin{cases} 0 & \text{se } v \text{ está na cabeça de } C \\ (\max_{u \in U_v} d(u)) + 1 & \text{caso contrário} \end{cases} \tag{2.1}$$

onde U_v são as variáveis dos átomos do corpo de C contendo v.

Até o momento, portanto, temos que as variáveis A e B da cabeça da *Bottom Clause* possuem profundidade 0, e as variáveis C, D e E presentes no corpo da *Bottom Clause* possuem profundidade 1. Se continuarmos o algoritmo estaremos permitindo que novos literais do corpo sejam gerados a partir dos novos termos introduzidos no nível 1, dados por *filha11*, *filha12* e *marido1*. Estes termos foram introduzidos como argumentos de saída e portanto agora poderão ser argumentos de entrada nos novos literais do corpo que serão gerados. Continuando o Algoritmo 2.1 e voltando para o passo 12 teremos:

A primeira declaração modeb encontrada é:

$$modeb(*, progenitor\_de(+mulher, -mulher))$$

que terá o seu argumento de entrada substituído por *filha11* e *filha12*, ambos do tipo *mulher*. Como o *recall* é $*$, podemos buscar quantos fatos tiverem para as substituições dadas. Apenas o fato $progenitor\_de(filha11, neta11)$ será encontrado. O termo *filha11* já está vinculado à variável C, e o termo *neta11* será vinculada à nova variável F. Portanto temos:

$$\begin{aligned}\perp = \{\ & sogra\_de(A,B), progenitor\_de(A,C), progenitor\_de(A,D),\\ & esposa\_de(A,E), progenitor\_de(C,F)\}\end{aligned}$$

Para a próxima declaração modeb dada por:

$$modeb(1, esposa\_de(+mulher, -homem))$$

teremos o argumento de entrada substituído por *filha11* e *filha12*, ambos do tipo *mulher*. Não podemos utilizar o termo *pai1* pois este é do tipo *homem*. Como o *recall* é 1, só poderemos buscar um fato para *filha11* como primeiro argumento e um fato para *filha12* como primeiro argumento. Os fatos encontrados serão $esposa\_de(filha11, marido1)$ e $esposa\_de(filha12, marido2)$. Apenas o termo marido2 não havia sido introduzido antes, e portanto é vinculado à nova variável G. Temos então:

$$\begin{aligned}\perp = \{\ & sogra\_de(A,B), progenitor\_de(A,C), progenitor\_de(A,D),\\ & esposa\_de(A,E), progenitor\_de(C,F), esposa\_de(C,G)\}\end{aligned}$$

Neste momento formamos o nível 2 da árvore de construção da *Bottom Clause*, mostrada na Figura 2.4, tal que as variáveis F e G possuem profundidade 2.

Se continuássemos o algoritmo e fôssemos para o próximo nível da árvore, procurariamos fatos que tivessem os termos *neta11*, *marido1* e *marido2* como argumentos de entrada nos literais do corpo, pois estes foram os novos termos introduzidos no nível anterior como argumentos de saída. Porém, olhando para o conhecimento preliminar deste exemplo, veremos que tais fatos não existem e portanto o algoritmo irá terminar. Portanto a *Bottom Clause* final encontrada é:

$$\begin{aligned}\perp = \{\ & sogra\_de(A,B), progenitor\_de(A,C), progenitor\_de(A,D),\\ & esposa\_de(A,E), progenitor\_de(C,F), esposa\_de(C,G)\}\end{aligned}$$

2.4 Operadores de Refinamento

Nesta seção serão explicados os principais conceitos relativos à reticulados, ordens de generalidade e operadores de refinamento, e que serão a base para o entendimento do trabalho proposto. Tais conceitos foram retirados de [3, 20, 31, 35, 42], e mais detalhes acerca deste tema podem ser encontrados em tais referências.

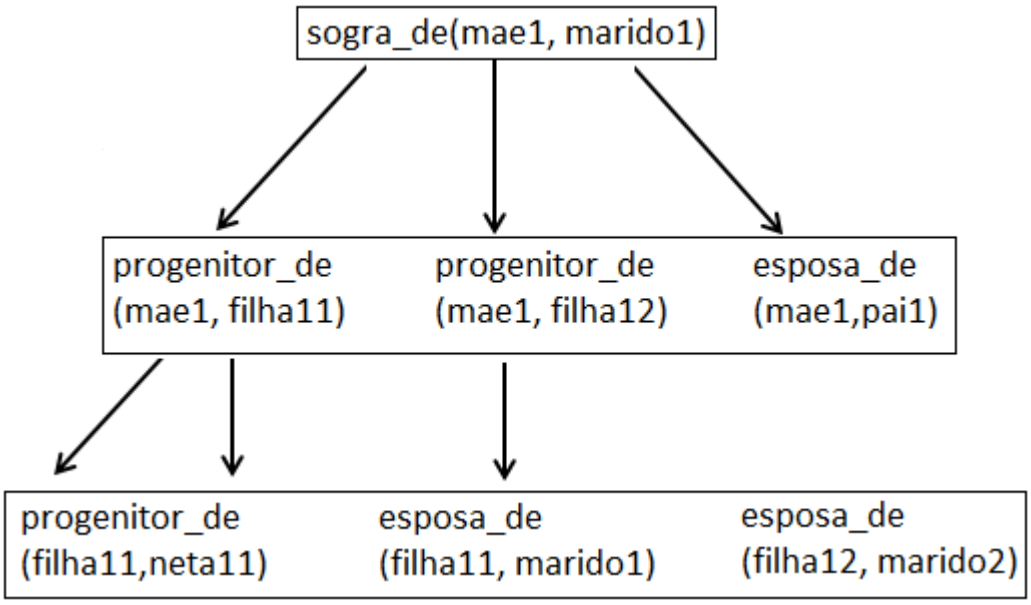

Figura 2.4: Árvore de construção da *Bottom Clause* - nível 2

Quando aprendendo ou revisando uma teoria, normalmente não é óbvio qual conjunto de cláusulas devemos considerar como sendo a teoria desejada. Se uma ou mais teorias corretas existem, então elas estão "escondidas"em algum lugar no conjunto de cláusulas da linguagem usada. Para achar tais teorias é preciso buscar entre as cláusulas permitidas por um conjunto de cláusulas com as propriedades certas. O conjunto de cláusulas que podem ser incluídas na teoria é chamado de *Espaço de Busca* [31].

No geral, achar a teoria desejada equivale a repetidamente ajustar a teoria aos exemplos através de passos de especialização e generalização. Tais passos são conhecidos como passos de refinamento, e são executados através de *operadores de refinamento descendentes* e *ascendentes*, respectivamente [31]. O que constitui uma *especialização* ou *generalização* de uma cláusula é determinado pela *ordem de generalidade* aplicada às cláusulas. Existem diferentes ordens de generalidade, sendo que duas das mais importantes são a ordem geral de θ*-subsumption* e a *implicação lógica*.

Todas as ordens de generalidade a serem consideradas nesta Tese são chamadas de *quasi-ordens*. Uma *quasi-ordem* pode ser vista como uma relação com certas propriedades. Uma relação R é definida num conjunto G e é um subconjunto dos pares ordenados dos elementos de G. Se um par $(a, b) \in R$ então escrevemos aRb.

Definição 5. [31] Seja R uma relação em G.
1. R é *reflexiva* se para todo $x \in G, xRx$ é verdade.
2. R é *simétrica* se para todo $x, y \in G, xRy$ implica que yRx.
3. R é *transitiva* se para todo $x, y, z \in G, xRy$ e yRz implica que xRz.

4. R é *antissimétrica* se para todo $x, y \in G, xRy$ e yRx implica que $x = y$.

Definição 6. [31] Seja R uma relação em G.
1. R é chamada de *quasi-ordem* em G, se R é reflexiva e transitiva. O par $\langle G, R\rangle$ é chamado de *conjunto quasi-ordenado*.
2. R é chamada de *ordem parcial* em G, se R é reflexiva, transitiva e antissimétrica. O par $\langle G, R\rangle$ é chamado de *conjunto parcialmente ordenado*.
3. R é chamada de *relação equivalente* em G, se R é reflexiva, simétrica e transitiva.

Definição 7 (relação de equivalência $\approx$)**.** [31] Seja $\langle G, \geq\rangle$ um conjunto quasi-ordenado. Para todo $x, y \in G$, escrevemos que $x \approx y$, isto é, x é equivalente a y, se e somente se $x \geq y$ e $y \geq x$.

Definição 8. [17] Uma teoria é **redundante** se é equivalente a um dos seus sub-conjuntos próprios; Uma parte da teoria é **redundante** se a remoção desta parte não altera semanticamente a teoria.

Definição 9. [31] Seja $\langle G, \geq\rangle$ um conjunto quasi-ordenado, e $S \subset G$. Um elemento $x \in G$ é chamado de **limite superior** de S se $x \geq y$ para todo $y \in S$. Um limite superior x de S é chamado de **limite superior mínimo (lub)** de S, se $z \geq x$ para todos os limites superiores z de S. Um elemento $x \in G$ é chamado de **limite inferior** de S se $y \geq x$ para todo $y \in S$. Um limite inferior x de S é chamado de **limite inferior máximo (glb)** de S, se $x \geq z$ para todos os limites inferiores z de S. Se para algum par $x, y \in G$ temos mais de um *lub*, então $x \sqcup y$ denota um *lub* arbitrário. Como todos os $lub's$ são equivalentes sob $\approx$, para cada x, y, um $x \sqcup y$ é equivalente a todos os outros $lub's$ de $\{x, y\}$. Alé disso, se $x \approx x'$ e $y \approx y'$, então $x \sqcup y \approx x' \sqcup y'$. Analogamente, temos que $x \sqcap y$ denota um *glb* arbitrário.

Definição 10. [31] Seja $\langle G, \geq\rangle$ um conjunto quasi-ordenado. Se para todo $x, y \in G$, existe um *lub* de $\{x,y\}$ e um *glb* de $\{x,y\}$ em G, então $\langle G, \geq\rangle$ é chamado de **reticulado**.

Seja $\mathcal{C}$ um conjunto de cláusulas, $S \subset \mathcal{C}$, e $\geq$ uma quasi-ordem em $\mathcal{C}$. As seguintes definições são usadas em ILP [31]:

- Se $C, D \in \mathcal{C}$ e $C \geq D$ então C é chamado uma *generalização* de D, e D é uma *especialização* de C.
- Um limite superior C $\in \mathcal{C}$ de S é chamado uma *generalização* de S.
- Um lub C $\in \mathcal{C}$ de S é chamado uma *generalização mínima (LG)* de S.
- Um limite inferior C $\in \mathcal{C}$ de S é chamado uma *especialização* de S.

- Um glb C $\in \mathcal{C}$ de S é chamado uma *especialização máxima (GS)* de S.

A *subsumption* proposicional é uma das ordens de generalidade em cláusulas mais simples, enquanto que a *θ-subsumption* é uma das mais importantes. Ambas as relações são reflexivas e transitivas, e portanto elas impõem uma quasi-ordem em um conjunto de cláusulas. Elas são definidas como:

Definição 11 (*Subsumption* Proposicional)**.** [35] Sejam as cláusulas C e D. Dizemos que C subsume D proposicionalmente se e somente se $C \subseteq D$.

Definição 12 (*θ-subsumption)***.** [31] Sejam as cláusulas C e D. Dizemos que C **subsume** D ($C \succeq D$) se existe uma substituição θ tal que $C\theta \subseteq D$. C **propriamente subsume** D ($C \succ D$) se $C \succeq D$ e $D \not\succeq C$. C e D são **subsume-equivalentes** ($C \sim D$) se $C \succeq D$ e $D \succeq C$.

Apesar de que na matemática o conceito de reticulado é normalmente definido em uma ordem parcial, a definição em uma quasi-ordem é mais conveniente em ILP, já que, geralmente, lidamos com quasi-ordem em cláusulas, mesmo quando o interesse é nas classes de equivalência de cláusulas. A relação de *θ-subsumption* em cláusulas, por exemplo, é reflexiva e transitiva, portanto ela impõe uma quasi-ordem no conjunto de cláusulas. Porém, ela não é antissimétrica já que podem existir hipóteses que cobrem exatamente o mesmo conjunto de exemplos. Tais hipóteses são chamadas **variantes sintáticas**, e não se restringem à renomeação de variáveis [35]. Observe, por exemplo, as cláusulas abaixo:

Exemplo 1. *[35]*

$pai(X, Y) \leftarrow mae(X, Y).$
$pai(X, Y) \leftarrow mae(X, Y), mae(X, Z).$
$pai(X, Y) \leftarrow mae(X, Y), mae(X, Z), mae(X, W).$

As cláusulas deste exemplo são sintaticamente diferentes, mas são equivalentes com relação à *θ-subsumption*, e portanto são variantes sintáticas. De qualquer forma, se temos um reticulado em uma quasi-ordem, também temos uma reticulado na ordem parcial sobre as classes de equivalência induzidas pela quasi-ordem [31]. Em [31] é mostrado como transformar uma quasi-ordem definida para algum conjunto G numa ordem parcial no conjunto de classes de equivalências de G.

A ordem de *θ-subsumption* também pode ser definida para teorias, como se segue:

Definição 13 (*θ-subsumption* em Teorias)**.** [2, 20] Uma teoria T_1 θ-subsume uma teoria T_2 ($T_1 \succeq T_2$) se e somente se $\forall c_2 \in T_2 \exists c_1 \in T_1$ tal que $c_1 \succeq c_2$. Duas teorias T_1 e T_2 são θ-equivalentes ($T_1 \sim T_2$) se e somente se $T_1 \succeq T_2$ e $T_2 \succeq T_1$.

O conceito de *θ-subsumption* em cláusulas é diferente do de *θ-subsumption* em teorias. Esta última generaliza a relação de superconjunto em um conjunto de cláusulas, ao contrário do que acontece em *θ-subsumption* em cláusulas, que considera a relação de subconjunto em um conjunto de literais. Logo, para generalizar uma teoria em particular, é suficiente adicionar uma cláusula que não é equivalente a nenhuma outra cláusula na teoria [20].

Definição 14 (Operador de Refinamento)**.** [42] Seja $\langle G, \geq \rangle$ um conjunto quasi-ordenado. Um **operador de refinamento descendente** para $\langle G, \geq \rangle$ é uma função ρ, tal que $\rho(C) \subseteq \{D | C \geq D\}$, para todo $C \in G$.

• Os conjuntos de refinamentos **um-passo**, **n-passos**, e **todos os refinamentos** de algum $C \in G$ são respectivamente:

$$\rho^1(C) = \rho(C)$$

$$\rho^n(C) = \{D | \text{ existe um } E \in \rho^{n-1}(C) \text{ tal que } D \in \rho(E)\}, n \geq 2$$

$$\rho^*(C) = \rho^1(C) \cup \rho^2(C) \cup \rho^3(C) \cdots$$

• Uma **ρ-cadeia** de C a D é uma sequência $C = C_0, C_1, \cdots, C_n = D$ tal que $C_i \in \rho(C_{i-1})$ para todo $1 \leq i \leq n$.

• ρ é **localmente finito** se para todo $C \in G$, $\rho(C)$ é finito e computável.

• ρ é **próprio** se para todo $C \in G, \rho(C) \subseteq \{D | C > D\}$.

• ρ é **completo** se para todo $C, D \in G$ tal que $C > D$, existe um $E \in \rho^*(C)$ tal que $D \sim E$.

• ρ é **fracamente completo** se $\rho^*(\square) = G$, onde $\square$ é o elemento mais geral de G.

• ρ é **não-redundante** se para todo $C, D, E \in G, E \in \rho^*(C)$ e $E \in \rho^*(D)$ implica $C \in \rho^*(D)$ ou $D \in \rho^*(C)$.

• ρ é **ideal** se ele é localmente finito, completo, e próprio.

• ρ é **ótimo** se ele é localmente finito, não-redundante e fracamente completo.

Conceitos análogos podem ser definidos para operadores de refinamento ascendentes. Um **operador de refinamento ascendente** para $\langle G, \geq \rangle$ é uma função δ, tal que $\delta(C) \subseteq \{D | D \geq C\}$, para todo $C \in G$.

Uma maneira de refinar teorias é simplesmente refinar uma de suas cláusulas. Operadores de refinamento de teorias geralmente empregam um operador de refinamento clausal subjacente. Ele também compartilham das mesmas propriedades definidas para os operadores de refinamento de cláusulas na Definição 14.

Um operador de refinamento induz um *grafo de refinamento* direcionado, que pode ser visto como o espaço de busca de candidatos a serem incluídos na teoria.

Neste grafo cada nó é um elemento de G, e existe uma aresta de C a D no caso de $D \in \rho(C)$. Um grafo de refinamento induzido por um operador ideal não possui ciclos (próprio), um número finito de arestas partem de cada nó (finitude local), e existe um caminho de tamanho finito de C para um membro da classe de equivalência de D quando $C \succ D$ (completude). Já num operador de refinamento ótimo existe exatamente um único caminho de C a D no grafo (uma única ρ-cadeia de C a D), o que significa que o grafo de refinamento se torna uma árvore, com $\square$ sendo a raíz [31].

A Figura 2.5 mostra parte de um grafo de refinamento (descendente) para a θ-*subsumption*, considerando um operador que ou unifica variáveis ou adiciona literais. Este operador de refinamento não é completo já que não inclui todos os possíveis refinamentos. Ele é próprio já que o grafo não contém ciclos. Ele é redundante porque não tem a estrutura de uma árvore e tem mais de um caminho de $p(x, y)$ para $p(x, x) \leftarrow q(x, z)$ [42].

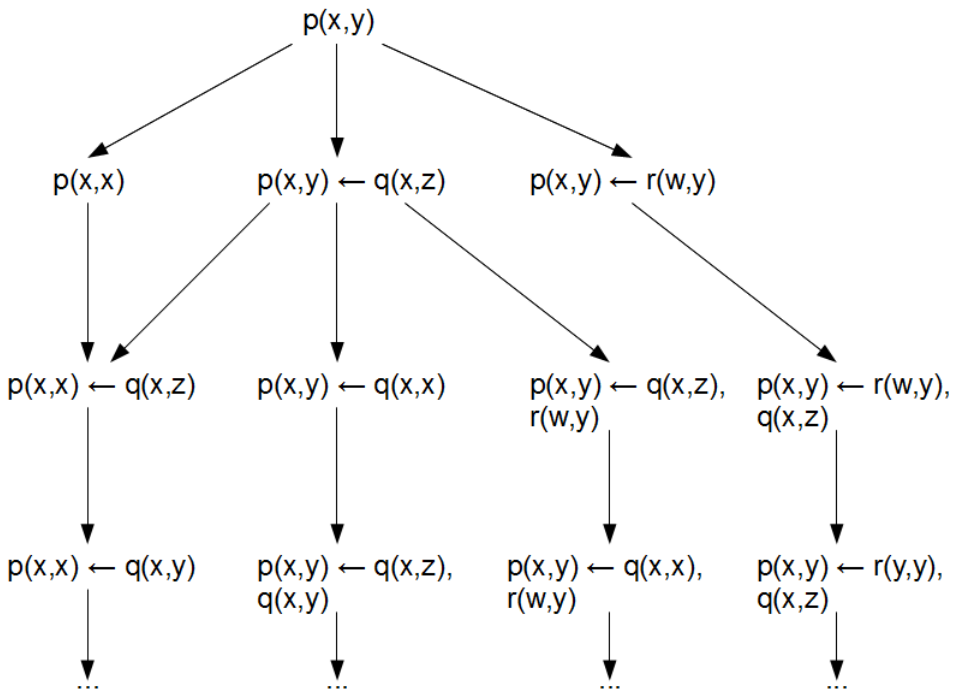

Figura 2.5: Parte de um grafo de refinamento representando refinamentos descendentes de uma cláusula

O FORTE_MBC usa a *Bottom Clause* para limitar o espaço de busca de cláusulas. A *Bottom Clause* é o elemento mais específico na linguagem de modos de profundidade limitada $L_i(M)$, que considera um conjunto de declarações de modos (cláusulas obedecendo a este conjunto de modos são pertencentes à linguagem $L(M)$) e um limite máximo de profundidade de variável i para restringir a busca por refinamentos que subsumam a *Bottom Clause*. A definição das linguagens $L(M)$ e

$L_i(M)$, assim como a definição formal de *Bottom Clause* são dadas a seguir:

Definição 15. (Linguagem de modos definida $L(M)$) [42] Seja C uma cláusula definida com uma ordenação total definida sobre os literais, e M o conjunto de declarações de Modos. $C = h \leftarrow b_1, ..., b_n$ está na linguagem de modos definida $L(M)$ se e somente se (1) h é o átomo de uma declaração *modeh* em M com todo *place-marker* +type e -type substituído por variáveis e todo *place-marker* #type substituído por um termo básico, e (2) todo átomo b_i no corpo de C é o átomo de uma declaração *modeb* em M com todo *place-marker* +type e -type substituído por variáveis e todo *place-marker* #type substituído por um termo básico, e (3) toda variável de +type em qualquer átomo b_i é ou do tipo +type em h ou do tipo -type em algum átomo b_j, $1 \leq j < i$.

Definição 16. ($L_i(M)$) [42] Seja C uma cláusula definida com uma ordenação total sobre os literais e M um conjunto de declarações de modos. C está em $L_i(M)$ se e somente se C está em $L(M)$ e todas as variáveis em C tem no máximo a profundidade i, de acordo com a definição 4.

Definição 17 (Cláusula mais específica ou *Bottom Clause* $\perp_i$)**.** [42] Seja h, i números naturais, B um conjunto de cláusulas de Horn, $e = a \leftarrow b_1, ..., b_n$ uma cláusula definida, M o conjunto de declarações de modos contendo exatamente um modeh m tal que $a(m) \succeq a$ e $\perp$ a cláusula definida mais específica (potencialmente infinita) tal que $B \wedge \perp \wedge \overline{e} \vdash_h \square$. $\perp_i$ é a cláusula mais específica em $L_i(M)$ tal que $\perp_i \succeq \perp$. Nesta Tese, iremos nos referir à *Bottom Clause* como $\overrightarrow{\perp}$.

De acordo com a Definição 16, cláusulas que são consideradas pelos operadores de refinamento de cláusulas do FORTE_MBC (i.e. cláusulas em $L_i(M)$) são definidas com uma ordenação total sobre os literais. Para caracterizar o espaço de busca e o refinamento de cláusulas do FORTE_MBC, usaremos as representações de **cláusulas ordenadas** e **subconjuntos ordenados** definidas em [42], que também foram consideradas para caracterizar refinamentos no Progol.

Definição 18 (Cláusula Ordenada)**.** [42]

Uma cláusula ordenada $\overrightarrow{C}$ é uma sequência de literais $L_1, L_2, ..., L_n$ e é denotada por $\overrightarrow{C} = L_1 \vee L_2 \vee ... \vee L_n$. O conjunto de literais de $\overrightarrow{C}$ é denotado por C.

Ao contrário de cláusulas convencionais, a ordem e a duplicação de literais importa para cláusulas ordenadas. Por exemplo, $\overrightarrow{C} = p(X) \vee \neg q(X)$, $\overrightarrow{D} = \neg q(X) \vee p(X)$ e $\overrightarrow{E} = p(X) \vee \neg q(X) \vee p(X)$ são cláusulas ordenadas diferentes, mas todas correspondem à mesma cláusula convencional $C = D = E = \{p(X), \neg q(X)\}$.

Definição 19 (Subconjunto Ordenado)**.** [42]

Sejam $\overrightarrow{C} = L_1 \vee L_2 \vee ... \vee L_l$ e $\overrightarrow{D} = M_1 \vee M_2 \vee ... \vee M_m$ cláusulas ordenadas. $\overrightarrow{C}$ é um subconjunto ordenado de $\overrightarrow{D}$, denotado por $\overrightarrow{C} \subseteq \overrightarrow{D}$, se existe uma função de seleção injetiva $s : [1..l] \rightarrow [1..m]$ tal que para cada (i, j) $\in s, L_i = M_j$.

Por exemplo, seja $\overrightarrow{C} = h(X) \leftarrow p(X), q(X, Y)$, $\overrightarrow{D} = h(X) \leftarrow q(X, Y), p(X), r(Y, Z)$ e $\overrightarrow{E} = h(X) \leftarrow p(X), q(X, Y), q(Y, X)$. As três cláusulas são cláusulas ordenadas diferentes. Temos que $\overrightarrow{C} \subseteq \overrightarrow{D}$, com $s = \{(1, 1), (2, 3), (3, 2)\}$ e $\overrightarrow{C} \subseteq \overrightarrow{E}$, com $s = \{(1, 1), (2, 2), (3, 3)\}$, tal que $C_1 = D_1 = E_1 = h(X)$.

Observe que a definição de subconjunto ordenado não exige que o mapeamento dos literais seja ordenado, ou seja, uma cláusula não é necessariamente subsequência da outra. No caso, a ordenação referida é devido apenas ao fato de que são consideradas cláusulas ordenadas na relação de subconjunto.

Capítulo 3

Revisão de Teorias de Primeira-Ordem a partir de Exemplos

A maioria das abordagens desenvolvidas em ILP lida com o aprendizado de teorias clausais de primeira-ordem a partir do zero, dado um conjunto de exemplos e um conhecimento preliminar, porém tambem existem abordagens que lidam com revisão de teorias [47], tais como FORTE [37], FORTE_MBC [10], AUDREY [46], CLINT [36], MIS [40], RUTH [1] e KRT [48], que usam uma teoria de primeira-ordem como ponto de partida e a revisam para que ela seja consistente com um conjunto de exemplos dados.

É possível que uma teoria previamente obtida esteja incorreta e/ou incompleta, seja porque ela foi extraída de um especialista do domínio e que pode estar se apoiando em suposições incorretas, ou porque existem novos exemplos que não podem ser explicados pela teoria atual. Neste caso, já que a teoria original contém informação relevante, seria vantajoso utilizar a teoria original como ponto de partida no processo de aprendizado, reparando-a ou melhorando-a, ao invés de descartá-las.

A tarefa de revisar teorias envolve mudar o conjunto de respostas da teoria dada, ou seja, melhorar a sua capacidade de inferência adicionando-se respostas que haviam sido perdidas ou removendo respostas incorretas. No primeiro caso estamos falando de generalização, e no segundo de especialização.

As teorias de primeira-ordem sendo revisadas normalmente possuem cláusulas que contém mais de um predicado. Portanto, a revisão de teorias deve ser capaz não apenas de modificar cláusulas individuais que definem um único predicado alvo, mas também deve saber lidar com o aprendizado de mútiplos predicados simultaneamente.

Considerando a linguagem de teorias como sendo de cláusulas definidas e a lin-

guagem de exemplos como sendo de átomos básicos, problema de revisão de teorias pode ser definido como se segue [47]:

Definição 20. *Dados*:

- Uma Teoria inicial H que pode ser modificada (a qual chamaremos de Teoria Corrente nesta Tese)
- Um conhecimento preliminar invariante (BK)
- Um conjunto de exemplos positivos E^+ e negativos E^-

Achar:

- Uma teoria revisada H' que, junto com o conhecimento preliminar BK, implique logicamente todos os exemplos positivos (completo do ponto de vista lógico), $BK \cup H' \models E^+$ e nenhum dos exemplos negativos (consistência), $\forall e^- \in E^- : BK \cup H' \not\models e^-$, e que esteja de acordo com o critério de qualidade, tal como o de minimalidade.

O esquema da tarefa de revisão de teorias é exibido na Figura 2.1.

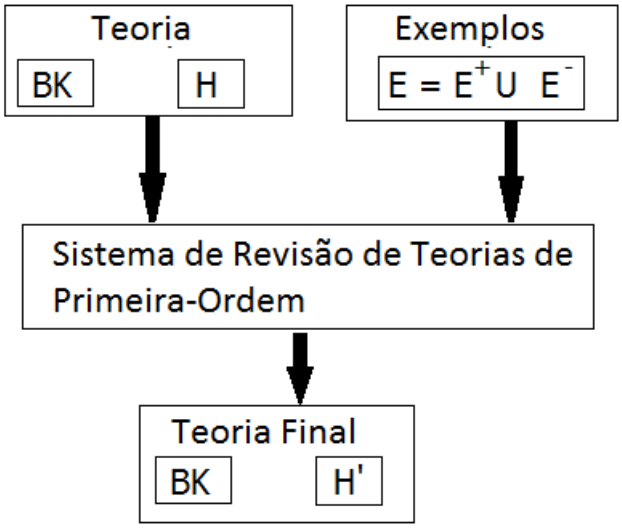

Figura 3.1: Esquema de um sistema de revisão de teoria, onde BK é o conhecimento preliminar, H' é a teoria inicial que pode ser modificada, E é o conjunto de exemplos positivos (E^+) e negativos (E^-) e H' é a teoria revisada pelo sistema

Sistemas de revisão de teorias assumem que a teoria inicial é aproximadamente correta. Sendo assim, apenas alguns pontos na teoria impedem que ela esteja totalmente correta. A ideia é, portanto, buscar tais pontos na teoria e revisá-los, ao invés de aprender uma nova teoria do zero. Geralmente, sistemas de revisão de teorias são compostos por três etapas de busca. Primeiro, eles buscam por pontos na teoria

responsáveis pela classificação incorreta de alguns exemplos. Segundo, eles buscam por possíveis modificações nestes pontos, através do uso de operadores de revisão. E finalmente, eles buscam pela melhor modificação entre todas as revisões propostas.

3.0.1 FORTE

Um exemplo de sistema de revisão de teorias é o FORTE (*First-Order Revision of Theories from Examples*) [37], que difere da maioria dos sistemas de revisão citados anteriormente pois faz revisão automática ao invés de interativa com o usuário, e utiliza operadores de generalização e especialização como espaço de busca.

FORTE é um sistema que revisa automaticamente teorias que contém cláusulas Horn de primeira-ordem livres de função. O FORTE executa uma busca *hill climbing* através do espaço de operadores de especialização e generalização, na tentativa de achar uma revisão mínima para uma teoria que a torne consistente com o conjunto de exemplos de treinamento. Dada uma teoria inicial incorreta e um conjunto de exemplos positivos e negativos, o FORTE deve achar uma teoria revisada que é consistente com os exemplos dados. FORTE revisa teorias iterativamente, usando uma busca *hill climbing* (ver Algoritmo 3.1). Cada iteração identifica pontos na teoria, chamados *pontos de revisão*, onde a revisão tem o potencial de melhorar a acurácia da teoria. É gerado então um conjunto de revisões, baseado nos pontos de revisão, selecionada a melhor revisão e implementada caso melhore a acurácia lógica da teoria. Para a nova teoria revisada, novos pontos de revisão são gerados. O processo de iteração continua até que nenhuma revisão melhore a teoria.

Para gerar pontos de revisão, a teoria corrente é testada no conjunto de treinamento. FORTE anota provas que falharam em exemplos positivos e provas que tiveram sucesso em exemplos negativos. A partir destas anotações ele identifica pontos na teoria para possível revisão. Cada ponto de revisão tem um potencial, que é o crescimento máximo em desempenho que pode ser proporcionado à teoria a partir de modificações nesse ponto (quantidade de exemplos que verificaram a necessidade de revisão no ponto). A pontuação de um operador é o crescimento real proporcionado ao desempenho da teoria após a revisão no ponto em que o operador foi aplicado (diferença entre exemplos que passaram a ser classificados corretamente e exemplos que deixaram de ser classificados corretamente). FORTE começa gerando revisões para o ponto de revisão com maior potencial. Para cada ponto de revisão escolhe-se a melhor revisão proposta, proporcionada pelo operador de maior pontuação. A revisão que realmente será efetuada é aquela definida como sendo a melhor revisão proposta entre todas. Este processo esá descrito no Algoritmo 3.1 a seguir.

Algoritmo 3.1 Algoritmo FORTE, proposto em [37]

```
1: repita
2:    Gera pontos de revisão
3:    Ordena pontos de revisão por potencial (do maior para menor)
4:    para cada ponto de revisão faça
5:       Gera possíveis revisões
6:       Atualiza melhor revisão encontrada
7:    até que o potencial do próximo ponto de revisão seja menor que a pontuação
      da melhor revisão atualizada
8:    se a melhor revisão melhorar a teoria então
9:       Implementa melhor revisão
10: até que nenhuma revisão melhore a teoria
```

Pontos de Revisão

Quando a teoria a ser revista é constituída de apenas um predicado, o exemplo selecionado, positivo ou negativo, indica que operação deve ser feita, se será uma generalização ou especialização.

No caso de uma teoria com múltiplas definições de predicados, muitas cláusulas podem estar envolvidas na prova de um exemplo negativo ou na não prova de um exemplo positivo. Dessa forma a indicação do tipo da operação não é imediata, tornando-se necessária a determinação dos pontos nessa teoria que precisam ser corrigidos (pontos de revisão). Dependendo do tipo de exemplo que esteja sendo considerado, podemos ter a definição de dois tipos de pontos de revisão: pontos de especialização e pontos de generalização. Pontos na teoria onde a prova de exemplos positivos falha são lugares onde a teoria precisa ser generalizada, e cláusulas usadas em provas de exemplos negativos são pontos onde a teoria precisa ser especializada.

A generalização ou especialização é efetivada pelos operadores de revisão. A especificação do ponto de revisão determina o tipo de operador de revisão que precisará ser aplicado para tornar a teoria consistente com a base de dados. A seguir descrevemos alguns dos operadores utilizados pelo FORTE [37].

- Operadores de Especialização:
  - *Exclusão de Regras* - Este operador comumente usado remove uma cláusula que tenha sido usada na prova de um exemplo negativo.
  - *Adição de Antecedentes* - Este operador adiciona antecedentes à uma cláusula incorreta para excluir exemplos negativos que estão sendo provados, não permitindo ao máximo possível que exemplos positivos deixem de ser provados.
- Operadores de generalização:

- *Exclusão de Antecedentes* - Este operador remove antecedentes que falharam na prova de exemplos positivos.
- *Adição de Regras* - Este operador gera novas cláusulas a partir de cláusulas existentes usando-se exclusão de antecedentes seguido de adição de antecedentes.
- *Adição de Novas Regras* - Este operador cria uma cláusula inteiramente nova.

Existem outras abordagens de operadores de especialização e generalização. Para mais detalhes em operadores de revisão sugerimos a leitura de [37] e [47].

3.1 FORTE_MBC

O custo da revisão de teorias de primeira-ordem no sistema FORTE é dominado pela busca por antecedentes a serem adicionados em uma cláusula, visto que o FORTE usa como espaço de busca todos os possíveis literais da base de conhecimento para serem adicionados na cláusula. No FORTE, os possíveis literais são gerados da mesma forma que é feita no sistema FOIL [34]: para cada predicado do conhecimento preliminar são gerados um conjunto de possíveis literais cujos argumentos são variáveis quaisquer, tal que as únicas restrições impostas são de que a tipagem definida para o predicado seja obedecida, e de que pelo menos uma variável do literal tenha aparecido antes na cláusula sendo revisada.

Apesar da forma como os literais são gerados ser ineficiente, ou seja, fazendo combinação de variáveis, o que realmente torna o processo de revisão custoso é a avaliação de cada um destes literais. O algoritmo de geração de novos antecedentes impõe apenas restrições de tipo nos argumentos dos predicados. Não é utilizada nenhuma definição de modos, ou seja, as variáveis não são restritas a serem de entrada e/ou de saída. O algoritmo também não impõe nenhuma restrição de *recall*, e também não limita a profundidade de variável da cláusula. Ao criar literais é realizada uma combinação de todas as possíveis variáveis que podem aparecer nesses literais, fazendo com que o número de antecedentes gerados em uma iteração seja muito grande. Frequentemente a combinação de variáveis gera muito mais literais do que necessário. Lembre que todos os literais gerados deverão ser avaliados para escolher aquele que será adicionado à cláusula. Entretanto, alguns literais podem não cobrir nem mesmo um exemplo positivo, visto que eles são gerados sem levar em consideração os exemplos (abordagem *top-down*).

Para tratar o problema mencionado, visando diminuir o espaço de busca de literais na adição de antecedentes, e consequentemente reduzir o tempo de execução do processo de geração de literais, no meu mestrado propusemos o sistema de revisão

FORTE_MBC [10]. O FORTE_MBC surgiu a partir do FORTE pela modificação do operador de refinamento de Adição de Antecedentes, mantendo o mesmo algoritmo principal do FORTE (3.1). FORTE_MBC passou a utilizar a cláusula mais específica (*Bottom Clause*), para limitar o espaço de busca de antecedentes a serem adicionados a uma cláusula, e também introduziu a declaração de modos para validar as cláusulas geradas na revisão. Ao introduzir a *Bottom Clause* o FORTE_MBC reduziu consideravelmente o espaço de busca de cláusulas, que por sua vez reduziu o espaço de busca de teorias, o que se mostrou essencial para reduzir o tempo de execução do processo de revisão em uma média de 55 vezes [10]. Além disso, ele gera teorias mais compreensíveis e não diminui significativamente a acurácia obtida pelo processo de revisão original, mantendo as acurácias mais elevadas quando comparado ao aprendizado do zero, assim como o sistema FORTE original.

As modificações introduzidas no FORTE_MBC afetaram alguns operadores de refinamento de teorias do FORTE, tal como o Adição de Antecedentes, o Exclusão de Antecedentes e o Adição de Regras. O operador de Exclusão de Regras não foi afetado. Como uma das contribuições desta Tese é a caracterização formal dos operadores de refinamento do FORTE_MBC, nesta seção descreveremos, em detalhes, estes operadores (Seções 3.2 e 3.3), mostramos um exemplo prático da aplicação destes no FORTE_MBC, dada uma teoria corrente. A seguir citaremos os operadores de teorias e cláusulas utilizados no FORTE_MBC, dando ênfase aos operadores afetados pela introdução da *Bottom Clause*, e detalharemos o algoritmo de Adição de Antecedentes, pois este será alterado por uma das propostas desta Tese, o algoritmo FORTE_BETH, a ser explicado no Capítulo 5. Os operadores do FORTE_MBC serão formalmente definidos no Capítulo 4.

3.2 Operadores de Especialização do FORTE_-MBC

A especialização de teorias no FORTE_MBC segue duas abordagens. A primeira corresponde a simplesmente excluir a cláusula incorreta usando o operador de refinamento de teorias de Exclusão de Regras. Suponha, por exemplo, a teoria corrente T abaixo. Neste exemplo estamos considerando o domínio do Alzheimer [14], e as declarações de modos para este domínio estão definidos no **Apêndice B**.

$$
\begin{aligned}
T : \overrightarrow{C_1} &= great\_ne(A,B) \leftarrow ring\_subst\_3(B,C), ring\_substitutions(A,D),\\
&\quad ring\_substitutions(B,E), gt(E,D), alk\_groups(A,E).\\
\overrightarrow{C_2} &= great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\
&\quad r\_subst\_2(A,E).\\
\overrightarrow{C_3} &= great\_ne(A,B) \leftarrow ring\_substitutions(A,C), gt(C,D),
\end{aligned}
$$

$ring\_subst\_2(A, E)$.

Seja $\overrightarrow{C_2} \in T$ um ponto de revisão de especialização. Aplicando-se o operador de Exclusão de Regras, o FORTE_MBC simplesmente exclui a cláusula $\overrightarrow{C_2}$ de T, obtendo a teoria $T_1 = \{\overrightarrow{C_1}, \overrightarrow{C_3}\}$.

A segunda abordagem corresponde a especializar uma cláusula incorreta através do operador de refinamento de teorias de Adição de Antecedentes, tal que a cláusula incorreta é excluída da teoria e uma ou mais especializações da mesma são adicionadas na teoria. Cada uma destas especializações é criada aplicando-se um operador de refinamento de cláusulas à cláusula incorreta. Tal operador, o qual chamaremos de operador de refinamento de cláusulas de Adição de Antecedentes, adiciona um ou mais literais à cláusula sendo revisada, tal que os literais são colocados mais à direita na cláusula. Tais antecedentes pertencem a uma *Bottom Clause* [24] criada a partir de um exemplo positivo provado pela cláusula corrente. Literais podem ser selecionados da *Bottom Clause* em qualquer ordem, desde que a cláusula resultante obedeça às declarações de modos e às restrições impostas pelo especialista.

A busca pelos antecedentes a serem adicionados a uma cláusula pode ser feita usando os algoritmos *Hill Climbing* e *Relational Pathfinding* [38]. Dependendo do método de busca escolhido, a pontuação da nova cláusula é calculada a cada antecedente adicionado (método *Hill Climbing*), escolhendo o antecedente com a melhor pontuação, ou calculada após um conjunto de antecedentes serem adicionados (método *Relational Pathfinding*). O algoritmo *Hill Climbing* de Adição de Antecedentes é descrito em Algoritmo 3.2. Mais detalhes sobre estes algoritmos podem ser achados em [10]. A busca para de adicionar antecedentes quando não há mais antecedentes para serem adicionados, o tamanho máximo permitido para a cláusula foi alcançado, ou não existem mais exemplos negativos sendo provados pela cláusula especializada.

A fim de criar a *Bottom Clause* explorada pelo operador de refinamento de cláusulas de Adição de Antecedentes, o FORTE_MBC começa a partir dos literais da cláusula corrente sob revisão, os quais serão os literais iniciais da *Bottom Clause*. Depois disso é aplicado o método de saturação do sistema Aleph [41], conforme explicado na Seção 2.3.3. O Algoritmo de Geração da *Bottom Clause* no FORTE_MBC é detalhado em A.1, no **Apêndice A**. Ao escolher um literal da *Bottom Clause* para adicionar na cláusula, o operador de refinamento de cláusulas de Adição de Antecedentes considera o literal exatamente como ele está na *Bottom Clause*, como é feito de forma padrão no Aleph, isto é, não é aplicada nenhuma substituição θ antes de adicionar o literal à cláusula, como é feito no Progol [24].

Para ilustrar a aplicação do operador de refinamento de teorias de Adição de Antecedentes e, consequentemente, do operador de refinamento de cláusulas de Adição

Algoritmo 3.2 Algoritmo de Adição de Antecedentes *hill climbing* do FORTE_-MBC

T = Teoria corrente
$\overrightarrow{C}$ = Cláusula corrente a ser revisada
CP = Conjunto de exemplos positivos provados pela cláusula
IP = Conjunto de exemplos negativos provados pela cláusula
$\overrightarrow{C\_original} = \overrightarrow{C}$
$CP\_original = CP$
$IP\_original = IP$

1: **repita**
2: Considere o primeiro exemplo I de CP
3: Gere a *bottom clause* $\overrightarrow{\perp}$ a partir de $\overrightarrow{C}$ e I
4: **repita**
5: Calcule a pontuação de $\overrightarrow{C}$ a partir de CP e IP
6: Retorne os possíveis antecedentes utilizando $\overrightarrow{\perp}$
7: Calcule as pontuações dos antecedentes retornados
8: **se** o antecedente com a melhor pontuação melhora a pontuação de $\overrightarrow{C}$ ao ser adicionado **então**
9: Adicione o antecedente à cláusula $\overrightarrow{C}$, formando a cláusula $\overrightarrow{C'}$
10: $NovoCP$ = Exemplos positivos provados por $\overrightarrow{C'}$
11: $NovoIP$ = Exemplos negativos provados por $\overrightarrow{C'}$
12: $\overrightarrow{C} = \overrightarrow{C'}$
13: $CP = NovoCP$
14: $IP = NovoIP$
15: **até que** não se tenha mais antecedentes, ou o tamanho máximo permitido para a cláusula tenha sido atingido, ou a pontuação da cláusula não seja mais melhorada.
16: **se** a nova cláusula $\overrightarrow{C'}$ é diferente da cláusula original $\overrightarrow{C\_original}$ **então**
17: Acrescente a nova cláusula $\overrightarrow{C'}$ à teoria T, formando a nova teoria T'
18: Calcule o novo conjunto de exemplos positivos provados ($NovoCP\_Teoria$) e exemplos negativos provados ($NovoIP\_Teoria$) pela nova teoria
19: $\overrightarrow{C} = \overrightarrow{C\_original}$
20: $CP = CP\_original - NovoCP\_Teoria$
21: $IP = IP\_original$
22: $T = T'$
23: **senão**
24: escolha um novo exemplo positivo ainda não usado de CP e recomeçe o algoritmo a partir da geração de uma nova *bottom clause*
25: **até que** não consiga mais especializar a cláusula, ou até que todos os exemplos positivos provados pela cláusula original tenham sido provados pelas cláusulas criadas

de Antecedentes, considere a mesma teoria T dada anteriormente e o mesmo ponto de especialização $\overrightarrow{C_2}$. Para especializar T, o operador de refinamento de teorias de Adição de Antecedentes irá gerar uma ou mais especializações de $\overrightarrow{C_2}$, onde cada especialização é gerada a partir do operador de cláusulas de Adição de Antecedentes,

utlilizando-se exemplos positivos e *Bottom Clauses* diferentes criadas a partir destes exemplos. Seja $great\_ne(p1, b1)$ o exemplo usado para criar a *Bottom Clause* para a primeira especialização da cláusula corrente $\overrightarrow{C_2}$. Considerando o limite de profundidade de variável i=2 e começando-se a partir dos literais de $\overrightarrow{C_2}$, a *Bottom Clause* criada é:

$$\begin{aligned}\overrightarrow{\perp} = great\_ne(A,B) \leftarrow\ & alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\ & r\_subst\_2(A,E), x\_subst(B,F,G), r\_subst\_1(B,H), r\_subst\_1(A,I),\\ & ring\_substitutions(A,D), polar(G,J), size(G,K), flex(G,L),\\ & h\_doner(G,M), h\_acceptor(G,N), pi\_doner(G,O), pi\_acceptor(G,P),\\ & polarisable(G,Q), sigma(G,R).\end{aligned}$$

O operador de refinamento de cláusulas de Adição de Antecedentes pode usar qualquer subconjunto desta *Bottom Clause* como especialização de $\overrightarrow{C_2}$, se a cláusula gerada obedecer aos modos definidos. Exemplos de tais subconjuntos são:

$$\begin{aligned}\overrightarrow{D_1} = great\_ne(A,B) \leftarrow\ & alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\ & r\_subst\_2(A,E), x\_subst(B,F,G), ring\_substitutions(A,D).\\ \overrightarrow{D_2} = great\_ne(A,B) \leftarrow\ & alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\ & r\_subst\_2(A,E), ring\_subst\_1(A,I), r\_subst\_1(B,H).\end{aligned}$$

Observe que em $\overrightarrow{D_2}$ os antecedentes $ring\_subst\_1(A,I)$ e $r\_subst\_1(B,H)$ adicionados não estão na mesma ordem em que aparecem na *Bottom Clause*.

O operador de refinamento de teorias de Adição de Antecedentes pode ainda adicionar outras especializações de $\overrightarrow{C_2}$. Por exemplo, a cláusula $\overrightarrow{D_3}$ pode ser criada a partir da *Bottom Clause* $\overrightarrow{\perp_1}$ gerada a partir do exemplo $great\_ne(m1, a1)$, como se segue:

$$\begin{aligned}\overrightarrow{\perp_1} = great\_ne(A,B) \leftarrow\ & alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\ & r\_subst\_2(A,E), r\_subst\_1(B,F), r\_subst\_1(A,G), r\_subst\_3(A,H),\\ & ring\_substitutions(A,D), gt(C,I), gt(C,J), gt(C,K), gt(K,D), gt(K,I),\\ & gt(K,J), gt(J,D), gt(J,I), gt(I,D).\end{aligned}$$

$$\begin{aligned}\overrightarrow{D_3} = great\_ne(A,B) \leftarrow\ & alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\ & r\_subst\_2(A,E), ring\_substitutions(A,D).\end{aligned}$$

Cada uma das especializações é criada a partir da cláusula original marcada como ponto de revisão e a partir de uma *Bottom Clause* diferente. A busca para de criar novas especializações quando não é mais possível especializar a cláusula original, ou todos os exemplos positivos provados pela cláusula original também são provados pelas cláusulas geradas. Para este exemplo, a teoria $T_2 = \{\overrightarrow{D_1}, \overrightarrow{D_3}, \overrightarrow{C_1}, \overrightarrow{C_3}\}$

é uma possível teoria resultante da aplicação do operador de refinamento de teorias de Adição de Antecedentes em T. Ela contém as especializações $\overrightarrow{D_1}$ e $\overrightarrow{D_3}$ de $\overrightarrow{C_2}$, sendo que $\overrightarrow{C_2}$ é removida da teoria.

A Definição 21 a seguir resume que tipo de teoria pode ser gerada no FORTE_-MBC aplicando-se os operadores de refinamento de especialização de teorias, dada uma teoria corrente T:

Definição 21. Seja $T = \{\overrightarrow{C_1}, \overrightarrow{C_2}, ..., \overrightarrow{C_n}\}$ a teoria corrente a ser revisada. Aplicando-se operadores de refinamento de especialização, o FORTE_MBC é capaz de gerar qualquer teoria $T' = \{\overrightarrow{D_1}, \overrightarrow{D_2}, ..., \overrightarrow{D_m}\}$ tal que $T' \neq T$ e tal que, para cada cláusula $\overrightarrow{D_j} \in T'$, $1 \leq j \leq m$, um de dois casos acontece:

- $\overrightarrow{D_j} \in T$.
- $\overrightarrow{D_j} \subseteq \overrightarrow{\perp_k}$, onde $\overrightarrow{\perp_k} \in L_i(M)$ é uma *Bottom Clause* criada a partir de alguma cláusula $\overrightarrow{C_k} \in T$, $1 \leq k \leq n$, e a partir de um exemplo positivo provado por ela, e tal que $\overrightarrow{C_k} \notin T'$ (Aplicando-se o operador de Adição de Antecedentes).

3.3 Operadores de Generalização do FORTE_MBC

A generalização de teorias no FORTE_MBC segue três abordagens. A primeira corresponde a generalizar uma cláusula incorreta através do operador de refinamento de teorias de Exclusão de Antecedentes, tal que a cláusula incorreta é excluída da teoria e uma ou mais generalizações da mesma são adicionadas à teoria. Cada uma destas generalizações é criada a partir da cláusula original aplicando-se um operador de refinamento de cláusulas à cláusula incorreta. Tal operador, o qual chamaremos de operador de refinamento de cláusulas de Exclusão de Antecedentes, exclui um ou mais antecedentes da cláusula incorreta se a cláusula resultante obedecer aos modos definidos. A pontuação da nova cláusula é calculada a cada novo antecedente excluído, escolhendo-se o antecedente com a melhor pontuação, ou é calculada após um conjunto de antecedentes serem excluídos, dependendo do método de busca considerado. Antecedentes são removidos até que todos os exemplos positivos sejam provados pela cláusula corrente ou a pontuação da cláusula não seja mais melhorada. A busca para de criar novas generalizações quando não é mais possível melhorar a pontuação da cláusula original, ou todos os exemplos positivos não provados por esta já foram provados pelas cláusulas geradas.

Para ilustrar a aplicação do operador de refinamento de teorias de Exclusão de Antecedentes e, consequentemente, do operador de refinamento de cláusulas de Exclusão de Antecedentes, considere a mesma teoria T dada na seção anterior, e seja $\overrightarrow{C_1} \in T$ um ponto de revisão de generalização. Para generalizar T, o operador de

refinamento de teorias de Exclusão de Antecedentes irá gerar uma ou mais generalizações de $\overrightarrow{C_1}$, onde cada generalização é gerada a partir do operador de cláusulas de Exclusão de Antecedentes. Aplicando-se o operador de refinamento de cláusulas de Exclusão de Antecedentes, o FORTE_MBC pode usar qualquer subsequência de $\overrightarrow{C_1}$ como uma possível generalização, se esta obedecer aos modos definidos. Exemplos de tais subsequências são:

$$\overrightarrow{D_1} = great\_ne(A, B) \leftarrow ring\_substitutions(A, D), ring\_substitutions(B, E), gt(E, D), alk\_groups(A, E).$$
$$\overrightarrow{D_2} = great\_ne(A, B) \leftarrow ring\_subst\_3(B, C), ring\_substitutions(B, E), gt(E, D).$$

Observe que a cláusula $\overrightarrow{D_3}$ abaixo não poderia ser gerada pelo operador de refinamento de cláusulas de Exclusão de Antecedentes, já que a mesma não obedece aos modos:

$$\overrightarrow{D_3} = great\_ne(A, B) \leftarrow ring\_subst\_3(B, C), ring\_substitutions(A, D), gt(E, D), alk\_groups(A, E).$$

Em $\overrightarrow{D_3}$, o literal $gt(E, D)$ tem a variável E como entrada, mas ela não é gerada como variável de saída em nenhum literal da cláusula vindo antes de $gt(E, D)$.

A teoria $T_3 = \{\overrightarrow{D_1}, \overrightarrow{C_2}, \overrightarrow{C_3}\}$ é portanto um exemplo de uma teoria gerada pela aplicação do operador de refinamento de teorias de Exclusão de Antecedentes em T, removendo-se a cláusula $\overrightarrow{C_1}$ de T e adicionando-se a cláusula $\overrightarrow{D_1}$ acima. Observe que, neste exemplo, o operador de refinamento de teorias de Exclusão de Antecedentes adicionou uma única generalização da cláusula $\overrightarrow{C_1}$, mas poderia ter adicionado mais.

A segunda abordagem de generalização do FORTE_MBC corresponde a aplicar o operador de refinamento de teorias de Adição de Regras. Este começa a partir de uma cópia da cláusula incorreta e cria uma cláusula generalizada a partir desta, excluindo-se os antecedentes da mesma forma que é feita pelo operador de refinamento de cláusulas de Exclusão de Antecedentes. Tal cláusula generalizada também deve obedecer aos modos. Se a nova cláusula estiver provando exemplos negativos então o operador de refinamento de teorias de Adição de Antecedentes é aplicado para gerar uma ou mais especializações desta cláusula. A cláusula incorreta original é mantida na teoria. É importante ressaltar que, no operador de Adição de Regras, a Adição de Antecedentes só será aplicada se ao menos um antecedente tiver sido removido da cláusula original.

Como exemplo, suponha que o operador de Adição de Regras é aplicado à cláusula $\overrightarrow{C_1}$ ao invés do operador de refinamento de teorias de Exclusão de Antecedentes. Após remover antecedentes de $\overrightarrow{C_1}$, utilizando-se o operador de refinamento de cláusulas de Exclusão de Antecedentes, uma possível cláusula generalizada

é $\overrightarrow{D_4} = great\_ne(A, B) \leftarrow alk\_groups(A, E)$. Renomeando-se a variável E para C em $\overrightarrow{D_4}$, de forma a manter a ordem alfabética crescente das variáveis, a nova cláusula é passada como cláusula corrente para o operador de refinamento de teorias de Adição de Antecedentes.

A partir de exemplos positivos diferentes provados por $\overrightarrow{D_4}$ e de *Bottom Clauses* geradas a partir deles, duas possíveis especializações criadas são:

$$\overrightarrow{D_5} = great\_ne(A, B) \leftarrow alk\_groups(A, C), r\_subst\_2(A, D).$$
$$\overrightarrow{D_6} = great\_ne(A, B) \leftarrow alk\_groups(A, C), r\_subst\_2(A, D),$$
$$ring\_subst\_3(A, E), ring\_substitutions(A, C), ring\_subst\_3(B, F).$$

Portanto, a teoria $T_4 = \{\overrightarrow{D_5}, \overrightarrow{D_6}, \overrightarrow{C_1}, \overrightarrow{C_2}, \overrightarrow{C_3}\}$ é um exemplo de uma teoria gerada pela aplicação do operador de Adição de Regras a T, mantendo-se a cláusula $\overrightarrow{C_1}$ em T e adicionando-se as cláusulas $\overrightarrow{D_5}$ e $\overrightarrow{D_6}$ acima.

O número de cláusulas especializadas que podem ser inseridas na teoria depende do método de busca usado. Para este exemplo, a teoria que apenas considera a inclusão da cláusula $\overrightarrow{D_6}$ em T também está no grafo de refinamento. Ambas as teorias podem ser geradas pelo operador, mas não necessariamente as duas serão criadas durante a busca por uma teoria revisada.

Finalmente, a terceira abordagem de generalização do FORTE_MBC corresponde a aplicar o operador de refinamento de teorias de Adição de Novas Regras, que insere uma cláusula inteiramente nova na teoria. O operador parte de uma cláusula com corpo vazio e a partir dela aplica o operador de refinamento de teorias de Adição de Antecedentes. O Adição de Novas Regras é aplicado quando um determinado exemplo positivo não é provado pela teoria e não existe nenhuma cláusula na teoria que tenha na cabeça o mesmo predicado do exemplo.

A Definição 22 a seguir resume que tipo de teoria pode ser gerada no FORTE_MBC pela aplicação de operadores de refinamento de generalização de teorias, dada uma teoria corrente T:

Definição 22. Seja $T = \{\overrightarrow{C_1}, \overrightarrow{C_2}, ..., \overrightarrow{C_n}\}$ a teoria corrente a ser revisada. Aplicando-se operadores de refinamento de generalização, o FORTE_MBC é capaz de gerar qualquer teoria $T' = \{\overrightarrow{D_1}, \overrightarrow{D_2}, ..., \overrightarrow{D_m}\}$ tal que $T' \neq T$ e tal que para cada cláusula $\overrightarrow{D_j} \in T'$, $1 \leq j \leq m$, um de quatro casos acontece:

- $\overrightarrow{D_j} \in T$.
- $\overrightarrow{D_j}$ é subsequência de $\overrightarrow{C_k} \in T$, $1 \leq k \leq n$, tal que $\overrightarrow{C_k} \notin T'$, e $\overrightarrow{D_j} \in L_i(M)$ (Aplicando-se o operador de Exclusão de Antecedentes).
- $\overrightarrow{D_j} \subseteq \overrightarrow{\perp}$, onde $\overrightarrow{\perp} \in L_i(M)$ é uma *Bottom Clause* a partir de alguma cláusula $\overrightarrow{D'}$ e a partir de um exemplo positivo provado por esta, tal que $\overrightarrow{D'} \in L_i(M)$ é

uma subsequência de $\overrightarrow{C_k} \in T$, $1 \leq k \leq n$, sendo que $\overrightarrow{C_k}$ também está em T' (Aplicando-se o operador de Adição de Regras).

- $\overrightarrow{D_j} \subseteq \overrightarrow{\perp}$, onde $\overrightarrow{\perp} \in L_i(M)$ é uma *Bottom Clause* a partir de um exemplo positivo não provado por T, e tal que o predicado da cabeça de $\overrightarrow{D_j}$ não aparece na cabeça de nenhuma cláusula em T (aplicando-se o operador de Adição de Novas Regras).

Capítulo 4

Caracterização Formal do Espaço de Busca e Operadores de Refinamento do FORTE_MBC

4.1 Introdução

Ao introduzir a *Bottom Clause* e declaração de modos para limitar o espaço de busca, o FORTE_MBC reduziu consideravelmente o espaço de busca de cláusulas, que por sua vez reduziu o espaço de busca de teorias, o que se mostrou essencial para reduzir o tempo de execução do processo de revisão em uma média de 55 vezes [10]. Além disso, ele gera teorias mais compreensíveis e não diminui significativamente a acurácia obtida pelo processo de revisão original, mantendo as acurácias mais elevadas quando comparado ao aprendizado do zero, assim como o sistema FORTE original [10].

No entanto, a eficácia e a eficiência do aprendizado como um processo de refinamento depende fortemente de outros fatores, tais como as propriedades do espaço de busca e, como consequência, dos operadores de refinamento [13]. Tradicionalmente, em ILP, os sistemas são definidos através de uma caracterização formal de seu espaço de busca e dos seus operadores de refinamento, assim algoritmos e operadores mais eficientes podem ser projetados [4, 11, 13, 20, 42]. O FORTE_MBC já melhorou a eficiência do FORTE, mas seu espaço de busca e operadores de refinamento não foram devidamente caracterizados, até o momento.

Buscar num espaço de hipóteses limitado por uma cláusula mais específica é a base de sistemas conhecidos de ILP, como Progol [24] e Aleph [41]. Em [42], tendo o sistema Progol como instância, a estrutura e as propriedades de tal espaço

de hipóteses é caracterizado pela introdução da *subsumption order* relativa a uma *Bottom Clause*. Porém, tal ordem de generalidade não caracteriza propriamente o espaço de busca do FORTE_MBC, como será visto na Seção 4.5. Além disso, o Progol se restringe a um espaço de cláusulas, e lida apenas com aprendizado a partir do zero, executando uma busca descendente, de forma que apenas operadores de refinamento de especialização são considerados.

Por outro lado, Midelfart [20] introduz uma extensão de *subsumption* em teorias que leva em consideração uma teoria mais específica, chamada de Teoria *Bottom*, que é uma teoria formada por *Bottom Clauses*. Porém, os resultados teóricos apresentados em [20] não se aplicam ao espaço de teorias do FORTE_MBC (Seção 4.5), e, como em [42], também só consideram um espaço de busca de especialização.

Neste capítulo iremos caracterizar formalmente a estrutura de reticulado do espaço de busca do FORTE_MBC, e também iremos estudar as propriedades dos seus operadores de refinamento, as quais foram introduzidas na Definição 14 do Capítulo 2.

O FORTE_MBC revisa teorias usando operadores de refinamento de teorias, alguns dos quais usam operadores de refinamento de cláusulas. Com exceção do operador de Exclusão de Regras, todos os outros operadores do FORTE_MBC usam, internamente, um operador de refinamento de cláusulas para especializar ou generalizar uma cláusula. A introdução de uma *Bottom Clause* para limitar o espaço de busca de especialização de cláusulas afetou o espaço de busca de um conjunto de operadores de refinamento de cláusulas. Porém, a mudança no espaço de cláusulas também leva a uma mudança no espaço de teorias, pelo menos no espaço em que os operadores de refinamento de teorias que usam os operadores de refinamento de cláusulas modificados atuam.

Já que temos tanto operadores de teoria quanto de cláusulas, podemos pensar em dois reticulados diferentes: um de teorias e outro de cláusulas, de acordo com alguma linguagem e estruturados por alguma ordem de generalidade. Teorias no reticulado de teorias são conjunto de cláusulas que estão no reticulado de cláusulas. Também, de acordo com a Definição 13 de *subsumption* em teorias, a ordem de generalidade que estrutura o reticulado de teorias deve incluir em sua definição a ordem de generalidade usada para estruturar o reticulado de cláusulas.

Como normalmente é feito, para caracterizar formalmente o espaço de busca e operadores de refinamento do FORTE_MBC seguiremos os passos a seguir:

- Definir a linguagem G e a ordem de generalidade R.
- Mostrar que R é uma quasi-ordem e que o par $\langle G, R\rangle$ é um reticulado.
- Definir formalmente os operadores e seus refinamentos.

- Caracterizar os operadores de acordo com as propriedades dadas na Definição 14.

A seguir, na Seção 4.2 caracterizamos o espaço de busca de cláusulas do FORTE_MBC, assim como os operadores de refinamento de generalização e especialização de cláusulas. Para isso seguimos os passos dados acima, definindo a linguagem de cláusulas considerada pelo FORTE_MBC e a ordem de generalidade que estrutura o espaço de busca, que leva em consideração uma *Bottom Clause*. Então, os operadores de refinamento de cláusulas são formalmente definidos e caracterizados. Na Seção 4.3 o mesmo é feito mas considerando o espaço de busca e operadores de refinamento de teorias do FORTE_MBC. Em seguida, na Seção 4.4 propomos operadores de refinamento ideais para FORTE_MBC. Na Seção 4.5 apresentamos alguns trabalhos relacionados citados anteriormente ([42] e [20]).

4.2 Espaço de busca e operadores de refinamento de cláusulas do FORTE_MBC

No FORTE_MBC existem dois operadores de refinamento de cláusulas:

- **Exclusão de Antecedentes**: gera uma única cláusula generalizada quando o operador de revisão de teorias de Exclusão de Antecedentes é aplicado.
- **Adição de Antecedentes**: gera uma única cláusula especializada quando o operador de revisão de Adição de Antecedentes é aplicado.

Com relação ao operador de refinamento de cláusulas de Adição de Antecedentes, as possíveis cláusulas que este pode gerar são as que partem da cláusula corrente sob revisão e possuem literals adicionados de uma *Bottom Clause* correspondente. Como os literais vindos da *Bottom Clause* podem ser selecionados em qualquer ordem, desde que respeitem aos modos definidos, tais cláusulas são subconjuntos ordenados da *Bottom Clause*, de acordo com a Definição 19 de *subconjunto ordenado*. No caso do operador de refinamento de cláusulas de Exclusão de Antecedentes, as possíveis cláusulas criadas são subsequências da cláusula corrente, e também devem obedecer aos modos definidos. Por suposição, todas as cláusulas na teoria corrente são relevantes [20], i.e., elas explicam exemplos diretamente, de forma que cada cláusula é subconjunto de alguma *Bottom Clause* da linguagem, e portanto as cláusulas criadas pelo operador de refinamento de cláusulas de Exclusão de Antecedentes também são subconjuntos ordenados de alguma *Bottom Clause* (neste caso elas são estritamente subsequências), mesmo que a *Bottom Clause* não seja explicitamente gerada. Como substituições θ não são aplicadas, o espaço de cláusulas do FORTE_MBC é um espaço proposicional, e, para caracterizá-lo, introduzimos uma ordem de generalidade

chamada **Subsumption proposicional relativa a uma Bottom Clause**. Esta ordem de generalidade é baseada na *Subsumption* proposicional da Definição 11 do Capítulo 2, mas alterada para ser relativa a uma *Bottom Clause*, assim como em [42]. A linguagem e a ordem de generalidade do espaço de cláusulas do FORTE_MBC são definidas a seguir:

Definição 23 ($L_\perp(M)$)**.** Seja $\overrightarrow{\perp} \in L_i(M)$ uma *Bottom Clause* e $\overrightarrow{C}$ uma cláusula ordenada e definida livre de função. $\overrightarrow{C}$ está em $L_\perp(M)$ se e somente se $\overrightarrow{C} \in L_i(M)$ e $\overrightarrow{C}$ é um subconjunto ordenado de $\overrightarrow{\perp}$.

Definição 24 (*Subsumption* Proposicional relativa a uma *Bottom Clause*)**.** Seja $\overrightarrow{\perp}$ e $L_\perp(M)$ como definidos anteriormente, $\overrightarrow{C}, \overrightarrow{D} \in L_\perp(M)$ duas cláusulas ordenadas e definidas. Dizemos que $\overrightarrow{C}$ subsume proposicionalmente $\overrightarrow{D}$ relativo a uma *Bottom Clause* $\overrightarrow{\perp}$, denotado por $\overrightarrow{C} \succeq_\perp \overrightarrow{D}$, se $\overrightarrow{C} \subseteq \overrightarrow{D}$ (a menos de renomeação de variáveis). $\overrightarrow{C}$ e $\overrightarrow{D}$ são **subsume-equivalentes** ($\overrightarrow{C} \sim \overrightarrow{D}$) se $\overrightarrow{C} \succeq_\perp \overrightarrow{D}$ e $\overrightarrow{D} \succeq_\perp \overrightarrow{C}$.

Teorema 1. $\langle L_\perp(M), \succeq_\perp \rangle$ *é um* **reticulado**.

Demonstração. Para mostrar que $\langle L_\perp(M), \succeq_\perp \rangle$ é um reticulado, é suficiente notar que para toda cláusula $\overrightarrow{C}, \overrightarrow{D} \in L_\perp(M)$, o **lub** é dado pela interseção das cláusulas $\overrightarrow{C}$ e $\overrightarrow{D}$ ($\overrightarrow{C} \sqcup \overrightarrow{D} = \overrightarrow{C} \cap \overrightarrow{D}$), enquanto que o **glb** é dado pela união das cláusulas $\overrightarrow{C}$ e $\overrightarrow{D}$ ($\overrightarrow{C} \sqcap \overrightarrow{D} = \overrightarrow{C} \cup \overrightarrow{D}$, desconsiderando literais repetidos e colocando os literais de $\overrightarrow{C}$ e $\overrightarrow{D}$ na ordem em que aparecem na $\overrightarrow{\perp}$, para garantir a conformidade com os modos). As cláusulas dadas por $\overrightarrow{C} \cap \overrightarrow{D}$ e $\overrightarrow{C} \cup \overrightarrow{D}$ são ambas subconjuntos ordenados de $\overrightarrow{\perp}$, e portanto uma generalização mínima e uma especialização máxima de duas cláusulas em $L_\perp(M)$ existem. Também podemos afirmar que a relação de *subsumption* proposicional relativa a uma *Bottom Clause* é uma quasi-ordem, já que esta é baseada no relação de subconjunto, que por sua vez é reflexiva e transitiva. □

Ambos os operadores de refinamento de cláusulas do FORTE_MBC buscam neste reticulado, porém a busca pode começar de qualquer ponto do reticulado, dependendo da cláusula corrente sendo revisada, como mostrado na Figura 4.1. O reticulado é limitado inferiormente por uma *Bottom Clause* e superiormente pela cláusula contendo apenas a cabeça da regra, representada por □ na figura.

Tendo caracterizado a estrutura de reticulado do espaço de cláusulas do FORTE_MBC, procedemos para a definição formal dos operadores de refinamento de cláusulas, começando-se pelo Adição de Antecedentes.

Definição 25 (Operador de especialização de cláusulas de Adição de Antecedentes ρ_h)**.** Seja $\overrightarrow{\perp}$ e $L_\perp(M)$ como definidos anteriormente, e $\overrightarrow{C}$ e $\overrightarrow{C'}$ duas cláusulas ordenadas e definidas em $L_\perp(M)$. $\langle \overrightarrow{C'}, \overrightarrow{\perp'} \rangle$ pertence a $\rho_h(\langle \overrightarrow{C}, \overrightarrow{\perp} \rangle)$ se e somente se:
$\overrightarrow{C'} = \overrightarrow{C} \cup \{l\}$, onde $l \in \overrightarrow{\perp}$, e $\overrightarrow{\perp'} = \overrightarrow{\perp} - \{l\}$.

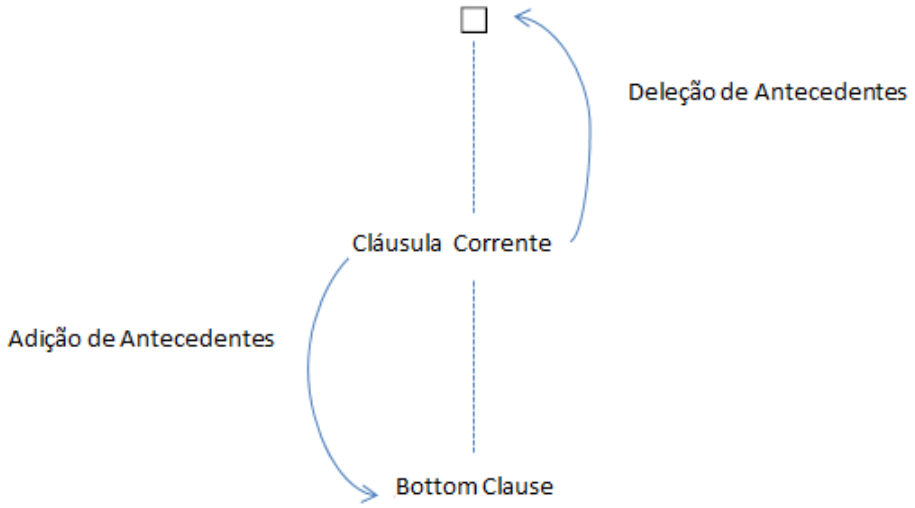

Figura 4.1: Reticulado de cláusulas e espaço de atuação dos operadores

Este operador é ideal para o reticulado definido, como é provado a seguir:

Teorema 2. *No reticulado* $\langle L_\perp(M), \succeq_\perp \rangle$*, o operador de refinamento* ρ_h *é ideal.*

Demonstração.

(*localmente finito*) Como a *Bottom Clause* usada como espaço de busca de literais é uma cláusula livre de função e é limitada por uma profundidade de variável i, ela é finita. Portanto o operador ρ_h é **localmente finito**.

(*próprio*) Para um operador de refinamento ser próprio ele deve computar apenas especializações próprias de uma cláusula, para não ficar preso numa sequência de cláusulas equivalentes após repetidas aplicações do operador. Como o operador ρ_h não duplica literais vindos da *Bottom Clause* e não aplica substituições de variáveis, ele não é capaz de gerar cláusulas equivalentes a partir da cláusula corrente. Logo, o operador ρ_h é **próprio**.

(*completo*) Sejam $\overrightarrow{C}$ e $\overrightarrow{D}$ duas cláusulas em $L_\perp(M)$, tal que $\overrightarrow{C}$ e $\overrightarrow{D}$ não são equivalentes. Para o operador ρ_h ser completo, cada especialização deve ser alcançada por um número finito de aplicações do operador, i.e., existe um caminho de tamanho finito a partir de $\overrightarrow{C}$ até um membro da classe de equivalência de $\overrightarrow{D}$ quando $\overrightarrow{C} \succ_\perp \overrightarrow{D}$. Se $\overrightarrow{C} \succ_\perp \overrightarrow{D}$, i.e., $\overrightarrow{C}$ é um subconjunto ordenado de $\overrightarrow{D}$ e ambos são subconjuntos ordenados de $\overrightarrow{\perp}$, então os literais em $\overrightarrow{D} - \overrightarrow{C}$ estão também em $\overrightarrow{\perp}$, e portanto o operador ρ_h pode adicioná-los a $\overrightarrow{C}$ em um número finito de passos de refinamento, de forma a gerar $\overrightarrow{D}$ (ou uma cláusula equivalente a $\overrightarrow{D}$). Assim, ρ_h é **completo**.

Tendo provado que ρ_h é localmente finito, próprio e completo para $\langle L_\perp(M), \succeq_\perp \rangle$ provamos que ρ_h é **ideal** para $\langle L_\perp(M), \succeq_\perp \rangle$. □

Para sistemas ILP tais como Progol[24] e Aleph[41], ambos lidando com aprendizado do zero, é suficiente ter operadores de refinamento fracamente completos e de preferência não-redundantes. Isto é verdade porque, dado que a busca começa a partir do elemento mais geral $\square$, um operador fracamente completo é capaz de gerar todas as cláusulas que estão em uma determinada linguagem G definida para o reticulado, conforme Definição 14. A preferência por não-redundância é para evitar que a busca alcance a mesma cláusula por caminhos diferentes no grafo de refinamento, o que pode ser ruim no caso de uma busca completa. Porém, para algoritmos onde a busca pode começar de um ponto qualquer do espaço de busca, como no FORTE_-MBC, o ideal é ter operadores de refinamento completos, mesmo que redundantes [42], ou seja, se estamos partindo de uma cláusula $\overrightarrow{C}$ qualquer, e queremos que o operador gere uma cláusula $\overrightarrow{D}$, tal que $\overrightarrow{C} \succeq \overrightarrow{D}$, temos que garantir que existe um caminho de $\overrightarrow{C}$ a $\overrightarrow{D}$ no grafo de refinamento do operador, o que é garantido se o operador for completo, conforme Definição 14.

De acordo com [4, 31] um operador de refinamento não pode ser completo (requerimento para idealidade) e não-redundante (requerimento para otimalidade) ao mesmo tempo. Como o operador ρ_h é completo, ele também é **redundante**, já que a ordem dos literais na *Bottom Clause* não é respeitada, e portanto diferentes permutações da mesma cláusula podem ser geradas, desde que tais permutações obedecam aos modos definidos. A Figura 4.2 ilustra a redundância deste operador.

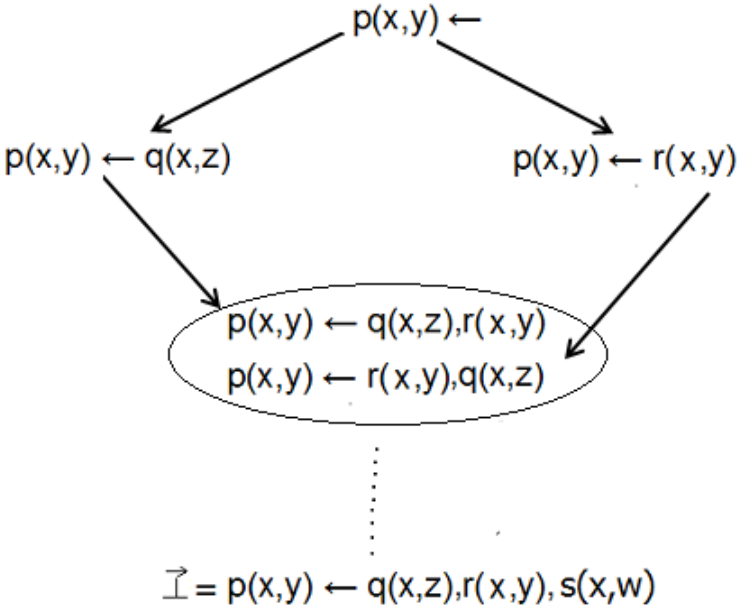

Figura 4.2: Exemplo de redundância do operador ρ_h

Suponha que, na figura 4.2, todos os predicados p, q e r possuem modos $(+,-)$. Para este exemplo e para os próximos exemplos ao longo deste capítulo, representaremos os modos apenas pela indicação se os seus argumentos são de entrada $(+)$ ou de saída $(-)$, assumindo que os tipos e o *recall* são respeitados na formação da cláusula.

A figura ilustra parte de um grafo de refinamento dada uma cláusula corrente $p(X,Y) \leftarrow$, e dada uma *Bottom Clause* criada a partir de um exemplo positivo provado por ela. Um possível refinamento especificado neste espaço de busca é a adição do literal $r(X,Y)$ à cláusula $p(X,Y) \leftarrow q(X,Z)$, gerando a cláusula $p(X,Y) \leftarrow q(X,Z), r(X,Y)$. Outro possível refinamento mostrado é a adição do literal $q(X,Z)$ a outra cláusula dada por $p(X,Y) \leftarrow r(X,Y)$, que também está no espaço definido, gerando a cláusula $p(X,Y) \leftarrow r(X,Y), q(X,Z)$. As duas cláusulas geradas em refinamentos distintos são equivalentes com respeito à $\succeq_\perp$, ou seja, aplicando-se o operador de refinamento de cláusulas de Adição de Antecedentes em cláusulas distintas do espaço, chegamos à cláusulas equivalentes por dois caminhos diferentes no grafo de refinamento, demonstrando a redundância do operador.

A seguir, procedemos para a definição formal do operador de generalização de cláusulas de Exclusão de Antecedentes.

Definição 26 (Operador de generalização de cláusulas de Exclusão de Antecedentes δ_h)**.** Seja $L_\perp(M)$ como definida anteriormente, e $\overrightarrow{C}$ e $\overrightarrow{C'}$ duas cláusulas ordenadas e definidas em $L_\perp(M)$. $\overrightarrow{C}'$ pertence a $\delta_h(\overrightarrow{C})$ se e somente se:
$\overrightarrow{C}' = \overrightarrow{C} - \{l\}$, onde $l \in \overrightarrow{C}$ e $\overrightarrow{C}'$ é uma subsequência de $\overrightarrow{C}$.

O operador δ_h é *localmente finito* e *próprio* para o reticulado $\langle L_\perp(M), \succeq_\perp \rangle$ definido, porém ele não é *completo* e portanto não é *ideal*, como provado pelos teoremas a seguir:

Teorema 3. *No reticulado $\langle L_\perp(M), \succeq_\perp \rangle$, o operador de refinamento δ_h é localmente finito e próprio.*

Demonstração.

(*localmente finito*) Como a cláusula corrente é finita, o operador δ_h é **localmente finito**.

(*próprio*) Dadas duas cláusulas $\overrightarrow{C}$ e $\overrightarrow{D}$, tal que $\delta_h(\overrightarrow{C}) = \overrightarrow{D}$, temos que $\overrightarrow{D} = \overrightarrow{C} - \{l\}$. Claramente $\overrightarrow{D}$ é subconjunto ordenado de $\overrightarrow{C}$, mas para $\overrightarrow{C}$ ser subconjunto ordenado de $\overrightarrow{D}$, mais de um literal de $\overrightarrow{C}$ teria que ser mapeado no mesmo literal de $\overrightarrow{D}$, i.e., $\overrightarrow{C}$ teria literais duplicados, o que não é permitido. Portanto o operador δ_h é **próprio**. □

Teorema 4. *No reticulado $\langle L_\perp(M), \succeq_\perp \rangle$, o operador de refinamento δ_h não é completo.*

Demonstração. A Figura 4.3 mostra parte do reticulado $\langle L_\perp(M), \succeq_\perp \rangle$, onde $\overrightarrow{C_1}$ é a cláusula corrente sendo revisada e $\overrightarrow{\perp}$ é uma *Bottom Clause* tal que $\overrightarrow{C_1}$ é um subconjunto ordenado de $\overrightarrow{\perp}$. Assuma modos $(+,+)$ e $(+,-)$ para os predicados p e q, respectivamente. Para cada par de cláusulas $\overrightarrow{C_i}$ e $\overrightarrow{C_j}$, se $\overrightarrow{C_i}$ está conectada a $\overrightarrow{C_j}$ por uma aresta, então $\overrightarrow{C_j} \succ_\perp \overrightarrow{C_i}$, para $i < j$. Arestas pontilhadas representam passos de refinamento que não são considerados pelo operador de Exclusão de Antecedentes δ_h.

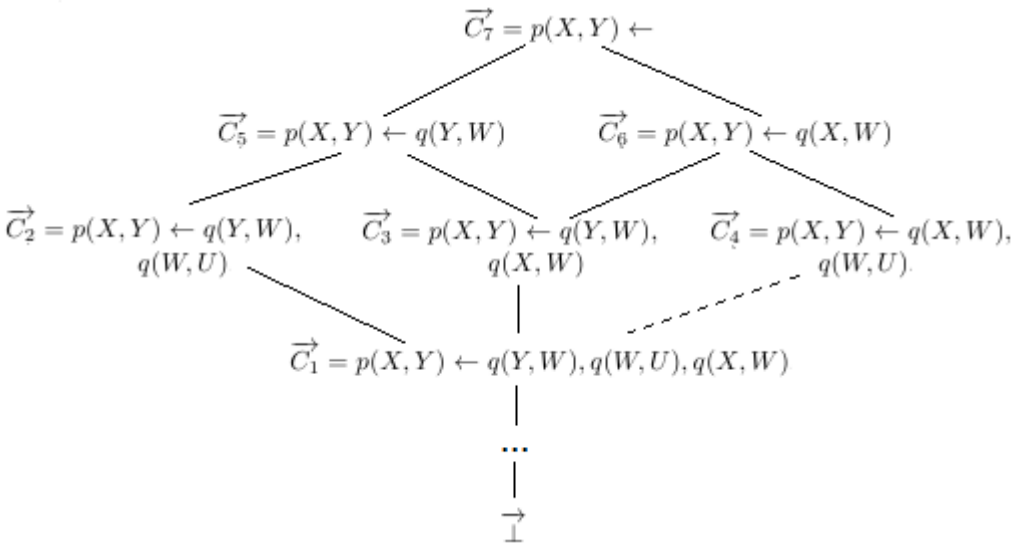

Figura 4.3: Parte do reticulado $\langle L_\perp(M), \succeq_\perp \rangle$, onde C_1 é a cláusula corrente.

Observe que a cláusula $\overrightarrow{C_4}$ está no reticulado pois $\overrightarrow{C_4} \succ_\perp \overrightarrow{C_1}$. Porém, $\delta_h(\overrightarrow{C_1})$ não consegue gerar $\overrightarrow{C_4}$, já que $\overrightarrow{C_4}$ não é subsequência de $\overrightarrow{C_1}$. Este exemplo portanto, é um contra-exemplo para a completude do operador δ_h. Apesar de não ser completo, o operador δ_h é **redundante**. Note que a cláusula $\overrightarrow{C_5}$, por exemplo, pode ser gerada a partir de dois caminhos diferentes no reticulado. □

4.2.1 Considerações a respeito dos operadores de refinamento de cláusulas

Vimos que ambos os operadores de refinamento de cláusulas de Adição de Antecedentes e Exclusão de Antecedentes são localmente finitos e próprios para o espaço de busca definido, porém apenas o operador de Adição de Antecedentes é ideal para este espaço. O operador de Exclusão de Antecedentes não é ideal porque não é completo, já que este gera apenas subsequências da cláusula corrente, ao invés de subconjuntos ordenados desta.

Pelo fato de ser ideal, e portanto completo, o operador de Adição de Antecedentes é também redundante. No FORTE_MBC esta redundância não torna o processo de revisão mais caro. Durante a busca por uma especialização de uma cláusula, dado

que um certo literal ou conjunto de literais da *Bottom Clause* foi escolhido para ser adicionado à cláusula, não é possível fazer *backtracking*, i.e., escolher um outro literal ou conjunto de literais ao invés do primeiro escolhido. Logo, apesar de existirem cláusulas equivalentes no reticulado, apenas uma delas será efetivamente gerada durante a busca no espaço de cláusulas. Porém, se for desejável que o operador seja não-redundante, pode-se considerar a mesma abordagem do operador original do sistema Progol (operador ρ_0 definido em [24]), que considera que as cláusulas são subsequências da *Bottom Clause*. Desta forma, é imposta uma ordem aos literais da *Bottom Clause*, de forma que estes só podem ser adicionados à cláusula corrente na ordem em que estão na *Bottom Clause*. Porém, ao fazer isso, a completude do operador é perdida, assim como também deixa de ser ideal, o que não é interessante para operadores que começam a busca de um ponto qualquer do reticulado.

Por outro lado, o operador de Exclusão de Antecedentes não é completo mas é redundante, como exemplificado na Figura 4.3. Porém, apesar de isso permitir que a mesma cláusula seja gerada por caminhos diferentes no grafo de refinamento, a busca executada no FORTE_MBC só permitiria que um único caminho fosse seguido.

4.3 Espaço de busca de teorias e operadores de refinamento de teorias do FORTE_MBC

Na seção anterior caracterizamos o espaço de busca de cláusulas do FORTE_MBC definindo o reticulado $\langle L_\perp(M), \succeq_\perp \rangle$, onde $L_\perp(M)$ é a linguagem de cláusulas e $\succeq_\perp$ é a *subsumption* proposicional relativa a uma *Bottom Clause*. Com relação ao espaço de busca de teorias do FORTE_MBC, este consiste de teorias relevantes tal que cada teoria é um conjunto de cláusulas de $L_\perp(M)$, i.e., cada teoria está em $2^{L_\perp(M)}$. Como $\succeq_\perp$ é a ordem de generalidade que estrutura o reticulado de cláusulas, iremos considerar a mesma ordem para estruturar o reticulado de teorias, tal que dadas duas teorias T_1 e T_2 em $2^{L_\perp(M)}$, $T_1 \succeq_\perp T_2$ implica que $\forall C_2 \in T_2, \exists C_1 \in T_1 | C_1 \succeq_\perp C_2$.

Podemos usar a mesma ordem de generalidade pois, primeiramente, toda cláusula em uma dada teoria é subconjunto de alguma *Bottom Clause*, visto que a teoria é relevante. Segundo que, aplicando os operadores de revisão do FORTE_MBC a teorias T_1 e T_2, seja para especializar T_1 gerando T_2, ou para generalizar T_2 gerando T_1, a relação de ordem entre T_1 e T_2 é dada pela $\succeq_\perp$ aplicada a teorias, conforme explicado a seguir:

- Se aplicarmos especialização por Adição de Antecedentes a uma teoria T_1, especializando uma determinada cláusula $\overrightarrow{C_1}$, e gerando uma teoria T_2, $\overrightarrow{C_1}$ será subconjunto ordenado de qualquer cláusula $\overrightarrow{C_2}$ de T_2 que não está em T_1 e, portanto, $T_1 \succeq_\perp T_2$.

- Se aplicarmos especialização por Exclusão de Regras a uma teoria T_1, gerando T_2, as cláusulas de T_2 estão em T_1 e, portanto, $T_1 \succeq_\perp T_2$.

- Se aplicarmos generalização por Exclusão de Antecedentes a uma teoria T_2, deletando-se uma cláusula $\overrightarrow{C_2}$ e adicionando-se generalizações desta, gerando assim uma teoria T_1, as cláusulas de T_1 que não estão em T_2 são subconjuntos ordenados de $\overrightarrow{C_2}$ em T_2 e, portanto, $T_1 \succeq_\perp T_2$.

- Por fim, se aplicarmos generalização adicionando-se novas cláusulas à teoria T_2, gerando T_1, seja por Adição de Regras ou por Adição de Novas Regras, a relacao de $T_1 \succeq_\perp T_2$ continua válida, pois todas as cláusulas de T_2 estão em T_1.

Os teoremas a seguir demonstram que o par $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$ é um reticulado. Para isso, primeiramente é demonstrado que a ordem de generalidade $\succeq_\perp$ também é uma quasi-ordem para a linguagem $2^{L_\perp(M)}$ de teorias, e em seguida que cada par de teorias em $2^{L_\perp(M)}$ tem uma especialização máxima e uma generalização mínima em $2^{L_\perp(M)}$ (baseado nos lemas de especialização máxima e generalização mínima em um conjunto de teorias ordenados por *θ-subsumption*, apresentados em [20]).

Teorema 5. *$\succeq_\perp$ é uma quasi order em $2^{L_\perp(M)}$.*

Demonstração. Para toda teoria $T \in 2^{L_\perp(M)}$ temos que $T \succeq_\perp T$, pois toda cláusula da teoria é subconjunto de si mesma, portanto a relação $\succeq_\perp$ aplicada a teorias de $2^{L_\perp(M)}$ é reflexiva. Seja T_1, T_2 e T_3 teorias em $2^{L_\perp(M)}$ tal que $T_1 \succeq_\perp T_2$ e $T_2 \succeq_\perp T_3$. Então para toda cláusula $\overrightarrow{C_2} \in T_2$, existe uma cláusula $\overrightarrow{C_1} \in T_1$ tal que $\overrightarrow{C_1} \succeq_\perp \overrightarrow{C_2}$, e para toda cláusula $\overrightarrow{C_3} \in T_3$, existe uma cláusula $\overrightarrow{C_2} \in T_2$ tal que $\overrightarrow{C_2} \succeq_\perp \overrightarrow{C_3}$. Como a relação $\succeq_\perp$ é transitiva na linguagem de cláusulas $L_\perp(M)$, temos que para toda cláusula $\overrightarrow{C_3} \in T_3$ existe uma cláusula $\overrightarrow{C_1} \in T_1$ tal que $\overrightarrow{C_1} \succeq_\perp \overrightarrow{C_3}$. Logo $T_1 \succeq_\perp T_3$. A relação $\succeq_\perp$ é reflexiva e transitiva, e portanto é uma quasi-ordem em $2^{L_\perp(M)}$. □

Teorema 6. *Para cada par de teorias T_1 e T_2 em $2^{L_\perp(M)}$, existe uma especialização máxima de T_1 e T_2 em $2^{L_\perp(M)}$.*

Demonstração. Uma especialização máxima de T_1 e T_2 é dada por:
$T' = \{gs_H(\overrightarrow{C_1}, \overrightarrow{C_2}) | \langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle \in T_1 \times T_2$, tal que $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ possuem a mesma cabeça$\}$.

onde $gs_H(\overrightarrow{C_1}, \overrightarrow{C_2})$ é uma especialização máxima de $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$. T' é formada por especializações máximas de pares de cláusulas de $T_1 \times T_2$, considerando apenas pares que possuem o mesmo predicado na cabeça. Temos dois casos:

Caso 1: $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle \in T_1 \times T_2$, tal que $\overrightarrow{C_1} \not\subseteq \overrightarrow{C_2}$ e $\overrightarrow{C_2} \not\subseteq \overrightarrow{C_1}$. Neste caso, se existe uma *Bottom Clause* $\overrightarrow{\perp}$ tal que tanto $\overrightarrow{C_1}$ quanto $\overrightarrow{C_2}$ são subconjuntos ordenados de $\overrightarrow{\perp}$, uma especialização máxima de $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle$ também existe e será, no pior caso, a própria $\overrightarrow{\perp}$.

Caso 2: $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle \in T_1 \times T_2$, tal que $\overrightarrow{C_1} \subseteq \overrightarrow{C_2}$ ou $\overrightarrow{C_2} \subseteq \overrightarrow{C_1}$. Neste caso, $\overrightarrow{C_1} = \overrightarrow{C_2}$ ou existe uma *Bottom Clause* $\overrightarrow{\perp}$ tal que tanto $\overrightarrow{C_1}$ quanto $\overrightarrow{C_2}$ são subconjuntos ordenados de $\overrightarrow{\perp}$, e uma especialização máxima de $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle$ também existe.

Se T' é uma especialização máxima, então $T' \succeq_\perp S$ para todas as especializações S de T_1 e T_2. Logo temos que $\forall \overrightarrow{C} \in S$, $\exists \overrightarrow{D} \in T' | \overrightarrow{D} \succeq_\perp \overrightarrow{C}$. Para toda cláusula $\overrightarrow{C} \in S$, ou $\overrightarrow{C}$ é uma especialização de um par em $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle \in T_1 X T_2$, ou $\overrightarrow{C} \in T_1 \cap T_2$. Se $\overrightarrow{C}$ é uma especialização de um par $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle$, então existe $\overrightarrow{D} \in T'$ tal que $\overrightarrow{D} \succeq_\perp \overrightarrow{C}$, porque T' tem todas as especializações máximas dos pares em $T_1 \times T_2$, incluindo o par $\langle \overrightarrow{C_1}, \overrightarrow{C_2} \rangle$, e pelo fato de ser especialização máxima podemos afirmar que $\overrightarrow{D} \succeq_\perp \overrightarrow{C}$. Se $\overrightarrow{C} \in T_1 \cap T_2$, então $\overrightarrow{C} \in T_1$ e $\overrightarrow{C} \in T_2$, e portanto $gs_H(\overrightarrow{C}, \overrightarrow{C}) \in T'$. Mas $gs_H(\overrightarrow{C}, \overrightarrow{C}) = \overrightarrow{C}$, logo $\overrightarrow{C} \in T'$, e $\overrightarrow{C} \succeq_\perp \overrightarrow{C}$. Provamos que $\forall \overrightarrow{C} \in S, \exists \overrightarrow{D} \in T' | \overrightarrow{D} \succeq_\perp \overrightarrow{C}$, portanto $T' \succeq_\perp S$ e T' é uma especialização máxima. □

Teorema 7. *Para cada par de teorias T_1 e T_2 em $2^{L_\perp(M)}$, existe uma generalização mínima de T_1 e T_2 em $2^{L_\perp(M)}$.*

Demonstração. Uma generalização mínima para $\{T_1, T_2\}$ é dada por $T' = T_1 \cup T_2$. T' é uma generalização de $\{T_1, T_2\}$, i.e., $T' \succeq_\perp T_1$ e $T' \succeq_\perp T_2$, já que todas as cláusulas em T_1 e T_2 também estão em T'. T' é uma generalização mínima de $\{T_1, T_2\}$, i.e., para toda teoria S tal que $S \succeq_\perp T_1$ e $S \succeq_\perp T_2$, temos que $S \succeq_\perp T'$. Temos que para toda cláusula $\overrightarrow{C}$ de T', existe uma cláusula $\overrightarrow{C_i}$ de T_i (T_1 ou T_2) tal que $\overrightarrow{C} = \overrightarrow{C_i}$. Para esta cláusula $\overrightarrow{C_i}$, existe uma cláusula $\overrightarrow{D}$ de S tal que $\overrightarrow{D} \succeq_\perp \overrightarrow{C_i}$. Logo $\overrightarrow{D} \succeq_\perp \overrightarrow{C}$. Então, para toda cláusula $\overrightarrow{C}$ de T' existe uma cláusula $\overrightarrow{D}$ de S tal que $\overrightarrow{D} \succeq_\perp \overrightarrow{C}$, e portanto $S \succeq_\perp T'$. □

Teorema 8. *O par $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$ é um* **reticulado**.

Demonstração. De acordo com os teoremas anteriores, $\succeq_\perp$ é uma quasi-ordem quando aplicado a teorias em $2^{L_\perp(M)}$, e cada par de teorias em $2^{L_\perp(M)}$ tem uma generalização mínima e uma especialização máxima em $2^{L_\perp(M)}$. Logo $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$ é um **reticulado**. □

Como visto anteriormente, para especializar uma teoria no FORTE_MBC podemos usar os operadores de refinamento de teorias de Adição de Antecedentes e Exclusão de Regras, e para generalizar uma teoria podemos usar os operadores de refinamento de teorias de Exclusão de Antecedentes, Adição de Regras e Adição de Novas Regras. Ambos os operadores de especialização de teorias do FORTE_MBC podem ser aplicados a um ponto de revisão de especialização e propor uma revisão para este, porém apenas uma destas revisões propostas será escolhida, como determinado pelo algoritmo de busca *Hill Climbing* do FORTE_MBC, definido em

Algoritmo 3.1. Nesta Tese, para caracterizarmos formalmente os operadores de teoria do FORTE_MBC, iremos combinar estes dois operadores em um único operador de especialização de teorias, tal que este possui dois possíveis passos de refinamento, como é normalmente feito em trabalhos que envolvem definição de operadores de refinamento de teorias [13, 20, 35]. A mesma observação pode ser feita para os operadores de generalização de teorias do FORTE_MBC.

Ambos os operadores de refinamento de teorias do FORTE_MBC buscam no reticulado definido, porém a busca pode começar de qualquer ponto deste reticulado, dependendo da teoria corrente sendo revisada, como mostrado na Figura 4.4. O reticulado é limitado inferiormente pela teoria vazia {} (no caso em que todas as cláusulas da teoria corrente são excluídas, aplicando-se o operador de teorias de Exclusão de Regras) e superiormente por uma teoria formada por cláusulas contendo apenas a cabeça (uma cláusula para cada predicado alvo que se deseja aprender, caso que pode acontecer quando o operador de Adição de Regras foi usado para se ter uma cláusula para cada predicado definido no conhecimento preliminar, e em conjunto foi usado o operador de Exclusão de Antecedentes, para excluir todos os antecedentes do corpo das cláusulas), representada na figura por {□}.

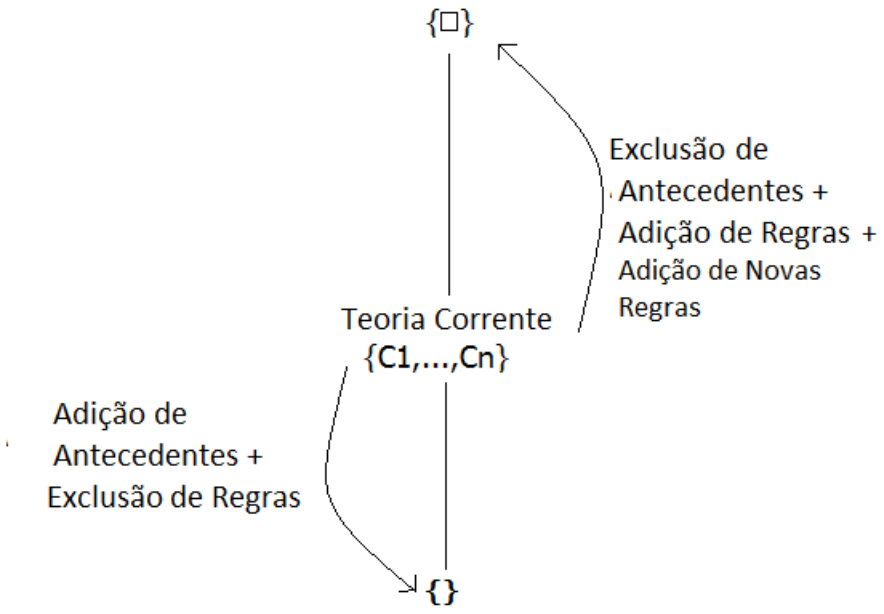

Figura 4.4: Reticulado de teorias e espaço de busca dos operadores de refinamento de teorias

Procedemos agora à definição formal dos operadores de refinamento de teorias do FORTE_MBC, começando-se pelo de especialização.

Definição 27 (Operador de especialização de teorias do FORTE_MBC ρ_T)**.** $T_2 \in \rho_T(T_1)$ se e somente se:

$T_2 = (T_1 \setminus \{\overrightarrow{C}\}) \cup S | S = \{\overrightarrow{C_1}, ..., \overrightarrow{C_n}\}$, tal que $\overrightarrow{C_i} \in \rho_h^*(\overrightarrow{C})$, para $1 \leq i \leq n$, onde ρ_h é o operador de refinamento de cláusulas de Adição de Antecedentes, ou

$T_2 = T_1 \setminus \{\overrightarrow{C}\}$, aplicando o operador de Exclusão de Regras.

A definição acima indica que, em um passo de refinamento do operador ρ_T, ou uma nova teoria é retornada contendo uma ou mais especializações de uma das cláusulas da teoria, tal que a cláusula original é removida, ou uma cláusula é excluída da teoria original. Observe a diferença entre o operador de especialização de cláusulas e o operador de especialização de teorias do FORTE_MBC: cada passo de refinamento do operador de cláusulas corresponde a adicionar um antecedente à cláusula corrente sob revisão. Após um determinado número de passos de refinamento, uma nova cláusula é criada e adicionada à teoria. Já com relação ao operador de teorias, se o operador de revisão de Adição de Antecedentes é aplicado, cada passo de refinamento corresponde a adicionar uma ou mais cláusulas novas inteiras à teoria (através do operador de refinamento de cláusulas de Adição de Antecedentes). Após um determinado número de passos de refinamento, uma nova teoria é criada como resultado da revisão.

O operador ρ_T é *localmente finito* e *completo* (e portanto *redundante*) para o reticulado definido, mas este operador não é *próprio* como provado a seguir:

Teorema 9. *No reticulado* $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$*, o operador de refinamento* ρ_T *é localmente finito e completo.*

Demonstração.

(*localmente finito*) Lembrando o operador de refinamento de teorias de Adição de Antecedentes apresentado no Capítulo 3.1, este adiciona especializações de uma determinada cláusula até que todos os exemplos positivos provados originalmente pela cláusula estejam sendo provados pelas suas especializações. Como o número de exemplos positivos é finito, assim como o número de *Bottom Clauses* que podem ser usadas para a geração destas especializações, o número de cláusulas que podem ser adicionadas pelo primeiro passo de refinamento do operador ρ_T é finito. Por fim, caso consideremos o segundo passo de refinamento, correspondente a deletar cláusulas da teoria, como a teoria é finita, o número de refinamentos possíveis também é. Logo, o operador ρ_T é **localmente finito**.

(*completo*(prova baseada em [13])) Suponha as teorias $T', T'' \in 2^{L_\perp(M)}$ tal que $T' \succeq_\perp T''$, e seja n = $|T''|$. Então $\forall \overrightarrow{D_i} \in T''$, $1 \leq i \leq n$, $\exists \overrightarrow{C_i} \in T' | \overrightarrow{C_i} \succeq_\perp \overrightarrow{D_i}$. Como o operador de refinamento de cláusulas de Adição de Antecedentes ρ_h é completo, então se $\overrightarrow{D_i}$ (ou uma cláusula equivalente a $\overrightarrow{D_i}$) $\notin T'$, $\overrightarrow{D_i} \in \rho_h^{k_i}(\overrightarrow{C_i})$, para algum k_i. Começando-se a partir de $T_1 = T'$ e aplicando o primeiro passo de refinamento de ρ_T iterativamente, obtemos para cada T_j um refinamento $T_{j+1} = (T_j \setminus \{\overrightarrow{C_j}\}) \cup S | S =$

$\{\overrightarrow{D_{j_1}}, ..., \overrightarrow{D_{j_m}}\}$ tal que $\overrightarrow{D_{j_i}} \in \rho_h^*(\overrightarrow{C_j}), 1 \leq i \leq m$, escolhendo-se sempre uma cláusula $\overrightarrow{C_j}$ tal que $\overrightarrow{C_j} \in (T_j \setminus T'')$ tal que $\exists \overrightarrow{D} \in T'' | \overrightarrow{C_j} \succeq_\perp \overrightarrow{D}$. Eventualmente, acharemos um $T_k \in \rho_T^k(T')$, $k \leq max_{i=1}^n(k_i)$, tal que para todo i, $\overrightarrow{D_i} \in T_k$. Se T_k é maior que T'' em número de cláusulas, então o passo de refinamento de Exclusão de Regras pode ser aplicado para deletar as cláusulas sobressalentes, até obtermos T'' (ou uma teoria equivalente a T''). Como existe um caminho finito de T' a T'' (ou a uma teoria equivalente) no grafo de refinamento, o operador ρ_T é **completo**. Note que o refinamento de Exclusão de Regras é essencial para a completude de ρ_T. Note também que apesar do operador ser completo ele não garante que a melhor teoria é achada, já que o FORTE_MBC executa uma busca gulosa. Porém se uma busca completa é usada, FORTE_MBC pode achar a melhor teoria para este espaço limitado. □

Como o operador ρ_T é completo, ele também é **redundante**. Se considerarmos o primeiro passo de refinamento onde cada especialização é gerada aplicando-se o operador de cláusulas de Adição de Antecedentes, podemos ter redundância porque este operador é redundante. Se considerarmos o passo de refinamento correspondente a excluir uma cláusula, podemos ter redundância de acordo com o seguinte exemplo:

Exemplo 2. *Suponha as teorias T_1 e T_2 abaixo:*

$$T_1 : \overrightarrow{C_1} = h(A, B) \leftarrow b(A, C).$$
$$\overrightarrow{C_2} = h(A, B) \leftarrow f(B, C), b(C, A).$$
$$\overrightarrow{C_3} = h(A, B) \leftarrow d(A, C), d(C, B).$$

$$T_2 : \overrightarrow{C_1} = h(A, B) \leftarrow b(A, C).$$
$$\overrightarrow{C_2} = h(A, B) \leftarrow f(B, C), b(C, A).$$
$$\overrightarrow{C_4} = h(A, B) \leftarrow d(A, C), m(C, B).$$

T_1 não pertence ao refinamento de T_2, porque a cláusula $\overrightarrow{C_3}$ de T_1 não pode ser obtida por exclusão de regras e nem por adição de antecedentes a nenhuma cláusula de T_2. O oposto também é verdade, i.e., T_2 não pertence ao refinamento de T_1, pois a cláusula $\overrightarrow{C_4}$ de T_2 não pode ser obtida a partir de T_1 aplicando-se exclusão de regras ou adição de antecedentes à cláusula. Dado isso, podemos obter a teoria $T = \{\overrightarrow{C_1}, \overrightarrow{C_2}\}$ a partir de caminhos diferentes, deletando-se $\overrightarrow{C_3}$ de T_1, ou deletando-se $\overrightarrow{C_4}$ de T_2.

Teorema 10. *No reticulado $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$, o operador de refinamento ρ_T* **não** *é próprio.*

Demonstração. Vimos que o operador de refinamento de cláusulas de Adição de

Antecedentes é próprio, porém isto não é suficiente para garantir que ρ_T seja próprio, como mostrado no exemplo 3.

Exemplo 3. *Suponha a teoria T_2 abaixo, vista no exemplo do capítulo 3.1:*

$$T_2 : \overrightarrow{C_1} = great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D), r\_subst\_2(A,E), ring\_substitutions(A,D).$$
$$\overrightarrow{C_2} = great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D), r\_subst\_2(A,E), x\_subst(B,F,G), ring\_substitutions(A,D).$$
$$\overrightarrow{C_3} = great\_ne(A,B) \leftarrow ring\_subst\_3(B,C), ring\_substitutions(A,D), ring\_substitutions(B,E), gt(E,D), alk\_groups(A,E).$$
$$\overrightarrow{C_4} = great\_ne(A,B) \leftarrow ring\_substitutions(A,C), gt(C,D), ring\_subst\_2(A,E).$$

A teoria T_2 foi criada aplicando-se o passo de refinamento correspondente ao operador de teorias de Adição de Antecedentes à teoria T, tal que ambas as cláusulas $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ foram introduzidas. Observe que $\overrightarrow{C_1}$ é subconjunto de $\overrightarrow{C_2}$. Suponha que T_2 é agora a teoria corrente a ser revisada, e para especializá-la é aplicado o passo de refinamento de Exclusão de Regras do operador ρ_T, deletando-se a cláusula $\overrightarrow{C_2}$ desta, gerando a teoria $T_3 = \{\overrightarrow{C_1}, \overrightarrow{C_3}, \overrightarrow{C_4}\}$. Temos que T_2 e T_3 são teorias equivalentes com relação à subsumption proposicional relativa a uma Bottom Clause, i.e, $T_2 \succeq_\perp T_3$ e $T_3 \succeq_\perp T_2$, como provado a seguir:

- *$T_2 \succeq_\perp T_3$: já que estamos especializando T_2 para obter T_3.*
- *$T_3 \succeq_\perp T_2$: temos que as cláusulas $\overrightarrow{C_1}, \overrightarrow{C_3}$ e $\overrightarrow{C_4}$ de T_2 estão também em T_3. Com relação à cláusula $\overrightarrow{C_2} \in T_2$, existe a cláusula $\overrightarrow{C_1} \in T_3$ tal que $\overrightarrow{C_1} \succeq_\perp \overrightarrow{C_2}$, onde $\overrightarrow{\perp}$ é a mesma Bottom Clause usada para criar $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ quando T_2 foi criada. Portanto temos que $T_3 \succeq_\perp T_2$.*

Já que o refinamento da teoria T_2 induz uma teoria equivalente T_3, o operador ρ_T não é próprio.

□

A situação acima só acontece por conta da combinação de duas situações:

- A teoria corrente sob revisão tem no mínimo duas cláusulas tais que uma é subconjunto da outra, o que significa que a teoria é redundante.
- É aplicada uma combinação específica de passos de refinamento do operador ρ_T: o primeiro passo de refinamento que aplica operador de teorias de Adição de Antecedentes, seguido pelo passo de Exclusão de Regras.

Como ρ_T não é próprio, ele **não é ideal** para $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$.

Procedemos agora à definição formal do operador de generalização de teorias do FORTE_MBC:

Definição 28 (Operador de generalização de teorias do FORTE_MBC δ_T)**.** $T_2 \in \delta_T(T_1)$ se e somente se:

$T_2 = (T_1 \setminus \{\overrightarrow{C}\}) \cup S | S = \{\overrightarrow{C_1}, ..., \overrightarrow{C_n}\}$, tal que $\overrightarrow{C_i} \in \delta_h^*(\overrightarrow{C})$, para $1 \leq i \leq n$, aplicando-se o operador de teorias de Exclusão de Antecedentes, onde δ_h é o operador de cláusulas de Exclusão de Antecedentes, ou

$T_2 = T_1 \cup S | S = \{\overrightarrow{E_1}, ..., \overrightarrow{E_n}\}$, tal que $\overrightarrow{E_i} \in \rho_h^*(\overrightarrow{D})$ e $\overrightarrow{D} \in \delta_h^*(\overrightarrow{C})$, para $\overrightarrow{C} \in T_1$ e $1 \leq i \leq n$, aplicando-se o operador de Adição de Regras, ou

$T_2 = T_1 \cup S | S = \{\overrightarrow{C_1}, ..., \overrightarrow{C_n}\}$, tal que $\overrightarrow{C_i} \in \rho_h^*(\square)$, aplicando o operador de Adição de Novas Regras, onde $\square \in 2^{L_\perp(M)}$ é uma cláusula contendo apenas a cabeça e $\square \notin \overrightarrow{C}, \forall \overrightarrow{C} \in T_1$.

A definição acima indica que, em um passo de refinamento do operador, uma nova teoria é retornada incluindo uma ou mais generalizações de uma cláusula da teoria, tal que esta é removida da teoria, ou incluindo uma ou mais especializações de uma cláusula tal que esta é mantida na teoria, ou incluindo uma ou mais novas cláusulas para um predicado que não aparece na cabeça de nenhuma outra cláusula da teoria, tal que este predicado está definido no conhecimento preliminar.

O operador de generalização de teorias do FORTE_MBC é *localmente finito* para o reticulado definido, porém ele não é *completo* e não é *próprio*, como provado a seguir:

Teorema 11. *No reticulado $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$, o operador de refinamento δ_T é localmente finito.*

Demonstração. O passo de refinamento correspondente ao operador de teorias de Exclusão de Antecedentes cria um conjunto finito de subsequências, pois a cláusula original é finita. Considerando os refinamentos de Adição de Regras e Adição de Novas Regras, ambos introduzem uma ou mais cláusulas usando o operador de teorias de Adição de Antecedentes, que introduz um número de cláusulas finito, de acordo com o teorema 9. Portanto o operador δ_T é **localmente finito**. □

Teorema 12. *No reticulado $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$, o operador de refinamento δ_T* **não** *é completo.*

Demonstração. A prova será feita pelo contra-exemplo abaixo:

Exemplo 4. *Suponha as teorias $T_1, T_2 \in 2^{L_\perp(M)}$ abaixo, tal que $T_1 \succeq_\perp T_2$:*

T_1 :
$\overrightarrow{C_1} = p(X,Y) \leftarrow q(X,W), q(W,U).$
$\overrightarrow{C_2} = p(X,Y) \leftarrow q(Y,X).$

T_2 :
$\overrightarrow{C_3} = p(X,Y) \leftarrow q(Y,W), q(W,U), q(X,W).$
$\overrightarrow{C_2} = p(X,Y) \leftarrow q(Y,X).$

Assumindo modos $(+,+)$ e $(+,-)$ para os predicados p e q, respectivamente, temos que T_1 não está em $\delta_T^*(T_2)$. Se o passo de refinamento correspondente ao operador de teorias de Exclusão de Antecedentes é aplicado, ele não poderia gerar a cláusula $\overrightarrow{C_1}$ de T_1, já que $\overrightarrow{C_1}$ não é uma subsequência de $\overrightarrow{C_3}$ e nem de $\overrightarrow{C_2}$ em T_2. Na verdade, ele poderia tentar gerar a cláusula $p(X,Y) \leftarrow q(Y,W), q(W,U)$ pela remoção do literal $q(X,W)$ de $\overrightarrow{C_3}$, mas como esta cláusula não está obedecendo aos modos definidos, ela é descartada. Portanto, a cláusula $\overrightarrow{C_1}$ só poderia ter sido criada a partir do refinamento de Adição de Regras, mas neste caso, $\overrightarrow{C_3}$ deveria estar em T_1 (se $\overrightarrow{C_1}$ é criada a partir de $\overrightarrow{C_2}$, então $\overrightarrow{C_3}$ não é alterada. Se $\overrightarrow{C_1}$ é criada a partir de $\overrightarrow{C_3}$, então $\overrightarrow{C_3}$ é mantida na teoria). Observe que $\overrightarrow{C_1}$ não poderia ter sido gerada pelo Adição de Novas Regras, pois possui o mesmo predicado da cabeça das outras cláusulas. Não é possível gerar T_1 a partir de T_2 aplicando-se apenas o operador de generalização, sem combiná-lo com o operador de especialização (neste caso para excluir a cláusula $\overrightarrow{C_3}$), e portanto este contra-exemplo prova que o operador δ_T não é completo.

□

Mesmo o operador δ_T não sendo completo, ele é **redundante**. Considerando o passo de refinamento de Exclusão de Antecedentes, podemos ter redundância porque o operador de cláusulas de Exclusão de Antecedentes usado é redundante. Também podemos ter redundância na combinação das duas opções de refinamento, como mostrado no exemplo a seguir:

Exemplo 5. *Suponha a teoria corrente T_1 dada por:*

$T_1 : \overrightarrow{C_1} = h(A,B) \leftarrow p(A,C), p(C,D).$
$\quad\quad \overrightarrow{C_2} = h(A,B) \leftarrow r(A,D), r(D,E).$

e suponha duas teorias diferentes $T_2 = \{\overrightarrow{C_1}, \overrightarrow{C_2}, \overrightarrow{C_1'}\}$ e $T_3 = \{\overrightarrow{C_1}, \overrightarrow{C_2}, \overrightarrow{C_2'}\}$ que podem ser geradas a partir de T_1 aplicando-se o refinamento de Adição de Regras, onde $\overrightarrow{C_1'}$ e $\overrightarrow{C_2'}$ são dadas por:

$\overrightarrow{C_1'} = h(A,B) \leftarrow p(A,C), p(C,B), p(B,A).$ *(gerada pela exclusão de p(C,D) em $\overrightarrow{C_1}$ e adição de p(C,B) e p(B,A))*

$\overrightarrow{C_2'} = h(A,B) \leftarrow r(A,D), r(D.B), p(B,A)$. *(gerada pela exclusão de r(D,E) em* $\overrightarrow{C_2}$ *e adição de r(D,B) e p(B,A))*

No grafo de refinamento, o operador pode aplicar o passo de refinamento de Exclusão de Antecedentes nas teorias T_2 *e* T_3*, excluindo-se, por exemplo, os literais* $p(A,C)$ *e* $p(C,B)$ *de* $\overrightarrow{C_1'}$ *e* $r(A,D)$ *e* $r(D,B)$ *de* $\overrightarrow{C_2'}$*. Neste caso, o resultado será a mesma teoria* $T_4 = \{\overrightarrow{C_1}, \overrightarrow{C_2}, \overrightarrow{C_3}\}$*, onde* $\overrightarrow{C_3} = h(A,B) \leftarrow p(B,A)$*. Como o operador pode alcançar a mesma teoria a partir de dois caminhos diferentes no grafo de refinamento, o operador é* **redundante**.

Teorema 13. *No reticulado* $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$*, o operador de refinamento* δ_T **não** *é próprio.*

Demonstração. A prova será feita pelo contra-exemplo abaixo:

Exemplo 6. *Suponha a teoria* T_4 *abaixo do exemplo do Capítulo 3.1:*

$T_4 : \overrightarrow{C_5} = great\_ne(A,B) \leftarrow alk\_groups(A,C), r\_subst\_2(A,D).$
$\overrightarrow{C_4} = great\_ne(A,B) \leftarrow alk\_groups(A,C), r\_subst\_2(A,D),$
$ring\_subst\_3(A,E), ring\_substitutions(A,C), ring\_subst\_3(B,F).$
$\overrightarrow{C_3} = great\_ne(A,B) \leftarrow ring\_subst\_3(B,C), ring\_substitutions(A,D),$
$ring\_substitutions(B,E), gt(E,D), alk\_groups(A,E).$
$\overrightarrow{C_2} = great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D),$
$r\_subst\_2(A,E).$
$\overrightarrow{C_1} = great\_ne(A,B) \leftarrow ring\_substitutions(A,C), gt(C,D),$
$ring\_subst\_2(A,E).$

A teoria T_4 *foi criada aplicando-se o operador de revisão de Adição de Regras na teoria* $T = \{\overrightarrow{C_3}, \overrightarrow{C_2}, \overrightarrow{C_1}\}$*, tal que* $\overrightarrow{C_4}$ *e* $\overrightarrow{C_5}$ *foram as cláusulas introduzidas. Neste exemplo temos que* T_4 *e* T **não** *são teorias equivalentes, porém* T_4 *é uma teoria redundante, já que a cláusula* $\overrightarrow{C_5}$ *é subconjunto da cláusula* $\overrightarrow{C_4}$*. Já vimos anteriormente que, quando a teoria corrente a ser revisada é redundante, a especialização por Exclusão de Regras pode gerar uma teoria equivalente. Porém, neste caso, mesmo um simples refinamento de Exclusão de Antecedentes aplicado a* T_4 *poderia gerar uma teoria equivalente. Suponha a teoria* $T_5 = \{\overrightarrow{C_5}, \overrightarrow{C_4'}, \overrightarrow{C_3}, \overrightarrow{C_2}, \overrightarrow{C_1}\}$*, onde* $\overrightarrow{C_4'}$ *é dada por:*

$\overrightarrow{C_4'} = great\_ne(A,B) \leftarrow alk\_groups(A,C), r\_subst\_2(A,D),$
$ring\_subst\_3(A,E).$

$\overrightarrow{C_4'}$ *foi criada pela exclusão dos antecedentes* $ring\_substitutions(A,C)$ *e* $ring\_subst\_3(B,F)$ *da cláusula* $\overrightarrow{C_4} \in T_4$*. A nova teoria* T_5 *e a teoria anterior* T_4 *são equivalentes com relação à subsumption proposicional relativa a uma Bottom Clause, como provado a seguir:*

- $T_5 \succeq_\perp T_4$*: imediato, já que estamos generalizando* T_4 *para obter* T_5*.*
- $T_4 \succeq_\perp T_5$*: temos que as cláusulas* $\overrightarrow{C_5}, \overrightarrow{C_3}, \overrightarrow{C_2}$ *e* $\overrightarrow{C_1}$ *de* T_5 *pertencem também a* T_4*. Com relação à cláusula* $\overrightarrow{C_4'} \in T_5$*, temos que* $\overrightarrow{C_5} \in T_4$ *é subsequência desta. Portanto* $T_4 \succeq_\perp T_5$*.*

A situação anterior acontece porque a teoria corrente é redundante. Neste caso, a culpa é do operador de teorias de Adição de Antecedentes que é aplicado internamente no operador de Adição de Regras. É o operador de teorias de Adição de Antecedentes que permite a geração de teorias redundantes. Quando combinamos o passo de refinamento de Adição de Regras com o de Exclusão de Antecedentes, teorias equivalentes são geradas pelo operador δ_T, caracterizando um operador não próprio.

Como o operador δ_T não é completo e nem próprio, ele **não é ideal** para $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$. □

4.3.1 Considerações a respeito dos operadores de refinamento de teorias

Ambos os operadores de especialização e generalização de teorias do FORTE_MBC não são próprios, como provado nos teoremas 10 e 13. Isto acontece precisamente por causa do operador de teorias de Adição de Antecedentes, que permite a introdução de cláusulas redundantes na teoria, já que ele não leva em consideração as cláusulas que já estão na teoria quando criando uma nova cláusula. Apesar disso, a busca implementada no FORTE_MBC não consideraria as teorias redundantes criadas como possíveis revisões a serem implementadas já que estas teorias não melhoram a pontuação da Teoria Corrente. Portanto, o fato dos operadores de refinamento de teorias do FORTE_MBC não serem próprios não é um problema, pelo menos não para a busca implementada, mas pode ser tornar um problema se uma busca diferente for usada.

Apenas o operador ρ_T é completo para o reticulado definido. O operador δ_T não é completo por causa da não completude do operador de refinamento de cláusulas de Exclusão de Antecedentes. Ser completo é uma importante característica, especialmente porque no FORTE_MBC, como estamos fazendo revisão, a busca pode começar de qualquer ponto do grafo de refinamento. Completude leva à redundância (no caso do operador de generalização de teorias, ele é redundante mesmo não sendo completo), porém, o FORTE_MBC executa uma busca que não permite *backtracking*, tal que se um caminho no grafo de refinamento é escolhido, não é possível voltar atrás e escolher um outro caminho. Devido a isso, a redundância do operador não afeta a performance do sistema.

Além das redundâncias impostas pelos operadores de refinamento de teorias do FORTE_MBC, também podemos citar a redundância que pode acontecer quando combinando estes dois operadores durante a execução do algoritmo do FORTE_-MBC. A aplicação de passos de refinamento de especialização e generalização em uma ordem específica durante a busca por uma teoria revisada, pode também levar a teorias equivalentes. Porém, a revisão que leva à teoria equivalente não seria selecionada no FORTE_MBC, já que não melhora a pontuação da teoria. Portanto, este tipo de redundância também não afeta a performance do FORTE_MBC.

4.4 Operadores de refinamentos ideais para o FORTE_MBC

Nesta seção propomos soluções para cada um dos problemas citados nas seções anteriores, propondo operadores de refinamento ideais de teorias e cláusulas para o FORTE_MBC.

4.4.1 Uma solução para a não completude do operador de refinamento de cláusulas de Exclusão de Antecedentes do FORTE_MBC

Dada uma teoria corrente T e uma cláusula corrente $\overrightarrow{C} \in T$ a ser generalizada, o operador de cláusulas de Exclusão de Antecedentes desconsidera as subsequências de $\overrightarrow{C}$ que não estão obedecendo aos modos definidos. Porém, uma reorganização dos literais pode fazer com que algumas destas subsequências passem a obedecer aos modos. A seguir propomos um novo algoritmo para verificar se alguma subsequência gerada pelo operador de Exclusão de Antecedentes pode ser reorganizada de forma a ser retornada como um possível refinamento. O novo algoritmo é mostrado em Algoritmo 4.1.

O algoritmo começa considerando uma cláusula temporária $\overrightarrow{D}$ que tem apenas a cabeça da cláusula $\overrightarrow{C}$. Ele então escaneia a cláusula $\overrightarrow{C}$ da esquerda para a direita, verificando, para cada literal, se a sua inclusão em $\overrightarrow{D}$ resultaria numa cláusula que obedecesse aos modos, isto é, se as variáveis de entrada do literal já apareceram como entrada na cabeça da cláusula $\overrightarrow{D}$ ou como variáveis de saída no corpo de $\overrightarrow{D}$. Se este é o caso, o literal é removido de $\overrightarrow{C}$ e adicionado à $\overrightarrow{D}$. A cláusula $\overrightarrow{C}$ pode ser escaneada várias vezes, sempre da esquerda para a direita, até que não se tenha mais literais no corpo de $\overrightarrow{C}$, ou não seja mais possível adicionar literais de $\overrightarrow{C}$ em $\overrightarrow{D}$ (pois tal inclusão resultaria numa cláusula que não obedece aos modos). Se todos os literais de $\overrightarrow{C}$ foram adicionados à $\overrightarrow{D}$, então $\overrightarrow{D}$ é a cláusula reorganizada que

estávamos procurando.

Algoritmo 4.1 Algoritmo de Exclusão de Antecedentes para reorganizar cláusulas que não obedecem aos modos

1: $\overrightarrow{C}$ = Cláusula que não está obedecendo aos modos
2: $\overrightarrow{D}$ = Cabeça de $\overrightarrow{C}$
3: **repita**
4: **para cada** literal l no corpo de $\overrightarrow{C}$ **faça**
5: **se** $\overrightarrow{D} \cup \{l\}$ obedece aos modos **então**
6: $\overrightarrow{D} = \overrightarrow{D} \cup \{l\}$
7: $\overrightarrow{C} = \overrightarrow{C} - \{l\}$
8: **até que** não seja possível adicionar um literal do corpo de $\overrightarrow{C}$ a $\overrightarrow{D}$ ou não existem mais literais no corpo de $\overrightarrow{C}$
9: **se** $\overrightarrow{C}$ contém apenas a cabeça **então**
10: retorna a cláusula $\overrightarrow{D}$
11: **senão**
12: Cláusula $\overrightarrow{C}$ não pode ser reorganizada e deve ser descartada.

A seguir mostramos dois exemplos de execução do algoritmo proposto. Para ambos os exemplos assuma os modos $p(+,+), q(+,-), r(+,-,-)$ e $s(+,+,-)$ para os predicados p, q, r e s.

Exemplo 7. *Seja* $\overrightarrow{C_1} = p(X,Y) \leftarrow q(X,Z), q(Z,W), r(W,Z,T), q(Z,T), q(T,U), s(X,U,W)$ *a cláusula a ser generalizada.*

Seja $\overrightarrow{C} = p(X,Y) \leftarrow q(X,Z), r(W,Z,T), q(Z,T), q(T,U), s(X,U,W)$ *a cláusula gerada pela exclusão do antecedente* $q(Z,W)$ *de* $\overrightarrow{C_1}$. *O literal* $r(W,Z,T)$ *não está obedecendo aos modos definidos, já que* W *é uma variável de entrada e não possui uma variável de saída correspondente ocorrendo antes na cláusula.*

O algoritmo começa criando a cláusula $\overrightarrow{D} = p(X,Y) \leftarrow$. *Na primeira iteração, ou seja, no primeiro escaneamento da cláusula* $\overrightarrow{C}$, *os literais* $q(X,Z), q(Z,T), q(T,U)$ *e* $s(X,U,W)$ *são adicionados à cláusula* $\overrightarrow{D}$, *nesta ordem, e são removidos de* $\overrightarrow{C}$. *Na segunda iteração, o literal* $r(W,Z,T)$, *que é o único literal restante em* $\overrightarrow{C}$, *é então adicionado a* $\overrightarrow{D}$, *já que agora a variável de entrada* W *já apareceu como variável de saída em algum literal do corpo de* $\overrightarrow{D}$. *A cláusula final correspondente a uma reorganização dos literais de* $\overrightarrow{C}$ *obedecendo aos modos é dada por* $\overrightarrow{D} = p(X,Y) \leftarrow q(X,Z), q(Z,T), q(T,U), s(X,U,W), r(W,Z,T)$.

Exemplo 8. *Seja* $\overrightarrow{C_2} = p(X,Y) \leftarrow q(X,Z), q(Z,W), r(W,Z,T), q(T,U), s(X,U,W)$ *a cláusula a ser generalizada.*

Seja $\overrightarrow{C} = p(X,Y) \leftarrow q(X,Z), r(W,Z,T), q(T,U), s(X,U,W)$ *a cláusula gerada pela exclusão do literal* $q(Z,W)$ *de* $\overrightarrow{C_2}$.

Começando-se a partir de $\overrightarrow{D} = p(X,Y) \leftarrow$*, na primeira iteração o literal* $q(X,Z)$ *é adicionado a* $\overrightarrow{D}$ *e removido de* $\overrightarrow{C}$*. Porém, ainda nesta iteração e na próxima, nenhum dos literais ainda em* $\overrightarrow{C}$ *podem ser adicionados a* $\overrightarrow{D}$*, já que todos possuem no mínimo uma variável de entrada que não aparece em* $\overrightarrow{D}$*. Portanto, neste exemplo, os literais de* $\overrightarrow{C}$ *não podem ser reorganizados de forma a obedecerem aos modos definidos.*

O novo operador de cláusulas de Exclusão de Antecedentes δ_h^{id} é definido como:

Definição 29 (Operador de refinamento de cláusulas de Exclusão de Antecedentes δ_h^{id})**.** Seja $L_\perp(M)$ como definido anteriormente, e $\overrightarrow{C}$ e $\overrightarrow{C}'$ duas cláusulas definidas e ordenadas em $L_\perp(M)$. $\overrightarrow{C}'$ pertence a $\delta_h^{id}(\overrightarrow{C})$ se e somente se:
$\overrightarrow{C}' = \overrightarrow{C} - \{l\}$, onde $l \in \overrightarrow{C}$

Na definição de δ_h^{id}, $\overrightarrow{C}'$ não é mais estritamente uma subsequência de $\overrightarrow{C}$ como em δ_h. Ao relaxar esta condição, um conjunto de cláusulas passa a fazer parte do grafo de refinamento do operador. Tais cláusulas são subconjuntos ordenados da cláusula corrente, e não apenas subsequências. O operador δ_h^{id} é **completo** para $\langle L_\perp(M), \succeq_\perp \rangle$, já que para quaisquer duas cláusulas $\overrightarrow{C}, \overrightarrow{D} \in L_\perp(M)$, tal que $\overrightarrow{C}$ e $\overrightarrow{D}$ não são equivalentes, e tal que $\overrightarrow{C} \succ_\perp \overrightarrow{D}$, $\overrightarrow{C}$ é um subconjunto ordenado de $\overrightarrow{D}$, e pode ser obtido pela exclusão dos literais em $\overrightarrow{D} - \overrightarrow{C}$ de $\overrightarrow{D}$, seguido da reorganização dos literais restantes, em um número finito de passos de refinamento de δ_h^{id}.

A condição que foi relaxada não afeta a finitude do operador e nem a característica de ser próprio, e portanto δ_h^{id} é **ideal** para $\langle L_\perp(M), \succeq_\perp \rangle$. A completude de δ_h^{id} implica em completude para o operador de generalização de teorias δ_T, como provado a seguir:

Teorema 14. *No reticulado* $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$*, o operador de refinamento* δ_T *que considera* δ_h^{id} *no passo de refinamento de Exclusão de Antecedentes é completo.*

Demonstração. Seja $T', T'' \in 2^{L_\perp(M)}$ tal que $T' \succeq_\perp T''$. Então $\forall \overrightarrow{D} \in T''$, $\exists \overrightarrow{C} \in T' | \overrightarrow{C} \succeq_\perp \overrightarrow{D}$. As cláusulas de T' podem ser de dois tipos:
1) É uma cláusula $\overrightarrow{C} | \overrightarrow{C} \succeq_\perp \overrightarrow{D}$, para alguma $\overrightarrow{D} \in T''$ (e neste caso estão incluídas as cláusulas que estão em ambas as teorias ou são equivalentes).
2) É uma cláusula $\overrightarrow{E} | \overrightarrow{E} \not\succeq_\perp \overrightarrow{D}$, para toda $\overrightarrow{D} \in T''$.
No caso 1, $\overrightarrow{C} \sim \overrightarrow{D}$ ou $\overrightarrow{C} \in \delta_h^{id^*}(\overrightarrow{D})$, já que δ_h^{id} é completo. No caso 2, se o predicado da cabeça de alguma cláusula $\overrightarrow{E}$ não aparece na cabeça de nenhuma das cláusulas de T'', então $\overrightarrow{E}$ pode ser criada pelo operador de Adição de Novas Regras, senão, seja $\overrightarrow{E}'$ a cláusula contendo apenas os literais em comum entre $\overrightarrow{E}$ e alguma cláusula $\overrightarrow{D}$ de T''. $\overrightarrow{E}'$ pode ser obtida por exclusão de antecedentes em uma cópia de $\overrightarrow{D}$, mantendo-se a cláusula $\overrightarrow{D}$ original na teoria, que é o primeiro passo de refinamento do operador de Adição de Regras. Para obter $\overrightarrow{E}$ a partir de $\overrightarrow{E}'$ podemos usar o

segundo passo do operador de Adição de Regras, que é o operador de refinamento de cláusulas de Adição de Antecedentes, o qual já foi provado ser completo. Como existe um caminho de tamanho finito a partir de T'' até T' (ou a uma teoria equivalente a T') no grafo de refinamento, o operador δ_T é **completo**. □

4.4.2 Uma solução para tornar os operadores de refinamento de teorias do FORTE_MBC próprios

Como já visto na Seção 4.3, em ambos os operadores de teoria, a culpa deles não serem próprios é do refinamento correspondente ao Adição de Antecedentes, que permite a geração de teorias redundantes. Para tornar os operadores próprios, podemos aplicar a mesma solução proposta por Fanizzi [13]: antes de aplicar qualquer passo de refinamento do operador (de especialização ou de generalização) em uma teoria T, a teoria T é transformada em uma teoria não redundante T_{nr} equivalente.

No caso do operador de especialização de teorias do FORTE_MBC, esta transformação aconteceria no primeiro passo de refinamento, depois da inclusão de cada nova especialização retornada pelo operador de refinamento de cláusulas de Adição de Antecedentes. Para isto, é suficiente verificar se a nova especialização $\overrightarrow{C}$ introduzida em alguma teoria T é subconjunto ou superconjunto de alguma especialização $\overrightarrow{D}$ inserida anteriormente. Se for o caso, a cláusula mais específica entre $\overrightarrow{C}$ e $\overrightarrow{D}$ deve ser removida. O exemplo a seguir ilustra que essa transformação não afeta o processo de revisão:

Exemplo 9. *Suponha a mesma teoria T_2 usada no exemplo que ilustra que o operador ρ_T não é próprio (Exemplo 3):*

$$
\begin{aligned}
T_2 : \overrightarrow{C_1} &= great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\
&\quad r\_subst\_2(A,E), ring\_substitutions(A,D).\\
\overrightarrow{C_2} &= great\_ne(A,B) \leftarrow alk\_groups(A,C), gt(C,D), alk\_groups(B,D),\\
&\quad r\_subst\_2(A,E), x\_subst(B,F,G), ring\_substitutions(A,D).\\
\overrightarrow{C_3} &= great\_ne(A,B) \leftarrow ring\_subst\_3(B,C), ring\_substitutions(A,D),\\
&\quad ring\_substitutions(B,E), gt(E,D), alk\_groups(A,E).\\
\overrightarrow{C_4} &= great\_ne(A,B) \leftarrow ring\_substitutions(A,C), gt(C,D),\\
&\quad ring\_subst\_2(A,E).
\end{aligned}
$$

As cláusulas $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ foram as duas cláusulas introduzidas em T_2 pela aplicação do passo de refinamento correspondente ao Adição de Antecedentes. Suponha que $\overrightarrow{C_2}$ foi introduzida primeiro. Se a solução proposta acima for aplicada, $\overrightarrow{C_2}$, que é mais específica que $\overrightarrow{C_1}$, seria removida no momento em que $\overrightarrow{C_1}$ fosse colocada na teoria. Isto não é um problema porque, se for necessário, $\overrightarrow{C_2}$ pode ser gerada em

outro momento do processo de revisão, simplesmente especializando a cláusula $\overrightarrow{C_1}$ *pela introdução do literal* $x\_subst(B, F, G)$ *a partir de alguma Bottom Clause. A Bottom Clause necessária é garantida de existir, já que o exemplo positivo para criá-la é também provado por* $\overrightarrow{C_1}$*, e era provado pela cláusula* $\overrightarrow{C_2}$ *previamente removida.*

Ao considerar esta solução, as cláusulas inseridas pelo operador na teoria não podem ser subconjuntos ou superconjuntos de cláusulas que já estavam na teoria antes da revisão começar, mas apenas de cláusulas inseridas durante a aplicação do operador de refinamento, já que agora começamos a revisão a partir de uma teoria não redundante. No exemplo anterior, $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ foram geradas a partir de alguma cláusula $\overrightarrow{C}$, i.e, elas são superconjuntos de $\overrightarrow{C}$, mas $\overrightarrow{C}$ não é mantida na teoria. Se a teoria anterior à revisão não é redundante, então $\overrightarrow{C}$ não é subconjunto nem superconjunto de $\overrightarrow{C_2}$ e $\overrightarrow{C_3}$, e portanto também não o são as cláusulas $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$.

No caso do operador de generalização de teorias do FORTE_MBC, esta transformação aconteceria nos refinamentos correspondentes ao Adição de Regras e Adição de Novas Regras, também no momento em que fosse inserir uma nova especialização retornada pelo Adição de Antecedentes. Podemos usar a mesma abordagem anterior, porém, a verificação a ser feita nas cláusulas inseridas é um pouco diferente da que é feita no caso do operador de especialização. Observe o exemplo a seguir:

Exemplo 10. *Suponha a mesma teoria* T_4 *usada no exemplo que ilustra que o operador* δ_T *não é próprio (Exemplo 6):*

$$
\begin{aligned}
T_4 : \overrightarrow{C_5} &= great\_ne(A, B) \leftarrow alk\_groups(A, C), r\_subst\_2(A, D).\\
\overrightarrow{C_4} &= great\_ne(A, B) \leftarrow alk\_groups(A, C), r\_subst\_2(A, D),\\
&ring\_subst\_3(A, E), ring\_substitutions(A, C), ring\_subst\_3(B, F).\\
\overrightarrow{C_3} &= great\_ne(A, B) \leftarrow ring\_subst\_3(B, C), ring\_substitutions(A, D),\\
&ring\_substitutions(B, E), gt(E, D), alk\_groups(A, E).\\
\overrightarrow{C_2} &= great\_ne(A, B) \leftarrow alk\_groups(A, C), gt(C, D), alk\_groups(B, D),\\
&r\_subst\_2(A, E).\\
\overrightarrow{C_1} &= great\_ne(A, B) \leftarrow ring\_substitutions(A, C), gt(C, D),\\
&ring\_subst\_2(A, E).
\end{aligned}
$$

As cláusulas $\overrightarrow{C_4}$ *e* $\overrightarrow{C_5}$ *foram introduzidas em* T_4 *aplicando-se o operador de Adição de Regras na cláusula* $\overrightarrow{C_3}$*, que foi mantida na teoria. Observe que, neste exemplo, a cláusula* $\overrightarrow{C_5}$ *é subconjunto da cláusula* $\overrightarrow{C_4}$*, que foi introduzida num passo anterior do operador de Adição de Regras, mas também é subconjunto da cláusula* $\overrightarrow{C_2}$ *(renomeando-se a variável* D *para* E *no literal* $r\_subst\_2(A, D)$*), que já estava na teoria antes de se começar a revisão.*

Esta situação pode acontecer porque, no caso do operador de Adição de Regras, as novas cláusulas introduzidas não são necessariamente superconjuntos da cláusula

original que é mantida na teoria. Elas começam a partir de uma cláusula temporária (originada pela exclusão de antecedentes de uma cópia da cláusula original), e qualquer literal vindo de uma determinada Bottom Clause pode ser adicionado (mesmo que sejam literais que também aparecem nas outras cláusulas da teoria). Portanto, estas novas cláusulas podem ser subconjuntos ou superconjuntos de qualquer cláusula que já está na teoria.

Devido à situação exemplificada anteriormente, toda vez que for inserir uma nova cláusula, aplicando-se o Adição de Regras ou o Adição de Novas Regras, é necessário verificar se a nova cláusula $\overrightarrow{C}$ inserida em alguma teoria T, é subconjunto ou superconjunto de alguma cláusula $\overrightarrow{D}$ da teoria como um todo. Se for o caso, a cláusula mais específica entre $\overrightarrow{C}$ e $\overrightarrow{D}$ deve ser removida.

Ao considerar a solução acima, ambos os operadores de teorias passam a ser próprios para $\langle 2^{L_\perp(M)}, \succeq_\perp \rangle$. Como eles já eram localmente finitos e completos (no caso do operador de generalização, após considerar δ_h^{id}), eles agora são **ideais** para o reticulado definido.

4.5 Trabalhos relacionados

Nesta seção discutimos alguns trabalhos relacionados, mostrando porque os espaços de busca de cláusulas e teorias do FORTE_MBC não podem ser descritos pela ordem geral de *θ-subsumption*, ou por ordens de generalidade restritas e alternativas baseadas em *θ-subsumption*, citadas na introdução.

4.5.1 *θ-subsumption*

O espaço de busca do FORTE_MBC não pode ser descrito pela ordem geral de *θ-subsumption*. Uma razão para isso é que o FORTE_MBC apresenta a mesma incompletude do sistema Progol, que também limita o espaço de busca de cláusulas por uma *Bottom clause*. Tal incompletude é devido ao fato de que, quando selecionando literais da *Bottom clause* para adicionar a uma cláusula, cada literal pode ser selecionado apenas uma vez. Outra razão é que o FORTE_MBC não permite substituições de variáveis nos literais vindos da *Bottom clause*. O exemplo 11 abaixo, adaptado de [42], ilustra essas incompletudes:

Exemplo 11. *[42] Seja* $\overrightarrow{C} = p(X) \leftarrow q(X, X)$ *a cláusula sob revisão em alguma teoria corrente* T*, e* $\overrightarrow{\perp} = p(X) \leftarrow q(X, X), r(X, Y), s(Y, Z)$ *a Bottom Clause criada a partir de* $\overrightarrow{C}$ *e algum exemplo positivo provado pela mesma. Considere modos* $p(+)$ *,* $q(+, -)$*,* $r(+, -)$ *e* $s(+, -)$ *para os predicados p, q, r e s. Aplicando-se os operadores de refinamento Exclusão de Antecedentes, Adição de Antecedentes ou Adição de Regras do FORTE_MBC os possíveis refinamentos de* $\overrightarrow{C}$ *obtidos são:*

$\overrightarrow{C_1} = p(X) \leftarrow$. *(Aplicando-se Exclusão de Antecedentes)*
$\overrightarrow{C_2} = p(X) \leftarrow q(X,X), r(X,Y)$. *(Aplicando-se Adição de Antecedentes)*
$\overrightarrow{C_3} = p(X) \leftarrow q(X,X), r(X,Y), s(Y,Z)$. *(Aplicando-se Adição de Antecedentes)*
$\overrightarrow{C_4} = p(X) \leftarrow r(X,Y)$.
$\overrightarrow{C_5} = p(X) \leftarrow r(X,Y), s(Y,Z)$.
($\overrightarrow{C_4}$ e $\overrightarrow{C_5}$ geradas a partir do Adição de Regras, com $\overrightarrow{C}$ mantida na teoria)

No entanto as seguintes cláusulas que θ-subsumem $\overrightarrow{\perp}$ não são consideradas pelos refinamentos do FORTE_MBC:

$\overrightarrow{D_1} = p(X) \leftarrow q(X,Y), q(Y,X),$
$\overrightarrow{D_2} = p(X) \leftarrow q(X,Y), q(Y,Z), q(Z,X),$
$\overrightarrow{D_3} = p(X) \leftarrow q(X,Y), q(Y,Z), q(Z,W), q(W,Y),$
.....

Neste exemplo a cláusula $\overrightarrow{D_i}$ só pode ser construída se mais de um literal de $\overrightarrow{D_i}$ puder ser mapeado no mesmo literal $q(X,X)$ de $\overrightarrow{\perp}$ aplicando-se um conjunto de substituições. Porém, tanto o uso do mesmo literal da Bottom Clause como a aplicação de substituições nos literais vindo da Bottom Clause não são permitidos no FORTE_MBC.

4.5.2 Ordens de generalidade restritas baseadas em θ-subsumption

O sistema Progol também limita o seu espaço de busca de cláusulas por uma *Bottom Clause*, tal que dada uma cláusula ordenada $\overrightarrow{C}$ e uma $\overrightarrow{\perp}$ em $L_i(M)$, $\overrightarrow{C}$ está na linguagem do Progol se existe uma substituição θ tal que $\overrightarrow{C}\theta$ é subconjunto ordenado de $\overrightarrow{\perp}$ [42]. Para caracterizar este espaço de refinamento, em [42] foi introduzida a ordem de *subsumption* relativa a uma *Bottom Clause*, propondo-se operadores de refinamento ideais. Nesta ordem de generalidade, dadas duas cláusulas ordenadas $\overrightarrow{C}$ e $\overrightarrow{D}$ da linguagem de cláusulas definida para o Progol, $\overrightarrow{C}$ subsume $\overrightarrow{D}$ relativa a uma *Bottom Clause* se existe uma substituição θ tal que $\overrightarrow{C}\theta$ é um subconjunto ordenado de $\overrightarrow{D}$ relativa a uma *Bottom Clause*.

No Progol, refinamentos de uma cláusula são construídos pela adição de literais que são generalizações de um determinado literal de $\overrightarrow{\perp}$. Isso significa que se uma cláusula $\overrightarrow{C}$ subsume $\overrightarrow{D}$ relativa a uma $\overrightarrow{\perp}$, para cada literal L_i de $\overrightarrow{C}$ que é mapeado em um literal equivalente M_j de $\overrightarrow{D}$, tanto L_i quanto M_j são generalizações do mesmo literal da $\overrightarrow{\perp}$.

Quando escolhendo um literal da *Bottom Clause* para adicionar na cláusula, o operador de Adição de Antecedentes do FORTE_MBC considera o literal como ele

está na *Bottom Clause*, como é feito de forma padrão no Aleph, isto é, ele não aplica substituições θ antes de adicionar um literal a uma cláusula, como é feito no Progol. O Progol usa substituições θ para permitir *variable splitting*, já que existem teorias que não podem ser achadas caso o *variable splitting* não seja considerado. *Variable splitting* é uma configuração opcional da fase de saturação do Aleph, quando literais de igualdade entre variáveis são inseridos na *Bottom Clause* para manter a equivalência [42]. Já que o FORTE_MBC aplica o método de saturação do Aleph, ele também pode permitir *variable splitting* setando-se o parâmetro mencionado. Se o mesmo é considerado, o θ será usado apenas na geração da *Bottom Clause*, mas não quando for selecionar um literal desta. O *exemplo 12* abaixo, adaptado de [42], ilustra o uso de *variable splitting* no FORTE_MBC:

Exemplo 12. *[42] Seja* $\overrightarrow{C} = p(X,Y) \leftarrow q(X,X)$ *a cláusula sob revisão em alguma teoria corrente* T*. Considere os modos* $p(+,-), q(+,+), q(+,-), q(-.+)$ *e* $q(-,-)$ *para os predicados* p *e* q*, profundidade de variável* i*=2 e conhecimento preliminar* $B = \{q(t1,t1) \leftarrow, q(t2,t2) \leftarrow\}$*. Assuma que o exemplo usado para gerar a Bottom Clause é* $e = p(t1,t2) \leftarrow$*. Considerando-se o parâmetro de variable splitting, a Bottom Clause* $\overrightarrow{\perp}$ *criada a partir de* $\overrightarrow{C}, e$ *e* B *é:*

$$\begin{aligned}\overrightarrow{\perp} = p(X,Y) \leftarrow\ & q(X,X), q(X,V), V = X, q(Z,X), Z = X, q(S,T), T = S, T = X,\\ & S = X, q(U,W), W = U, W = Y, U = Y, q(W,W), q(Y,W), q(W,U), q(U,U),\\ & q(Y,U), q(W,Y), q(U,Y), q(Y,Y), q(Y,R), R = Y, q(M,Y), M = Y,\\ & q(K,L), L = K, L = Y, K = Y.\end{aligned}$$

Usando-se $\overrightarrow{\perp}$*, Os refinamentos no FORTE_MBC podem gerar as cláusulas* $\overrightarrow{C_1}$*,* $\overrightarrow{C_2}$ *e* $\overrightarrow{C_3}$ *(que é o lgg (least general generalization) de* $\overrightarrow{C_1}$ *e* $\overrightarrow{C_2}$*) abaixo:*

$\overrightarrow{C_1} = p(X,Y) \leftarrow q(X,X), q(Y,W)$. *(Aplicando-se Adição de Antecedentes)*
$\overrightarrow{C_2} = p(X,Y) \leftarrow q(Z,X), q(Y,Y)$. *(Aplicando-se Adição de Regras)*
$\overrightarrow{C_3} = p(X,Y) \leftarrow q(Z,X), q(U,U), q(Y,W)$. *(Aplicando-se Adição de Regras)*

Entretanto, se variable splitting não é considerado, a Bottom Clause criada a partir de $\overrightarrow{C}, e$ *e* B *seria* $\overrightarrow{\perp_1} = p(X,Y) \leftarrow q(X,X), q(Y,Y)$*. Neste caso, seriam necessárias substituições* $\{Y \setminus W\}$ *e* $\{X \setminus Z\}$ *em* $\overrightarrow{\perp_1}$ *para gerar* $\overrightarrow{C_1}$ *e* $\overrightarrow{C_2}$*, respectivamente. Mesmo que o FORTE_MBC fosse capaz de aplicar tais substituições, ele não poderia criar a cláusula* $\overrightarrow{C_3}$*, já que isto exigiria que mais de um literal de* $\overrightarrow{C_3}$ *fosse mapeado no mesmo literal de* $\overrightarrow{\perp_1}$*.*

A cláusula $\overrightarrow{C_3}$, que é o lgg de $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$ só pode ser inserida na teoria pela aplicação do operador de Adição de Regras, e apenas se existir uma *Bottom Clause* que possua os literais de $\overrightarrow{C_3}$. Nós não podemos garantir que o lgg de quaisquer duas

cláusulas em $L_{\perp}(M)$ também está em $L_{\perp}(M)$. No exemplo anterior, $\overrightarrow{C_3}$ só está na linguagem por causa do modo $q(-,-)$ usado para gerar o literal $q(U,U)$. Além disso, o lgg de duas cláusulas pode ser uma cláusula que não necessariamente obedece aos modos definidos, ou que é subconjunto ordenado de alguma *Bottom Clause*. Ao introduzirmos a *Bottom Clause* e declaração de modos no FORTE_MBC optamos por eficiência mesmo restringindo o espaço de busca, tal que as cláusulas neste espaço dependem dos modos definidos, dos exemplos e do conhecimento preliminar. Observe que as cláusulas $\overrightarrow{D_i}$ do *Exemplo 11* não podem ser criadas mesmo que o *variable splitting* seja considerado.

No Exemplo 12 podemos ver a diferença com relação ao Progol. O Progol não usa *variable splitting* na geração da *Bottom Clause*, portanto, neste exemplo, a *Bottom Clause* usada pelo Progol seria $\overrightarrow{\perp_1}$. A partir desta *Bottom Clause*, o Progol seria capaz de gerar as cláusulas $\overrightarrow{C_1}$ e $\overrightarrow{C_2}$, usando-se substituições θ para implementar *variable splitting*. Porém o Progol não poderia gerar a cláusula $\overrightarrow{C_3}$, porque isso ainda exigiria que mais de um literal de $\overrightarrow{C_3}$ fosse mapeado no mesmo literal de $\overrightarrow{\perp}$. Apesar do fato de o Progol aplicar substituições θ nos literais da *Bottom Clause*, ele apenas seleciona o literal uma única vez, considerando apenas uma única substituição possível. Portanto, se a *Bottom Clause* tiver três literais, a cláusula gerada terá no máximo três literais também. Neste sentido, ao permitir *variable splitting* na geração da *Bottom Clause*, o FORTE_MBC se torna mais flexível e com um espaço de busca de cláusulas maior.

Enquanto em [42] é estudado o espaço de cláusulas limitado por uma *Bottom Clause*, em [20] é discutido o problema de indução de teorias clausais através de um espaço de busca consistindo de teorias, tal que este espaço é limitado por uma teoria mais específica, chamada de Teoria *Bottom*, uma teoria que é um conjunto de *Bottom Clauses*. Como o espaço de busca de teorias pode ser muito grande, em [20] foi imposta uma restrição de que todas as cláusulas em uma teoria devem ser relevantes. Considerando essa restrição, cada cláusula em uma teoria T, tal quet $B \cup T \models E$, onde B é o conhecimento preliminar e E é o conjunto de exemplos positivos, tem que θ-subsumir alguma cláusula na Teoria *Bottom* definida. Portanto, para todas as cláusulas $\overrightarrow{C} \in T$ existe uma *Bottom Clause* $\perp$ na Teoria *Bottom* tal que $\overrightarrow{C}$ θ-subsume $\perp$. Ao introduzir o conceito de Teoria *Bottom*, Midelfart [20] propôs um espaço de busca restrito que ainda é ordenado pela *θ-subsumption* mas com uma linguagem mais restrita: a linguagem de teorias subsumindo alguma Teoria *Bottom* específica.

No FORTE_MBC não existe uma Teoria *Bottom* gerada a priori antes de uma determinada teoria ser revisada. Entretanto, existe um conjunto de *Bottom Clauses* que podem ser criadas a partir de um conjunto de exemplos positivos provados pela teoria corrente sendo revisada, mas tal conjunto depende de quais cláusulas da

teoria são marcadas como pontos de revisão e de quais operadores de refinamento serão aplicados. Mesmo que o FORTE_MBC considerasse esse conjunto de possíveis *Bottom Clauses* como sendo a Teoria *Bottom* tal como definida em [20], esta teoria não seria o limite inferior do reticulado de teorias. Como, no FORTE_MBC, é possível deletar qualquer cláusula da teoria corrente através do operador de Exclusão de Regras, a teoria mais específica que se pode obter é a teoria vazia, e não a Teoria *Bottom*. Se no FORTE_MBC considerássemos uma Teoria *Bottom*, esta seria apenas uma teoria de referência, no sentido de que quando uma cláusula da teoria corrente precisa ser especializada, a *Bottom Clause* a ser usada estaria na Teoria *Bottom* correspondente.

O próximo exemplo mostra duas teorias: uma que não pode ser gerada no FORTE_MBC e uma que pode mas não está na linguagem restrita definida em [20]. Neste exemplo iremos considerar uma teoria corrente T que é relevante, e uma Teoria *Bottom* $\bot_T$ tal que cada cláusula em T subsume alguma *Bottom Clause* em $\bot_T$.

Exemplo 13. *Seja T a teoria corrente e $\bot_T$ a Teoria Bottom, como se segue:*

$$T : \overrightarrow{C} = p(X) \leftarrow q(X,X).$$
$$\overrightarrow{D} = p(X) \leftarrow r(X,Y).$$

$$\bot_T : \overrightarrow{\bot_C} = p(X) \leftarrow q(X,X), s(X,Y), s(Y,Z)$$
$$\overrightarrow{\bot_D} = p(X) \leftarrow r(X,Y), s(Y,Z), s(Z,X).$$

onde $\overrightarrow{\bot_C}$ e $\overrightarrow{\bot_D}$ são Bottom Clauses criadas a partir de exemplos positivos provados pelas cláusulas $\overrightarrow{C}$ e $\overrightarrow{D}$, respectivamente. Considere os modos $p(+), q(+,-), r(+,-)$ e $s(+,-)$ para os predicados p, q, r e s. A teoria T_1 abaixo não poderia ser criada aplicando-se apenas os operadores de especialização do FORTE_MBC:

$$T_1 : \overrightarrow{C_1} = p(X) \leftarrow q(X,X), q(X,Y), q(Y,Z), q(Z,W), q(W,Y),$$
$$\overrightarrow{D_1} = p(X) \leftarrow r(X,Y), s(Y,Z).$$

A cláusula $\overrightarrow{C_1} \in T_1$ é uma especialização da cláusula $\overrightarrow{C}$, mas só pode ser criada se mais de um literal desta for mapeado no mesmo literal $q(X,X)$ de $\overrightarrow{\bot_C}$. Entretanto, temos que T θ-subsume T_1, pois para cada cláusula em T_1 temos alguma cláusula em T que a θ-subsume. Além disso, tanto T quanto T_1 θ-subsumem $\bot_T$. Portanto, T_1 não está no espaço de busca do FORTE_MBC, mas está no espaço definido em [20], tendo $\bot_T$ como Teoria Bottom. Como o espaço de busca de cláusulas do FORTE_MBC não é ordenado pela θ-subsumption, o espaço de busca de teorias também não pode ser ordenado por θ-subsumption, como em [20].

Agora considere a teoria $T_2 = \{\overrightarrow{D}\}$ *gerada a partir de* T *pela exclusão da cláusula* $\overrightarrow{C}$ *através do operador de especialização de Exclusão de Regras do FORTE_MBC. A teoria* T_2 *é uma possível teoria gerada pelo FORTE_MBC, mas não está na linguagem definida por [20]. Isto é verdade já que* T_2 *não subsume mais a Teoria Bottom* $\perp_T$*, pois a Bottom Clause* $\overrightarrow{\perp_C} \in \perp_T$ *não é subsumida por nenhuma cláusula de* T_2*.*

Midelfart [20] também definiu um operador de especialização de teorias no qual também se tem os passos de refinamentos referentes a: incluir uma especialização de uma cláusula tal que a cláusula original é removida da teoria; deletar uma cláusula da teoria. Porém a exclusão de cláusula não é feita da mesma forma que no FORTE_-MBC, pois não se pode deletar qualquer cláusula, apenas aquelas que não são as únicas a subsumir uma determinada *Bottom Clause* da Teoria *Bottom*. Dessa forma, a teoria resultante não deixa de subsumir a Teoria *Bottom*.

Outros trabalhos propuseram operadores de refinamento ideais para ambos os espaços de especialização e generalização de teorias, considerando um modelo restrito baseado em θ-*subsumption*, chamado θ_{OI}-*subsumption* [11–13]. Esta ordem de generalidade restringe a *subsumption* ao princípio de *Object Identity* (OI), que significa que em uma cláusula, termos denotados por símbolos diferentes devem ser distintos, i.e., representam entidades diferentes do domínio. Porém, o espaço de busca do FORTE_MBC e seus operadores de refinamento não podem ser caracterizados por esta ordem, pois o princípio de *OI* não é verdade no FORTE_MBC. Para ilustrar isso, basta considerar o parâmetro de *variable splitting* ao criar uma *Bottom Clause*. Dessa forma, literais de igualdade serão adicionados à *Bottom Clause*, e tais literais se tornam possíveis antecedentes a serem adicionados a uma cláusula sob revisão. Tais literais contradizem o princípio de *OI*, já que diferentes variáveis estão denotando o mesmo termo. Mesmo que o princípio do *OI* fosse verdade para o FORTE_MBC, o espaço de busca não poderia ser ordenado pela θ_{OI}-*subsumption*, pois a mesma incompletude com relação à θ-*subsumption* apresentada antes valeria neste caso.

Capítulo 5

FORTE_BETH: Implementação no FORTE_MBC da abordagem proposta no sistema BETH

5.1 Introdução

Quando nos referimos a bases de dados muito grandes em ILP, geralmente estamos falando do enorme número de exemplos de treinamento. Porém outra medida de complexidade que é particularmente relevante é o tamanho dos exemplos, ou seja, o número de fatos usados para descrever os exemplos. Em sistemas como o Progol, Aleph e o FORTE_MBC, que usam a *Bottom Clause* como espaço de busca de antecedentes, a *Bottom Clause* pode ser muito grande, dependendo do número de átomos básicos usados para descrever os exemplos. Para limitar o tamanho da *Bottom Clause*, são usados parâmetros como:

- **limite de profundidade de variável** i: determina a profundidade máxima que a árvore de construção da *bottom clause* pode atingir, onde em cada nível da árvore novas variáveis são criadas a partir das variáveis do nível anterior.
- **definições de modos**: descrevem as relações (predicados) entre objetos de dados e tipos desses dados. Estas declarações permitem também informar se a relação pode ser utilizada na cabeça (declarações *modeh*) ou no corpo (declarações *modeb*) das regras geradas. Na declaração de modos também são descritos o tipo dos argumentos e o número máximo de instanciações de cada predicado (*recall* r).

Apesar da *Bottom Clause* gerada por estes sistemas ser limitada pelos parâmetros dados, sua complexidade é exponencial com respeito à profundidade de variável i, o que pode resultar num espaço de busca de cláusulas duplamente exponencial, visto que o tamanho deste reticulado, que é limitado por uma *Bottom Clause* $\overrightarrow{\perp}$, é 2^n, onde n = $|\overrightarrow{\perp}|$.

Em problemas onde os exemplos são muito grandes, o conhecimento preliminar(BK) pode conter muitos átomos básicos ou gerar muitos átomos básicos a partir de suas regras. Quando uma *Bottom Clause* é criada nestes sistemas, ela inclui todos os literais gerados a partir dos átomos básicos relevantes para um determinado exemplo (de acordo com os parâmetros impostos), porém, nem todos estes literais serão de fato avaliados como possíveis candidatos a serem adicionados na cláusula sendo revisada, devido à restrição de modos desta cláusula.

Além disso, como vimos no Capítulo 4, podemos usar o parâmetro de *variable splitting* como uma configuração opcional na fase de saturação do Aleph, quando literais de igualdade entre variáveis são inseridos na *Bottom Clause* para manter a equivalência [42]. Porém, introduzir literais de igualdade na *Bottom Clause* aumenta o espaço de busca consideravelmente. De acordo com [41], se *variable splitting* é setado, a *Bottom Clause* pode ser tornar extremamente grande para um número grande de co-referências de variáveis. No entanto, muitos destes literais podem, também, não serem avaliados para serem adicionados à cláusula que está sendo revisada.

Para lidar com este problema da complexidade exponencial da *Bottom Clause*, em [44] foi proposta uma abordagem chamada BETH que combina as abordagens top-down e bottom-up para gerar uma *Bottom Clause* que contém apenas os literais relevantes para o processo de construção de uma cláusula, ou seja, apenas os literais que poderiam ser adicionados à cláusula de acordo com os modos desta. A complexidade desta nova *Bottom Clause* deixou de ser exponencial e passou a ser linear com respeito ao tamanho n máximo permitido para uma cláusula, onde $n \geq i$.

Um ponto forte da abordagem top-down é que a geração dos literais é direcionada pela busca heurística, sendo que apenas são gerados os literais que de fato podem ser introduzidos na cláusula. Portanto, existe uma relação entre o conjunto de literais que podem ser incluídos numa *Bottom Clause* e a busca por uma boa cláusula. No BETH, a busca é usada como guia para selecionar um subconjunto relevante de átomos básicos do BK (ou gerados pelo BK através de regras) para serem incluídos na *Bottom Clause*.

Um ponto forte da abordagem bottom-up é o uso da *Bottom Clause* como espaço de busca de literais a serem considerados na especialização de uma cláusula. Estes literais são criados usando-se átomos básicos que descrevem um exemplo positivo dado, de forma que é garantido que a cláusula resultante da especialização irá cobrir

pelo menos um exemplo positivo.

O BETH tira vantagem dos pontos fortes destas duas abordagens, combinando-as em uma só. Nesta nova abordagem, a *Bottom Clause* não mais é construída a partir de um exemplo positivo, antes de se começar a busca por uma boa cláusula. Ao invés disso, depois que o exemplo positivo é escolhido, os literais são gerados de forma top-down, ou seja, guiados pela busca heurística, com a diferença de que os literais gerados são restritos àqueles que cobrem o exemplo positivo escolhido.

Para verificar as vantagens do novo algoritmo, o BETH foi comparado ao sistema Aleph, usando uma base de dados conhecida como Link Discovery, que na época em que o BETH foi proposto (em 2003) era uma das maiores bases em termos de número de átomos básicos no BK. Os experimentos apresentados em [44] mostraram que, considerando o conjunto de treinamento, o BETH treinou 25x mais rápido que o Aleph, gerando uma *Bottom Clause* 119x menor do que a gerada pelo Aleph.

Motivados por estes resultados, propomos o sistema FORTE_BETH, a partir da implementação da abordagem proposta pelo BETH no FORTE_MBC, de forma a reduzir o tamanho da *Bottom Clause* gerada e por consequência o tempo de especialização de uma cláusula, objetivando um processo de revisão mais eficiente.

Nas seções a seguir serão descritas, respectivamente, o algoritmo em detalhes proposto pelo BETH, a comparação de complexidades do tamanho da *Bottom Clause* no BETH e no Aleph (que vai ser a mesma do FORTE_MBC), o algoritmo FORTE_-BETH proposto e os resultados experimentais obtidos.

5.2 Algoritmo BETH [44]

O Sistema BETH combina as abordagens top-down e bottom-up de forma que o algoritmo não constrói a *Bottom Clause* a priori, mas a descobre durante a busca por uma boa cláusula.

O loop mais externo do BETH é simplesmente um algoritmo de cobertura como qualquer outro algoritmo típico de ILP:

- Acha uma boa cláusula que cobre um subconjunto não vazio de exemplos positivos;
- Remove, do conjunto de exemplos positivos, os exemplos positivos cobertos pela cláusula;
- Adiciona a nova cláusula à teoria (originalmente vazia);
- Repete os passos anteriores até que o conjunto inteiro de exemplos positivos sejam cobertos pela teoria;

- Retorna a teoria achada.

A busca por uma boa cláusula começa da cláusula mais geral, $\square = T \leftarrow true$, onde T é um predicado de mesmo nome e aridade do exemplo a ser escolhido, e a especializa adicionando-se um literal no seu corpo. O BETH permite que seja usado um *beam* de cláusulas de tamanho b, de forma a achar uma cláusula suficientemente boa. Assim, a cada especialização as b melhores cláusulas são selecionadas e colocadas no *beam*. Inicialmente o *beam* contém apenas a cláusula mais geral.

Adicionalmente, a *Bottom Clause* que limita o espaço de busca é computada. A *Bottom Clause* inicial é setada como $e \leftarrow true$, onde e é o exemplo escolhido aleatóriamente do conjunto de exemplos positivos. A *Bottom Clause* é expandida incrementalmente durante a busca por uma boa cláusula. Quando uma cláusula suficientemente boa de acordo com a heurística de busca é achada, a cláusula e a *Bottom Clause* são retornadas. O algoritmo que constrói uma cláusula é mostrado em Algoritmo 5.1.

Algoritmo 5.1 Algoritmo de Construção de uma Cláusula no BETH [44]

Dados um conjunto de especificações de predicados P dos predicados do BK, um *beam* Q de tamanho b, limite n de tamanho da cláusula, limite de profundidade de variável i, limite de *recall* r, um conjunto não-vazio de exemplos positivos Pxs, e um conjunto de exemplos negativos Nxs.

1: Escolha randomicamente um exemplo positivo $e \in Pxs$.
2: $\bot_0 = e \leftarrow true$.
3: $Q_0 = \{\square\}$.
4: $Q = Q_0$.
5: $\overrightarrow{\bot} = \overrightarrow{\bot_0}$.
6: **repita**
7: gera_refinamentos(Q; P; b; n; i; r; Pxs; Nxs; $\overrightarrow{\bot}$; Q'; $\overrightarrow{\bot'}$);
8: $Q = Q'$;
9: $\overrightarrow{\bot} = \overrightarrow{\bot'}$;
10: **até que** exista uma cláusula $\overrightarrow{C} \in Q$ que seja suficientemente acurada.
11: Retorne $\overrightarrow{C}$ e $\bot$.

A função chamada no Algoritmo 5.1 é responsável por gerar os possíveis refinamentos de cada cláusula no *beam*, ou seja, por adicionar literais às cláusulas de acordo com um conjunto de restrições de refinamento. Ao final, a função atualiza o *beam* com as b melhores cláusulas geradas, de acordo com alguma heurística de busca e retorna a *Bottom Clause* atualizada. A função $gera\_refinamentos$ está detalhada no Algoritmo 5.2.

Para achar os possíveis refinamentos de uma determinada cláusula $\overrightarrow{C_i}$ do *beam* primeiro é achada uma substituição θ que satisfaça o corpo da cláusula, então são construídos literais R_j (com variáveis aleatórias) para as especificações de predicados

Algoritmo 5.2 Gera_Refinamentos(Q; P; b; n; i; r; Pxs; Nxs; $\perp$; Q'; $\perp'$) [44]

1: Para cada cláusula $\overrightarrow{C_i}$ e para cada predicado $P_j \in P$, crie um literal R_j com variáveis aleatórias, onde R_j possui o mesmo nome e aridade de P_j.
2: Ache substituições θ, β tal que θ satisfaça $\overrightarrow{C_i}$, β satisfaça R_j, e $\overrightarrow{C_i}\theta$ e $R_j\beta$ satisfaçam as restrições de refinamento.
3: Considere no máximo r átomos básicos $R_j\beta$ para θ e β diferentes.
4: Para cada par de $\overrightarrow{C_i}\theta$ e $R_j\beta$ satisfazendo as restrições de refinamento, faça_literais($\overrightarrow{C_i}\theta$,$R_j\beta$,Lits) e adicione $R_j\beta$ ao corpo de $\overrightarrow{\perp}$.
5: Para cada $L \in Lits$, adicione L ao corpo de $\overrightarrow{C_i}$, formando $\overrightarrow{C_i'}$, e seja Q_i o conjunto de todas as $\overrightarrow{C_i'}$.
6: Avalie cada cláusula em Q_i por uma determinada heurística usando Pxs e Nxs.
7: Coloque apenas as b melhores cláusulas em Q'.
8: Seja $\overrightarrow{\perp'}$ a *Bottom Clause* resultante após adicionar todos os átomos básicos $R_j\beta$ ao corpo de $\perp$ para cada $\overrightarrow{C_i}$ e R_j.
9: Retorne Q' e $\overrightarrow{\perp'}$.

em P, e são achadas substituições β que tornem $R_j\beta$ um átomo básico, tal que $\overrightarrow{C_i}\theta$ e $R_j\beta$ satisfaçam as seguintes restrições de refinamento:

1) Um dos argumentos de $R_j\beta$ tem que aparecer em $\overrightarrow{C_i}\theta$, para garantir que a cláusula resultante é uma cláusula linkada.

2) $R_j\beta \notin \overrightarrow{C_i}\theta$ (para evitar literais repetidos).

São buscados pares de θ e β que satisfaçam as restrições de refinamento, sendo que no máximo r (recall) $R_j\beta$ átomos básicos distintos são usados. Para evitar achar repetidamente a mesma prova de uma determinada cláusula, o BETH mantém uma cache de provas para cada cláusula do *beam*. Se não forem encontrados $R_j\beta$ que satisfaçam as restrições de refinamento, um novo exemplo é escolhido aleatoriamente do conjunto de exemplos positivos e a busca recomeça. O BETH também permite que declarações de tipos e modos sejam usados para restringir os literais a serem gerados.

A função executada na linha 4 do Algoritmo 5.2 nada mais é do que uma função para retornar os literais variabilizados para serem avaliados, ou seja, dados a cláusula C, a substituição θ, e um átomo básico $R_j\beta$, a função substitui os argumentos no átomo básico por variáveis da cláusula que foram instanciadas (por θ) para estes argumentos do átomo básico, observando-se o limite de profundidade de variável. O algoritmo da função *faça_literais* é mostrado em Algoritmo 5.3.

Algoritmo 5.3 Faça_Literais [44]

Dada a cláusula $\overrightarrow{C}\theta$ da forma $e \leftarrow a_1, ..., a_n$ (onde $\overrightarrow{C}$ é a cláusula corrente sendo especializada, θ é a substituição que satisfaz $\overrightarrow{C}$, $e \in Pxs$, e $BK \models a_i$ para cada átomo básico a_i do corpo de $\overrightarrow{C}\theta$) e um átomo básico a_{n+1} tal que $BK \models a_{n+1}$.

1: Faça um conjunto de literais Lits tal que cada literal $L \in Lits$ possua o mesmo nome e aridade do átomo a_{n+1}, e tal que L satisfaça: 1) Suponha que a constante c_i é o i-ésimo argumento de a_{n+1}, e a variável V_i é o i-ésimo argumento de L. Se c_i aparece em $\overrightarrow{C}\theta$, então $c_i/V_i \in \theta^{-1}$; caso contrário, V_i é uma nova variável que não aparece em $\overrightarrow{C}$. 2) Não existe variável V em L tal que d(V) $>$ i, onde i é o limite de profundidade de variável.

2: Retorne Lits.

5.3 Comparação das Complexidades de Tamanho da *Bottom Clause*

Nesta seção resgataremos os teoremas enunciados em [44] que demonstram as complexidades das *bottom clauses* geradas, respectivamente pelo FORTE_MBC e pelo BETH, sendo a primeira exponencial com respeito ao limite de profundidade de variável i, e a segunda linear com respeito ao tamanho máximo da cláusula n, onde $n \geq i$.

5.3.1 Complexidade da *Bottom Clause* gerada no FORTE_MBC

No FORTE_MBC a *Bottom Clause* é construída levando-se em consideração a cláusula inicial a ser revisada, de forma que ela tem como ponto inicial esta cláusula, ou seja, as variáveis já existentes na cláusula possuem profundidade 0, ao contrário do Aleph que constrói a *Bottom Clause* do zero. No pior caso, a cláusula inicial será vazia e a *Bottom Clause* será a mesma gerada pelo Aleph. Porém, o ponto inicial de construção da *Bottom Clause* não modifica o fato dela ser exponencial com relação ao i. O teorema da complexidade abaixo se refere à complexidade da *Bottom Clause* no Progol, mas que é a mesma para o FORTE_MBC.

Teorema 15. *[24] Seja |M| a cardinalidade de M (conjunto de declarações de Modos). Suponha que o número de ocorrências de +type e -type em cada modeh é limitado pelas constantes j^- e j^+, respectivamente, e que o número de ocorrências de +type e -type em cada modeb é limitado pelas constantes j^+ e j^-, respectivamente. Suponha que o recall de cada mode $m \in M$ é limitado pela constante r. A cardinalidade de $\perp_i$ (a Bottom Clause gerada dado o limite de profundidade de variável i) é limitada por $(r|M|j^+j^-)^{ij^+}$.*

Demonstração. Os detalhes da prova deste teorema estão em [24] e serão omitidos aqui. □

O teorema mostra que a complexidade do tamanho da *Bottom Clause* construída pelo Progol/Aleph, e consequentemente pelo FORTE_MBC, é exponencial com respeito ao limite de profundidade de variável i, o que leva a buscar em um espaço duplamente exponencial.

5.3.2 Complexidade da *Bottom Clause* gerada no BETH

Seja $\perp(b, n, P, r, i)$ a *Bottom Clause* construída pelo algoritmo da seção anterior, dados os parâmetros b, n, P, r e i, que são o tamanho do beam, o tamanho máximo de cláusula, o conjunto de especificações de predicado, o limite de recall, e o limite de profundidade de variável, respectivamente.

Teorema 16. *[44] No pior caso, o tamanho de* $\perp(b, n, P, r, i)$ *é* $O(bn|P|r)$.

Demonstração. O número máximo de átomos básicos que 1) satisfazem as restrições de refinamento e 2) geram literais, observando o limite de profundidade de variável i, para qualquer cláusula sendo refinada é $|P|r$. Portanto, o número máximo de átomos básicos satisfazendo as restrições de refinamento depois de se adicionar n literais ao corpo da cláusula mais geral é $n|P|r$. Como se tem no máximo b cláusulas no *beam* em qualquer momento da busca, o número máximo de átomos básicos satisfazendo as restrições de refinamento é $bn|P|r$. Logo, no pior caso, a complexidade da *Bottom Clause* $\perp(b, n, P, r, i)$ é $O(bn|P|r)$. □

5.4 Algoritmo FORTE_BETH

O algoritmo FORTE_BETH proposto neste trabalho é uma combinação do algoritmo BETH com o de Adição de Antecedentes do FORTE_MBC. Como dito anteriormente, o FORTE_MBC utiliza a *Bottom Clause* como espaço de busca de literais a serem adicionados a uma cláusula sendo especializada. Ele utiliza o método de saturação do Aleph para gerar a *Bottom Clause*, de forma que esta, assim como no Aleph, é gerada a priori, antes que o processo de busca adicione literais à cláusula. Por causa disso, o reticulado de cláusulas é duplamente exponencial, visto que a complexidade de tamanho da *Bottom Clause* é exponencial. Visando tornar essa complexidade linear, assim como foi feito no BETH, alteramos o algoritmo de Adição de Antecedentes do FORTE_MBC para conter as alterações propostas pelo BETH, fazendo com que a *Bottom Clause* seja construída ao mesmo tempo em que a cláusula é especializada, incluíndo na *Bottom Clause* apenas os literais relevantes, ou seja, apenas os literais que realmente são avaliados durante o processo de revisão.

O Algoritmo 5.4 é uma reprodução do Algoritmo 3.2 de Adição de Antecedentes (apresentado no Capítulo 3.1) porém com as modificações necessárias à implementação do algoritmo FORTE_BETH. Tais modificações estão marcadas em negrito e são detalhadas a seguir:

- **Geração da primeira camada da *Bottom Clause* a partir da cláusula corrente $\overrightarrow{C}$ sendo revisada e de uma intância positiva I provada por esta**: Este passo é alterado no FORTE_BETH, pois, ao contrário do que é feito no algoritmo original de Adição de Antecedentes do FORTE_MBC, a *Bottom Clause* não é gerada a priori. No caso do FORTE_BETH, a *Bottom Clause* inicial é gerada utilizando-se o mesmo algoritmo de geração da *Bottom Clause* (Algoritmo A.1 descrito no **Apêndice A**), porém apenas uma iteração do algoritmo é executada, de forma que apenas uma camada de literais da *Bottom Clause* é gerada. Esta camada corresponde aos literais relevantes que são geradas a partir dos termos básicos que instanciaram a cláusula inicial.

- **Retorno dos possíveis antecedentes utilizando a $\overrightarrow{\perp}$ construída até o momento:** Como agora a *Bottom Clause* está sendo construída ao mesmo tempo em que a cláusula corrente vai sendo especializada, no momento que for buscar os literais da *Bottom Clause* para se adicionar na cláusula, a *Bottom Clause* só vai conter literais relevantes, ou seja, literais que se adicionados à cláusula resultariam em uma cláusula que respeita os modos definidos. Portanto, nesse passo do algoritmo, não precisamos mais verificar para cada literal da *Bottom Clause* se este respeita ou não aos modos quando adicionado à cláusula corrente. Todos os literais que estiverem na *Bottom Clause* vão ser considerados como possíveis literais a serem adicionados, desconsiderando apenas os literais que já foram adicionados anteriormente ou que já estavam na cláusula inicial.

- **Geração da próxima camada de literais da $\overrightarrow{\perp}$ a partir do antecedente escolhido:** Dado que um novo antecedente foi adicionado à cláusula, este antecedente pode ter introduzido novas variáveis que ainda não tinham aparecido no corpo da cláusula, sendo que estas são variáveis de saída no antecedente adicionado. Dessa forma, os literais da *Bottom Clause* que se deseja gerar são exatamente os literais que necessitam destas variáveis como entrada, de forma que uma futura inclusão destes literais na cláusula corrente resultaria em cláusulas obedecendo aos modos definidos. O algoritmo de geração desta próxima camada da *Bottom Clause* está definido em Algoritmo 5.5.

O Algoritmo 5.5 de geração da próxima camada da *Bottom Clause* é baseado no Algoritmo A.1 de Geração da *Bottom Clause* no FORTE_MBC, descrito no

Algoritmo 5.4 Algoritmo FORTE_BETH de Adição de Antecedentes *hill climbing* do FORTE_MBC

T = Teoria corrente
$\overrightarrow{C}$ = Cláusula corrente a ser revisada
CP = Conjunto de exemplos positivos provados pela cláusula
IP = Conjunto de exemplos negativos provados pela cláusula
$\overrightarrow{C\_original} = \overrightarrow{C}$
$CP\_original = CP$
$IP\_original = IP$

1: **repita**
2: Considere o primeiro exemplo I de CP
3: **Gere a primeira camada da *bottom clause* $\overrightarrow{\perp}$ a partir de $\overrightarrow{C}$ e I**
4: **repita**
5: Calcule a pontuação de $\overrightarrow{C}$ a partir de CP e IP
6: **Retorne os possíveis antecedentes utilizando $\overrightarrow{\perp}$**
7: Calcule as pontuações dos antecedentes retornados
8: **se** o antecedente com a melhor pontuação melhora a pontuação de $\overrightarrow{C}$ ao ser adicionado **então**
9: Adicione o antecedente à cláusula $\overrightarrow{C}$, formando a cláusula $\overrightarrow{C'}$
10: $NovoCP$ = Exemplos positivos provados por $\overrightarrow{C'}$
11: $NovoIP$ = Exemplos negativos provados por $\overrightarrow{C'}$
12: $\overrightarrow{C} = \overrightarrow{C'}$
13: $CP = NovoCP$
14: $IP = NovoIP$
15: **Gere a próxima camada de literais de $\overrightarrow{\perp}$ a partir do antecedente escolhido**
16: **até que** não se tenha mais antecedentes, ou o tamanho máximo permitido para a cláusula tenha sido atingido, ou a pontuação da cláusula não seja mais melhorada.
17: **se** a nova cláusula $\overrightarrow{C'}$ é diferente da cláusula original $\overrightarrow{C\_original}$ **então**
18: Acrescente a nova cláusula $\overrightarrow{C'}$ à teoria T, formando a nova teoria T'
19: Calcule o novo conjunto de exemplos positivos provados ($NovoCP\_Teoria$) e exemplos negativos provados ($NovoIP\_Teoria$) pela nova teoria
20: $\overrightarrow{C} = \overrightarrow{C\_original}$
21: $CP = CP\_original - NovoCP\_Teoria$
22: $IP = IP\_original$
23: $T = T'$
24: **senão**
25: escolha um novo exemplo positivo ainda não usado de CP e recomeçe o algoritmo a partir da geração de uma nova *Bottom Clause*
26: **até que** não consiga mais especializar a cláusula, ou até que todos os exemplos positivos provados pela cláusula original tenham sido provados pelas cláusulas criadas

Algoritmo 5.5 Algoritimo de Geração da próxima camada da *Bottom Clause*

l = literal que foi adicionado à cláusula corrente $\overrightarrow{C}$
$\overrightarrow{\perp}$ = *Bottom Clause* atual
$UsedTerms$ = conjunto de termos básicos que já foram usados como termos de entrada para gerar os literais que já estão em $\overrightarrow{\perp}$
i = profundidade de variável máxima $hash : Terms \rightarrow N$ é uma função que unicamente mapeia termos em variáveis diferentes.

1: $Terms$ = termos básicos que instanciaram l e que são do tipo $-Type$ na definição do modo para l
2: $InTerms = Terms - UsedTerms$
3: Remova de $InTerms$ todos os termos cuja profundidade de variável associada é igual a i
4: **para** cada declaração *modeb* de corpo b **faça**
5: **para** toda substituição possível θ de argumentos correspondendo a tipo $+Type$ por pelo menos 1 termo do conjunto $InTerms$ e por 0 ou mais termos do conjunto $UsedTerms$ **faça**
6: **repita**
7: **se** Prolog tem sucesso no objetivo b com substituição θ' **então**
8: **para** cada v/t em θ e θ' **faça**
9: **se** v corresponde a tipo $\sharp$ **então**
10: substitua v em b por t
11: **senão**
12: substitua v em b por v_k onde $k = hash(t)$
13: **se** v corresponde a tipo $-Type$ e $t \notin InTerms \cup UsedTerms$ **então**
14: Associe a v_k à profundidade de variável $(i' + 1)$, onde i' = máxima profundidade de variável entre os termos em θ
15: Adicione b a $\overrightarrow{\perp}$
16: **até que** se tenha executado o loop *Recall* vezes
17: Retorne $\overrightarrow{\perp}$ variabilizada usando-se a função $hash$.

Apêndice A, porém ,ao contrário deste, o Algoritmo 5.5 parte da *Bottom Clause* $\overrightarrow{\perp}$ parcialmente criada, e do último literal l adicionado à cláusula corrente $\overrightarrow{C}$.

O algoritmo começa por coletar os termos básicos que instanciaram l e que são do tipo $-Type$ na definição do modo para l, ou seja, que aparecem como termos de saída. Dentre estes termos, são considerados apenas aqueles que já não foram usados como termos de entrada na geração dos literais que já estão em $\overrightarrow{\perp}$. Dessa forma evitamos que literais já presentes em $\overrightarrow{\perp}$ sejam gerados novamente. Além disso, termos que já atingiram a profundidade de variável máxima também são descartados, pois qualquer novo termo que fosse introduzido a partir deste, teria uma profundidade de variável maior do que a máxima permitida. O conjunto final de termos a serem considerados é chamado de $InTerms$. A partir daí o comportamento do algoritmo é praticamente o mesmo do Algoritmo A.1, com algumas poucas diferenças:

- Para cada declaração *modeb* do corpo, representada por b, são buscadas substituições θ para os argumentos do tipo $+Type$, ou seja, para os argumentos de entrada. Nesta busca, precisamos garantir que pelo menos um argumento de entrada seja um termo do conjunto $InTerms$, porque senão estaríamos gerando literais que já foram gerados previamente.

- Após b ter sido retornado com sucesso pelo Prolog com uma substituição θ' é preciso verificar os termos de saída que instanciaram argumentos do tipo $-Type$ em b. Se um determinado termo gerado não tinha aparecido antes na *Bottom Clause* instanciada, ou seja, não está em $InTerms \cup UsedTerms$, é porque ele tem uma profundidade de variável maior do que a dos termos já existentes. Neste caso, deve-se associar a ele um valor de profundidade de variável uma a mais do que a máxima profundidade de variável entre os termos em Θ.

Cada literal b criado é adicionado à $\overrightarrow{\perp}$, e ao final, $\overrightarrow{\perp}$ é variabilizada usando-se a função *hash* que já guarda a relação variável/termo para cada termo de $\overrightarrow{\perp}$ desde o início da criação desta.

Representando-se a *Bottom Clause* como uma árvore, conforme falamos no Capítulo 3.1, o algoritmo FORTE_BETH irá gerar os níveis desta árvore aos poucos (não necessariamente todos os literais deste nível na árvore completa), ou seja, cada iteração gera um conjunto de literais, que são exatamente os que seriam selecionados no Algoritmo 3.2 para serem avaliados. Essa geração de cada nível tem semelhanças com o algoritmo de geração de refinamentos do BETH (Algoritmo 5.2), seguindo os mesmo passos apresentados lá, porém considerando os modos para restringir os possíveis literais.

A seguir mostramos um exemplo de como algoritmo FORTE_BETH funciona quando vai especializar uma determinada cláusula inicial marcada como ponto de

revisão. Neste exemplo estamos considerando o domínio do Alzheimer [14], cujos modos estão definidos no **Apêndice B**. Assuma o limite de profundidade i = 3.

Exemplo 14. *Seja* $\overrightarrow{C} = great\_ne(A,B) \leftarrow alk\_groups(B,C)$ *a cláusula a ser especializada, e* $great\_ne(kk1, bb1)$ *o exemplo escolhido para gerar a Bottom Clause. Se tivéssemos usando o algoritmo de Adição de Antecedentes do FORTE_MBC, que constrói a Bottom Clause a priori, a Bottom Clause* $\overrightarrow{\perp}$ *gerada teria 59 literais.* $\overrightarrow{\perp}$ *está representada na Figura 5.1 em formato de árvore, onde cada camada representa o conjunto de literais que são gerados a partir das variáveis novas introduzidas na camada anterior.*

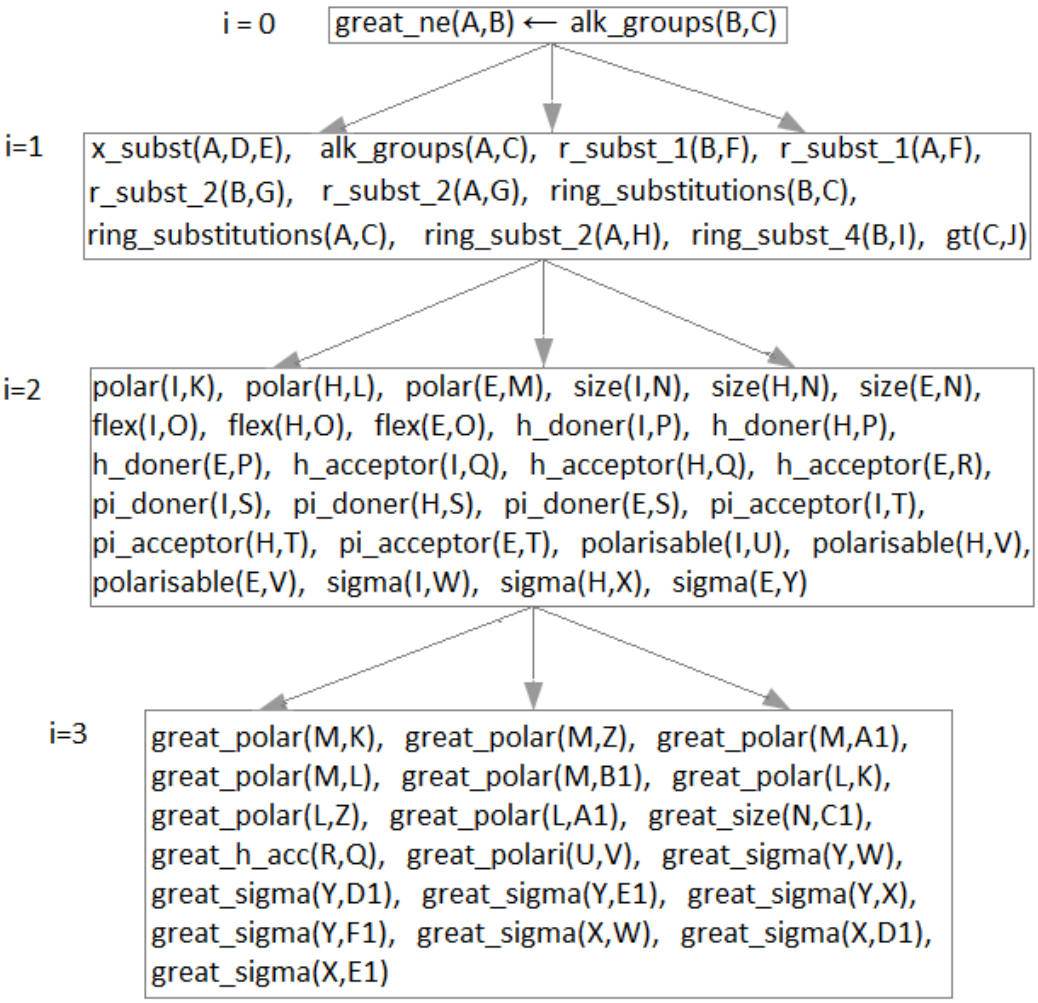

Figura 5.1: Bottom Clause gerada no FORTE_MBC

Na Figura 5.1, cada variável nova que aparece em uma camada possui profundidade de variável correspondente à altura da árvore. Na camada 0, todas as variáveis da clásula inicial são consideradas como variáveis de entrada para gerar os literais da camada 1, pois estamos considerando todas as variáveis de saída do corpo e as de entrada na cabeça. Na camada 1, as novas variáveis geradas (E,F,G,H,I,J) também

passarão a ser consideradas como variáveis de entrada para o próxima camada, e assim por diante, até atingir o limite máximo de profundidade de variável.

Como estamos seguindo o algoritmo FORTE_BETH. a Bottom Clause inicial, na verdade, contém os literais da Cláusula inicial $\overrightarrow{C}$*, e os literais gerados a partir dos termos usados para instanciar* $\overrightarrow{C}$*, termos estes provenientes da prova do exemplo* $great\_ne(kk1, bb1)$ *por esta cláusula.*

Seja $\overrightarrow{C'} = great\_ne(kk1, bb1) \leftarrow alk\_groups(bb1, 1)$ *a instanciação de* $\overrightarrow{C}$ *pelo exemplo* $great\_ne(kk1, bb1)$ *escolhido. Aplicando-se apenas a primeira iteração do algoritmo de geração de Bottom Clause do FORTE_MBC (Algoritmo A.1), considerando-se os modos definidos, e considerando-se os termos* $kk1, bb1$ *e* 1 *de* $\overrightarrow{C'}$ *como termos de entrada, os possíveis literais gerados são:*

$x\_subst(kk1, 6, f), alk\_groups(kk1, 1), r\_subst\_1(kk1, single\_alk(1)),$
$r\_subst\_1(bb1, single\_alk(1)), r\_subst\_2(kk1, aro(1)), r\_subst\_2(bb1, aro(1)),$
$ring\_substitutions(kk1, 1), ring\_substitutions(bb1, 1), ring\_subst\_2(kk1, cf3),$
$ring\_subst\_4(bb1, ch3), gt(1, 0).$

Usando a função hash, e considerando que os termos $kk1, bb1$ *e* 1 *já estão associados às variáveis* A, B *e* C*, respectivamente, os literais retornados variabilizados são:*

x_subst(A,D,E), alk_groups(A,C), r_subst_1(A,F), r_subst_1(B,F),
r_subst_2(A,G), r_subst_2(B,G), ring_substitutions(A,C),
ring_substitutions(B,C), ring_subst_2(A,H), ring_subst_4(B,I), gt(C,J).

Estes literais correspondem exatamente aos literais da camada de profundidade $i = 1$ *na Figura 5.1. A Bottom Clause* $\overrightarrow{\perp}$ *inicial é, portanto, dada por:*

$$\begin{aligned}\overrightarrow{\perp} = great\_ne(A, B) \leftarrow\ & alk\_groups(B, C), x\_subst(A, D, E), alk\_groups(A, C),\\ & r\_subst\_1(A, F), r\_subst\_1(B, F), r\_subst\_2(A, G), r\_subst\_2(B, G),\\ & ring\_substitutions(A, C), ring\_substitutions(B, C), ring\_subst\_2(A, H),\\ & ring\_subst\_4(B, I), gt(C, J).\end{aligned}$$

Continuando, tais literais são avaliados para se escolher o melhor a ser adicionado à cláusula $\overrightarrow{C}$*. Ao contrário do BETH, o FORTE_BETH não considera um beam de cláusulas, mas apenas uma só. Digamos que o melhor literal encontrado seja* $r\_subst\_2(A, G)$ *(instanciado por* $r\_subst\_2(kk1, aro(1))$*). A nova cláusula* $\overrightarrow{C}$ *é dada por* $\overrightarrow{C} = great\_ne(A, B) \leftarrow alk\_groups(B, C), r\_subst\_2(A, G)$*.*

O próximo passo é aplicar o algoritmo de geração da próxima camada da Bottom Clause, considerando-se o literal adicionado e a $\overrightarrow{\perp}$ *atual. Seguindo os passos do algoritmo 5.5, temos:*

- $UsedTerms = \{kk1, bb1, 1\}$
- *Profundidade de variável máxima* $i = 3$
- $InTerms = \{aro(1)\}$, *contendo apenas os termos ainda não usados como entrada e cuja profundidade de variável associada é menor que 3.*

Para o termo $aro(1)$*, não há nenhum átomo básico no BK que tenha este termo como entrada, de forma que o algoritmo 5.5 não incluirá nenhum novo literal em* $\overrightarrow{\perp}$*. Como se pode ver pela árvore completa de* $\overrightarrow{\perp}$*, nenhum literal da camada* $i = 2$ *foi gerado a partir da variável* G *associada ao termo* $aro(1)$*.*

Na próxima escolha de literal a ser adicionado, portanto, os literais considerados para avaliação são os mesmos gerados na camada 1 (desconsiderando o $r\_subst\_2(A, G)$ *que já está na cláusula). Digamos que o melhor literal encontrado seja* $ring\_subst\_2(A, H)$ *(instanciado por* $ring\_subst\_2(kk1, cf3)$*). A nova cláusula* $\overrightarrow{C}$ *é dada por* $\overrightarrow{C} = great\_ne(A, B) \leftarrow alk\_groups(B, C), r\_subst\_2(A, G), ring\_subst\_2(A, H)$*.*

O literal $ring\_subst\_2(A, H)$ *introduziu a variável* H*, instanciada pelo termo* $cf3$*. Executando o algoritmo 5.5, este termo está no* $InTerms$ *e é usado como entrada para a geração de novos literais. Os novos literais introduzidos em* $\overrightarrow{\perp}$ *são:*

$$polar(H, L), size(H, N), flex(H, O), h\_doner(H, P), h\_acceptor(H, Q),$$
$$pi\_doner(H, S), pi\_acceptor(H, T), polarisable(H, V), sigma(H, X)$$

Visto que H *tem profundidade de variável* 1*, as novas variáveis introduzidas (*K, L, M, N, O, P, Q, R *e* S*), serão associadas à profundidade de variável* 2*. Observe que na Figura 5.1, estes literais criados estão exatamente na camada 2. Observe ainda que nem todos os literais da camada 2 foram gerados, pois nem todos eram relevantes levando-se em conta a cláusula* $\overrightarrow{C}$ *atual. A Bottom Clause atualizada é dada por:*

$$\begin{aligned}\overrightarrow{\perp} = {} & great\_ne(A, B) \leftarrow alk\_groups(B, C), x\_subst(A, D, E), alk\_groups(A, C), \\ & r\_subst\_1(A, F), r\_subst\_1(B, F), r\_subst\_2(A, G), r\_subst\_2(B, G), \\ & ring\_substitutions(A, C), ring\_substitutions(B, C), ring\_subst\_2(A, H), \\ & ring\_subst\_4(B, I), gt(C, J), polar(H, L), size(H, N), flex(H, O), \\ & h\_doner(H, P), h\_acceptor(H, Q), pi\_doner(H, S), pi\_acceptor(H, T), \\ & polarisable(H, V), sigma(H, X).\end{aligned}$$

Continuando, suponha agora que o literal escolhido para ser adicionado a $\overrightarrow{C}$ *seja* $ring\_substitutions(A, C)$ *(instanciado por* $ring\_substitutions(kk1, 1)$*). A nova cláusula* $\overrightarrow{C}$ *é dada por* $\overrightarrow{C} = great\_ne(A, B) \leftarrow alk\_groups(B, C), r\_subst\_2(A, G), ring\_subst\_2(A, H), ring\_substitutions(A, C)$*.*

O literal adicionado não possui nenhum termo novo ainda não usado, visto que o termo 1 *já foi usado logo na primeira iteração do algoritmo, para gerar a Bottom Clause inicial. Portanto, nenhum novo literal será adicionado a* $\overrightarrow{\perp}$.

Suponha que neste momento o algoritmo FORTE_BETH para, pois não consegue mais especializar a cláusula, ou seja, nenhum outro literal de $\overrightarrow{\perp}$, *quando adicionado a* $\overrightarrow{C}$, *melhora a pontuação desta. Ao final do algoritmo, a Bottom Clause gerada contém apenas 22 literais, que é menos da metade da Bottom Clause gerada a priori no FORTE_MBC, que tem 59 literais. Portanto, no FORTE_MBC, 37 literais teriam sido gerados desnecessariamente, pois não foram relevantes na busca efetuada. A Figura 5.2 mostra a comparação entre a Bottom Clause gerada pelo FORTE_BETH e a gerada pelo FORTE_MBC. Os literais sublinhados são os literais da Bottom Clause gerada no FORTE_BETH.*

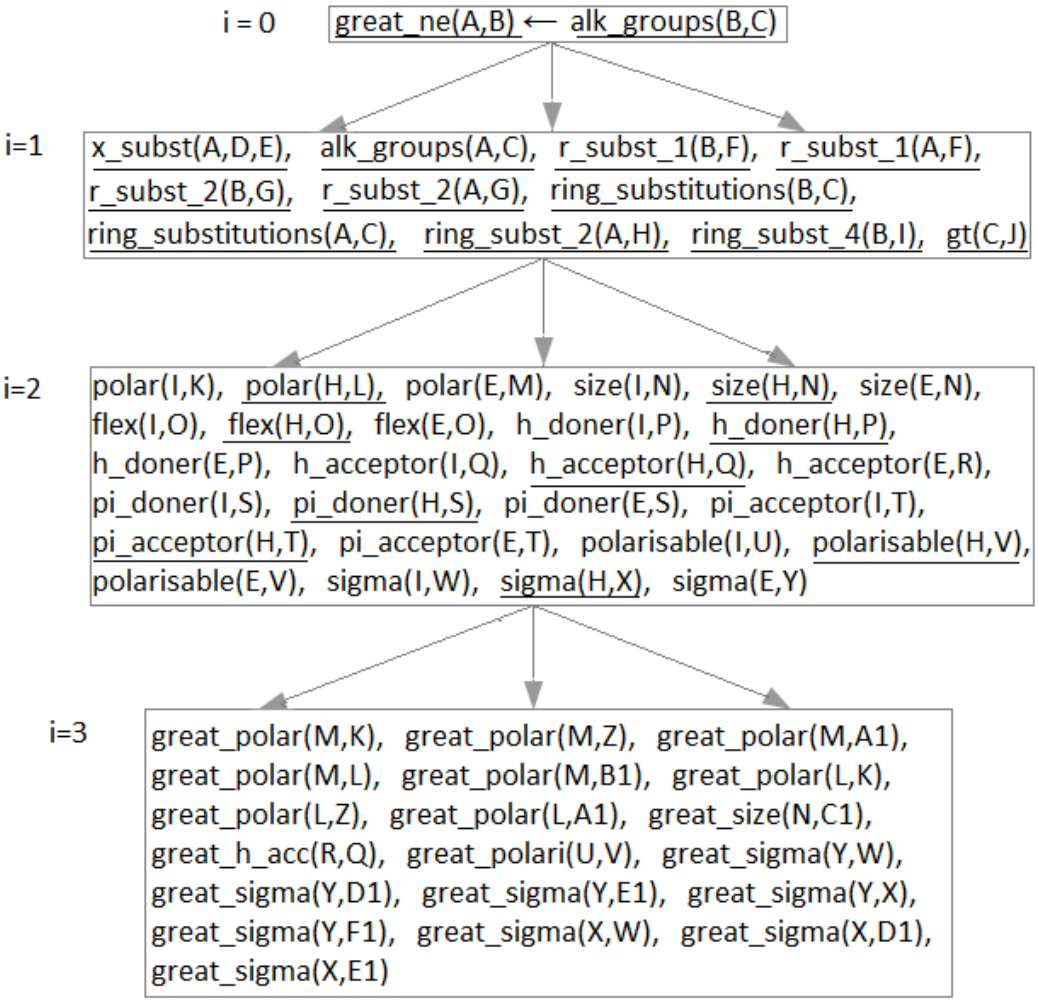

Figura 5.2: Bottom Clause gerada no FORTE_BETH comparada à do FORTE_-MBC

5.5 Resultados Experimentais

Nesta seção mostramos os experimentos realizados considerando 2 conjuntos de dados, com o objetivo de mostrar os resultados obtidos quando consideramos o algoritmo do FORTE_BETH para gerar *Bottom Clauses* durante a aplicação do operador de Adição de Antecedentes, de forma que apenas os literais relevantes para a revisão de uma cláusula são considerados.

Os domínios considerados são:

- UW-CSE [39]: consiste de informações sobre o Departamento de Ciências da Computação e Engenharia da Universidade de Washington. A base contém 113 exemplos positivos e 2061 exemplos negativos.

- Link_Discovery [44]: base gerada após os eventos de 11/09/2001, como parte de um projeto que visava detectar e prevenir o terrorismo. Ela foi usada no problema de identificar assassinos contratados no domínio do Crime Organizado Russo, e contém representações de pessoas, organizações, objetos, ações e muitos tipos de relações entre estes. A base contém 133 exemplos positivos e 499 exemplos negativos.

Nosso principal objetivo é investigar se o uso do algoritmo FORTE_BETH no operador de Adição de Antecedentes reduz não só o tempo de geração e tamanho das *Bottom Clauses* (como já provado pelo sistema BETH [44]), mas também o tempo de execução do processo de revisão como um todo.

Durante o treinamento, utilizamos validação cruzada k-fold para dividir o conjunto de entrada em conjuntos distintos de treinamento e teste [15]. Ambas as bases citadas já tem conjuntos (folds) pré-definidos, sendo k=5 para o UW-CSE, e k=6 para o Link_Discovery. Nós medimos, tanto no FORTE_MBC quanto no FORTE_BETH, o tempo médio de geração de *Bottom Clauses* por conjunto de treinamento (*fold*), o tempo médio de geração de uma única *Bottom Clause*, o tamanho médio de uma *Bottom Clause*, o tempo médio de treinamento por *fold*, a acurácia de teste (no caso do Link_Discovery), e a *f-measure* no conjunto de teste (no caso do UW-CSE, pois a base é desbalanceada). Todos os tempos estão expressos em segundos nos resultados. As diferenças significativas foram testadas utilizando *corrected two-tailed paired t-test* com $p < 0{,}05$[30].

O Aleph foi usado para gerar as teorias iniciais fornecidas para os algoritmos de revisão. Uma teoria diferente foi gerada para cada *fold*, e cada uma destas teorias foi revisada considerando seus respectivos *folds*, ou seja, os mesmos *folds* foram usados para gerar e revisar teorias.

Para execução do Aleph, FORTE_MBC e FORTE_BETH, utilizamos computadores Dells Optiplex core i7 - 4 cores com 8GB de memória RAM. A implementação

foi feita utilizando YAP Prolog, versão 5.1.3 [7].

Os parâmetros utilizados para geração das teorias no Aleph e consequente revisão das mesmas no FORTE_MBC e FORTE_BETH, para cada uma das bases de dados, são dados a seguir. Tais parâmetros foram setados de forma a gerar as melhores teorias no Aleph.

Parâmetros do Aleph para o UW-CSE:

- Tamanho máximo de cláusula (*clause length*) = 10
- Número mínimo de exemplos positivos cobertos por uma cláusula (minpos) = 10
- Limite superior no número de exemplos negativos cobertos por uma cláusula (noise) = 1000
- Função de avaliação: *m-estimate*, com *m default*
- limite no número de nós buscados no Aleph = 1000
- Acurácia mínima: 0,1

Parâmetros do Aleph para o Link_Discovery:

- Tamanho máximo de cláusula (*clause length*) = 15
- Função de Avaliação: m-estimate, com $m = 2$, conforme usado por [44].
- Profundidade de Variável $i = 5$

No UW-CSE foi usada a função de avaliação *coverage* no FORTE_MBC e FORTE_BETH, pois a função *m-estimate* não estava convergindo, e estava gerando uma grande variância entre os *folds*. Porém, ambas as funções de avaliação obtiveram resultados iguais quando calculando o *f-measure*. No Link_Discovery também foi usada a função de avaliação *coverage* para avaliar as revisões propostas, já que a função *m-estimate* também não estava convergindo.

Os experimentos conduzidos verificam a influência de três fatores no tempo de geração e tamanho da *Bottom Clause*, e o quanto a redução obtida no tempo de geração de *Bottom Clauses* afeta a redução no tempo de treinamento do processo de revisão. Os fatores são: 1) limite de profundidade de variável i, 2) conjunto de modos definidos e o número de ocorrências de $+Type$ em todos os *modeb*, 3) predicados *built-in* tais como os literais de igualdade introduzidos pelo *variable splitting*. Os dois primeiros fatores correspondem aos fatores de exponencialidade do tamanho da *Bottom Clause*, e a influência destes fatores é analisada na subseção 5.5.1. A influência do terceiro fator é analisada na subseção 5.5.2.

Os experimentos realizados focam na comparação entre o FORTE_MBC e o FORTE_BETH. Valores de tempo de treinamento e acurácia (ou *f-measure*) no sistema Aleph também são mostrados, visto que o Aleph gerou as teorias iniciais para os sistemas de revisão.

5.5.1 Influência dos fatores de exponencialidade

De acordo com o teorema 15, a cardinalidade da *Bottom Clause* gerada pelo método de saturação do Aleph, o qual é usado no FORTE_MBC, é limitada por $(r|M|j{+}j{-})^{ij^+}$, tal que a complexidade é exponencial com respeito a dois fatores: 1) limite de profundidade de variável i e 2) parâmetro j_+, que representa o número de ocorrências de $-type$ em um modo do tipo *modeh*, que define o modo do predicado da cabeça, e o número de ocorrências de $+type$ em um modo do tipo *modeb*, referente aos literais que aparecem no corpo de uma cláusula.

Para avaliar a influência destes fatores no tamanho da *Bottom Clause* e, consequentemente, no tempo de geração da mesma e no tempo total de treinamento do processo de revisão, fizemos os seguintes experimentos:

- Base UW-CSE com os valores $i = 2$ e $i = 3$, sendo que, nestes experimentos não estamos considerando o uso dos modos para o predicado *diff*, incluído no conjunto de modos da base UW-CSE mostrado no **Apêndice B**. O predicado *diff* é um predicado *built-in* incluído nos modos. Nestes experimentos estamos interessados apenas na influência dos fatores de exponencialidade, independente de predicados *built-in*. Os resultados são mostrados nas Tabelas 5.1 e 5.2.

- Base Link_Discovery com os valores $i = 2$, $i = 4$ e $i = 5$. Os resultados são mostrados nas Tabelas 5.3 e 5.4.

Quanto ao parâmetro j^+, ambas as bases UW-CSE e Link_Discovery possuem, para cada declaração de modo, $1 \leq j^+ \leq 2$, tal que os modos contém pelo menos, um argumento $+Type$, conforme podemos observar no **Apêndice B**.

Tabela 5.1: UW-CSE: Tempo e tamanho médio da *Bottom Clause*: FORTE_MBC X FORTE_BETH

UW-CSE	FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3
Tempo médio de geração das $\overrightarrow{\perp}$	1,95	38,91	0,38	0,44
Tempo médio de geração de uma única $\overrightarrow{\perp}$	0,06	1,16	0,01	0,01
Tamanho médio da $\overrightarrow{\perp}$	446,34	1954,47	70,79 •	70,95 •

Tabela 5.2: UW-CSE: Tempo Médio de Treinamento e *f-measure* de teste: Aleph X FORTE_MBC X FORTE_BETH

UW-CSE	Aleph		FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3	i=2	i=3
Tempo	14,85	34,52	103,0	166,88	93,66	98,68
f-measure	0,24	0,24	0,23	0,23	0,23	0,23

Para a base UW-CSE, foram criadas 164 *Bottom Clauses* durante o processo de revisão, considerando os 5 *folds* de treinamento. Pela Tabela 5.1 podemos observar que, quando comparando o FORTE_BETH ao FORTE_MBC, houve um redução considerável no tempo de geração das *Bottom Clauses*, e uma redução significativa (indicada pelo símbolo •), de acordo com o t-test, no tamanho médio de uma *Bottom Clause*, como já era esperado. Em particular, temos:

- Redução no tempo de criação de uma única *Bottom Clause*: para $i = 2$ foi de $6\times$. Para $i = 3$ foi de $116\times$.
- Redução no tempo de criação de *Bottom Clauses* por fold: para $i = 2$ foi de $5,13\times$. Para $i = 3$ foi de 88,43×.
- Redução no tamanho da *Bottom Clause*: para $i = 2$ foi de 6,3× menor. Para $i = 3$ foi de 27,55×.

Ao compararmos as reduções de tempo e redução de tamanho da *Bottom Clause* para os dois valores do limite de profundidade de variável i, temos que quando o i aumenta, as diferenças de tempo e tamanho aumentam consideravelmente, de forma exponencial, como era esperado.

Já a Tabela 5.2 mostra o tempo médio de treinamento do processo de revisão e a *f-measure* de teste obtida na base UW-CSE para os três sistemas: Aleph, FORTE_-MBC e FORTE_BETH. Apesar da grande redução no tamanho e no tempo de criação da *Bottom Clause*, principalmente para $i = 3$, mostrados na Tabela 5.1, esta diferença não está refletindo consideravelmente no tempo de treinamento do sistema. Observe na Tabela 5.2 que, mesmo para $i = 3$, a diferença de tempo de treinamento foi apenas 1,69× menor.

Quando comparando os resultados de tempo de treinamento e *f-measure* do FORTE_MBC e FORTE_BETH com o Aleph, percebemos, primeiramente, que o tempo no Aleph é bem menor que nos sistemas de revisão. Isso é esperado, pois estamos lidando com revisão de teorias inteiras como um todo, e não com aprendizado de cláusulas uma de cada vez. A revisão, normalmente, é um processo mais custoso que o aprendizado. Porém, de acordo com a Tabela 5.2, a *f-measure* de teste não foi significativa no FORTE_MBC e no FORTE_BETH quando comparadas

ao Aleph, o que mostra que, para a base UW-CSE em questão, a revisão não foi relevante.

Observe que a *f-measure* obtida para o FORTE_MBC e para o FORTE_BETH é a mesma, pois em ambos os sistemas os literais provenientes da *Bottom Clause* que são avaliados na Adição de Antecedentes, são os mesmos. Ambos os sistemas avaliam os literais que são relevantes para a revisão da cláusula, ou seja, aqueles que podem ser adicionados à cláusula sem que os modos definidos sejam desrespeitados. A diferença é que, no FORTE_BETH, apenas estes literais são gerados na *Bottom Clause*, enquanto que no FORTE_MBC, todos os literais relevantes para o exemplo usado na geração da *Bottom Clause* são incluídos nesta, mesmo que muitos deles nunca sejam selecionados como candidatos a serem adicionados na cláusula.

Nas Tabelas 5.3 e 5.4 mostramos os resultados obtidos na base do Link_Discovery, considerando $i = 2$, $i = 4$ e $i = 5$. Para esta base foram criadas, no total, 722 *Bottom Clauses* durante o processo de revisão.

Tabela 5.3: Link_Discovery: Tempo e tamanho médio da *Bottom Clause*: FORTE_MBC X FORTE_BETH

Link_Discovery	FORTE_MBC			FORTE_BETH		
	i=2	i=4	i=5	i=2	i=4	i=5
[1]	0,18	3,39	5,39	0,17	2,04	2,68
[2]	0,011	0,026	0,047	0,010	0,016	0,023
[3]	25,44	117,15	168,36	12,75 •	23,17 •	19,56 •

[1]Tempo médio de geração das $\overrightarrow{\perp}$
[2]Tempo médio de geração de uma única $\overrightarrow{\perp}$
[3]Tamanho médio da $\overrightarrow{\perp}$

Na Tabela 5.3 podemos observar que o tamanho da *Bottom Clause* no FORTE_BETH reduziu significativamente em 1,99× para $i = 2$, 5,05× pra $i = 4$ e 8,6× para $i = 5$, mostrando a influência do i no tamanho da *Bottom Clause*. No caso do FORTE_MBC e FORTE_BETH, temos que para $i = 4$ o tamanho, em média, da *Bottom Clause* está um pouco maior que para $i = 5$. Isso pode ser justificado pelo fato de que as teorias iniciais são diferentes para os dois valores de i.

Quanto ao tempo de criação das *Bottom Clauses*, comparando-se o FORTE_MBC com o FORTE_BETH, este só começa a mostrar uma redução maior a partir do $i = 5$, porém numa proporção menor que a obtida na base UW-CSE. No caso do Link_Discovery, a redução no tempo de geração da *Bottom Clause* foi de apenas 2,01× para $i = 5$, aproximadamente, comparado a uma redução de 6× obtido no UW-CSE, para $i = 2$, conforme mostrado na Tabela 5.1. Observe que as *Bottom Clauses* geradas no Link_Discovery são menores do que as geradas no UW-CSE. No caso do Link_Discovery, o limite de profundidade de variável i influencia no

tamanho da *Bottom Clause* mas não de forma significante, a ponto de fazer uma diferença muito grande no tempo de geração da *Bottom Clause*.

Observe que nem o limite de profundidade de variável i e nem o parâmetro j^+ influenciam no tamanho da *Bottom Clause* do FORTE_BETH, pois a complexidade de tamanho da *Bottom Clause* neste caso é linear, tal que no pior caso é $O(bn|P|r)$, conforme Teorema 16, ou seja, são relevantes apenas o tamanho máximo de cláusula n, o número de definições de predicado $|P|$, e o recall r. O tamanho do *beam* b, neste caso, é irrelevante, pois estamos considerando $b = 1$.

A Tabela 5.4 a seguir mostra o tempo médio de treinamento do processo de revisão e a acurácia de teste obtida, no Link_Discovery, para os três sistemas: Aleph, FORTE_MBC e FORTE_BETH.

Tabela 5.4: Link_Discovery: Tempo Médio de Treinamento e Acurácia de Teste: ALEPH X FORTE_MBC X FORTE_BETH

	Aleph			FORTE_MBC			FORTE_BETH		
	i=2	i=4	i=5	i=2	i=4	i=5	i=2	i=4	i=5
[1]	20,30	150,08	212,41	5,70	65,32	61,02	5,60	60,67	54,30
[2]	79,13	80,93	80,71	84,68 •	84,17 •	83,57	84,68 •	84,17 •	83,57

[1]Tempo de treinamento
[2]Acurácia de teste

Na Tabela 5.4 podemos observar que houve um aumento de acurácia do FORTE_MBC e do FORTE_BETH em relação ao Aleph, para todos os valores de i, sendo que, de acordo com o *ttest* aplicado, esta diferença é significativa nos casos de $i = 2$ e $i = 4$. No caso do Link_Discovery, as teorias geradas pelo Aleph para cada valor de i são diferentes e, portanto, a busca por teorias revisadas no FORTE_MBC e FORTE_BETH começam de pontos diferentes do espaço de busca. A teoria gerada com $i = 4$, por exemplo, foi um ponto de partida melhor para a revisão, pois gerou uma teoria mais acurada (significativamente) quando comparada à teoria inicial do Aleph, o que não acontece quando o $i = 5$. No Aleph, as acurácias obtidas para os diferentes valores de i não são significativamente diferentes.

Quanto ao tempo de treinamento, o tempo obtido para $i = 4$ e $i = 5$ são consideravelmente maiores do que para $i = 2$, apesar de que comparando o $i = 4$ com $i = 5$, não houve um aumento significativo. Porém, ao compararmos o FORTE_MBC com o FORTE_BETH, a diferença de tempo entre eles não é substancial, ou seja, no caso do Link_Discovery, não houve um ganho suficiente no tempo de geração das *Bottom Clauses* para fazer diferença no tempo total de treinamento do processo de revisão.

5.5.2 Influência do uso de predicados *built-in*

Além dos fatores mostrados na subseção anterior, um outro fator que também pode influenciar no tempo de geração e tamanho da *Bottom Clause* é o uso de predicados *built-in*. Predicado *built-in* são usados, por exemplo, para expressar igualdade entre variáveis quando o *variable splitting* é considerado. Tais literais de igualdade são inseridos de forma a manter a equivalência, conforme visto no Capítulo 4.

Para as bases sendo consideradas nestes experimentos, o uso do *variable splitting* não se faz necessário. Porém, para mostrar a influência que um *variable splitting* poderia ter no tamanho da *Bottom Clause*, iremos considerar, na base UW-CSE, o uso do predicado *diff* (com modos do tipo $(+,+)$), que também é um predicado *built-in*. Tal predicado, ao contrário dos predicados de igualdade, serve para diferenciar dois termos, conforme mostrado no **Apêndice B**. Observe que no modos definidos temos cinco definições de modos para o *diff*, todas elas com argumentos +type, o que também influencia.

As Tabelas 5.5 e 5.6 mostram os resultados obtidos para a base UW-CSE, considerando o predicado *diff* na criação da *Bottom Clause*, tanto para $i = 2$ quanto para $i = 3$. Repetimos os resultados obtidos no UW-CSE sem *diff* para fins de comparação.

Tabela 5.5: UW-CSE: Tempo e tamanho médio da *Bottom Clause*: FORTE_MBC X FORTE_BETH

UW-CSE com *diff*	FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3
Tempo médio de geração das $\overrightarrow{\perp}$	20,27	7379,62	0,39	0,5
Tempo médio de geração de uma única $\overrightarrow{\perp}$	0,59	216,24	0,01	0,01
Tamanho médio da $\overrightarrow{\perp}$	1726,17	42876,03	71,36 •	72,63 •
UW-CSE sem *diff*	FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3
Tempo médio de geração das $\overrightarrow{\perp}$	1,95	38,91	0,38	0,44
Tempo médio de geração de uma única $\overrightarrow{\perp}$	0,06	1,16	0,01	0,01
Tamanho médio da $\overrightarrow{\perp}$	446,340	1954,470	70,79 •	70,95 •

Ao considerar o predicado *diff*, o tempo médio de geração das *Bottom Clauses* no FORTE_MBC aumentou consideravelmente. Se compararmos com os resultados obtidos quando o *diff* não tinha sido considerado, o tempo médio de geração das *Bottom Clauses* no FORTE_MBC para $i = 3$ aumentou de 38,91 para 7379,62, o que corresponde a um aumento de 189,65 vezes. No entanto, observe que, no FORTE_-BETH, o uso do *diff* não influenciou tanto no tempo de geração e tamanho da *Bottom Clause*, pois literais para o predicado *diff* só são inseridos na *Bottom Clause*

Tabela 5.6: UW-CSE: Tempo Médio de Treinamento e *f-measure* de teste: Aleph X FORTE_MBC X FORTE_BETH

UW-CSE com *diff*	Aleph		FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3	i=2	i=3
Tempo	21,66	419,60	142,55	18537,27	93,87	114,00
f-measure	0,24	0,24	0,23	0,23	0,23	0,23
UW-CSE sem *diff*	Aleph		FORTE_MBC		FORTE_BETH	
	i=2	i=3	i=2	i=3	i=2	i=3
Tempo	14,85	34,520	103,00	166,8800	93,66	98,680
f-measure	0,24	0,24	0,23	0,23	0,23	0,23

se eles forem relevantes para a cláusula sendo revisada. O número de literais com o predicado *diff* inseridos na *Bottom Clause* está diretamente relacionado à quantidade de diferentes termos básicos que instanciam a mesma. Como no FORTE_BETH a *Bottom Clause* é criada dinâmicamente, esta terá menos termos do que se tivesse sido criada a priori, antes da revisão, como no caso do FORTE_MBC. Isto justifica a diferença no tamanho da *Bottom Clause* criada no FORTE_MBC, usando e não usando o *diff*. Observe que, para $i = 2$, o tamanho médio aumentou de 446,34 para 1726,17, enquanto que, para $i = 3$, aumentou de 1954,47 para 42876,03.

Os resultados da Tabela 5.5 para o UW-CSE com *diff* mostram uma redução de 59× no tempo de criação de uma única *Bottom Clause*, quando $i = 2$, na comparação entre o FORTE_BETH e o FORTE_MBC, e uma redução de 21624× quando $i = 3$. Já o tamanho da *Bottom Clause* gerada no FORTE_BETH reduziu em 24,2× para $i = 2$, e 590,33× para $i = 3$, quando comparado ao FORTE_BETH.

Apesar da grande redução no tamanho e no tempo de criação da *Bottom Clause*, o tempo gasto para gerar *Bottom Clauses* no FORTE_MBC e no FORTE_BETH podem ser muito pequenos quando comparados ao tempo de treinamento durante o processo de revisão de teorias, e portanto a diferença obtida na geração da *Bottom Clauses* pode não refletir tanto no tempo total de execução do sistema. Para $i = 2$, por exemplo, no FORTE_BETH, o tempo médio de geração de *Bottom Clauses* em um *fold* corresponde a 0,39s, enquanto que o tempo médio de treinamento para um *fold* é de 93,87s. Na tabela 5.6 podemos observar os tempos de treinamentos obtidos na base UW-CSE, tanto com o *diff* como sem, considerando $i = 2$ e $i = 3$. Temos que, para a grande maioria dos resultados obtidos, a diferença de tempo de treinamento entre o FORTE_MBC e o FORTE_BETH não é grande, não chegando a ser nem 2× menor. Porém há um caso em que a diferença nos tempos é razoavelmente grande e mais aparente, que é quando consideramos $i = 3$ e o uso do *diff*. Neste caso temos a relação de 18537,27 X 114,00 entre o FORTE_MBC e o FORTE_BETH, o que corresponde a uma redução de 162,6× no FORTE_BETH. Este resultado é um

indicativo de que, no FORTE_BETH, a redução no tempo de geração da *Bottom Clause* só vai efetivamente começar a fazer diferença no tempo total de treinamento da revisão quando a combinação das seguintes situações acontecer: uso de limites de profundidade de variável i maiores, uso de modos com vários argumentos do tipo $+type$, e uso de predicados *built-in*, como no *variable splitting* (o que também aumenta o número de argumentos $+Type$).

5.5.3 Conclusão

Como conclusão dos experimentos feitos temos que:

- O tempo gasto para gerar *Bottom Clauses* durante o processo de revisão de um teoria pode ser muito pequeno quando comparado ao tempo total de treinamento. Portanto, para que esse tempo faça alguma diferença (significativa) quando comparando o FORTE_MBC e o FORTE_BETH, é preciso que a diferença no tamanho da *Bottom Clause* seja realmente grande.

- Sabemos, pela análise de complexidade do tamanho da *Bottom Clause* feita pelo sistema BETH, que o tamanho é exponencial no limite de profundidade de variável i e no parâmetro j^+ (número de ocorrências de $-Type$ na cabeça e $+Type$ no corpo). Portanto, quanto maior for o i e quanto mais argumentos $+Type$ em um modo do corpo ou $-Type$ em um modo da cabeça forem definidos, maior será a *Bottom Clause*.

- Por fim, também podemos atribuir o tamanho exponencial de uma *Bottom Clause* ao uso de predicados *built-in*, como o *diff* no UW-CSE, e como os literais de igualdade que são introduzidos quando considerando o *variable splitting*. Estes predicados aumentam consideravelmente o tamanho da *Bottom Clause*, e quando combinados com um limite de profundidade i maior, aumentam ainda mais o tamanho, começando a ter uma influência maior no tempo de treinamento (18537,27 X 114 para $i = 3$ na Tabela 5.6).

Capítulo 6

Conclusão

O FORTE_MBC [10] surgiu a partir do FORTE pela modificação do operador de refinamento de Adição de Antecedentes. O FORTE_MBC passou a utilizar a cláusula mais específica, conhecida como *Bottom Clause*, para limitar o espaço de busca de antecedentes a serem adicionados a uma cláusula, e também introduziu a declaração de modos para validar as cláusulas geradas na revisão. De acordo com [10], o FORTE_MBC reduziu consideravelmente o espaço de busca de cláusulas, que por sua vez reduziu o espaço de busca de teorias, o que se mostrou essencial para reduzir o tempo de execução do processo de revisão em uma média de 55 vezes. Além disso, ele gera teorias mais compreensíveis e não diminui significativamente a acurácia obtida pelo processo de revisão original, mantendo as acurácias mais elevadas quando comparado ao aprendizado do zero, assim como o sistema FORTE original. No entanto, a eficácia e a eficiência do aprendizado como um processo de refinamento dependem fortemente de outros fatores, tais como as propriedades do espaço de busca e, como consequência, dos operadores de refinamento [13]. Tradicionalmente, em ILP, os sistemas são definidos através de uma caracterização formal desses fatores, assim algoritmos e operadores mais eficientes podem ser projetados, porém tais fatores não tinham sido devidamente caracterizados no FORTE_MBC, até a presente Tese.

Como contribuições desta análise dos operadores de refinamento e espaço de busca do FORTE_MBC nós:

- modificamos o modelo teórico apresentado em [42], introduzindo a *subsumption* proposicional relativa a uma *Bottom Clause*, de forma a caracterizar o espaço de busca de cláusulas do FORTE_MBC.

- estendemos os resultados teóricos apresentados de forma a caracterizar o espaço de busca de teorias do FORTE_MBC, onde as teorias são conjunto de cláusulas restritas por *Bottom Clauses* e declaração de modos. Nós definimos

um reticulado de teorias também ordenado pela *subsumption* proposicional relativa a uma *Bottom Clause* e provamos que os operadores de refinamento de teorias do FORTE_MBC não são ideais para este reticulado.

- Modificamos o operador de generalização de cláusulas de Exclusão de Antecedentes, tornando-o completo para o espaço de busca de cláusulas do FORTE_MBC, o que também tornou completo o operador de generalização de teorias do FORTE_MBC.

- Tornamos os operadores de teorias do FORTE_MBC próprios, ao considerar teorias não redundantes a cada passo de refinamento. Com isso, os operadores passaram a ser ideais para o reticulado definido, o que é desejado quando a busca começa de qualquer ponto do espaço de busca, como acontece no FORTE_MBC.

Os resultados obtidos são importantes para melhor entender o espaço restrito de refinamento de teorias de sistemas de revisão de teorias de primeira-ordem, que usam uma teoria inicial, um conjunto de *Bottom Clauses* e declaração de modos para restringir o espaço de busca, como é o caso do FORTE_MBC. Tais resultados podem ser aplicados a qualquer sistema de revisão de teorias de primeira-ordem tendo estas características.

Ao analisar a *Bottom Clause* como espaço de busca de literais e limitante do reticulado de cláusulas no FORTE_MBC, temos que, apesar da *Bottom Clause* reduzir o espaço de busca, quando comparado a um espaço ordenado por θ-*subsumption*, por exemplo, ela também pode resultar num espaço de busca exponencial, dependendo dos parâmetros considerados na sua construção. Visando diminuir este espaço de busca e possivelmente aumentar a eficiência do sistema, nesta Tese implementamos o sistema FORTE_BETH, modificando o algoritmo de Adição de Antecedentes do FORTE_MBC para utilizar o algoritmo proposto em [44], de forma que a *bottom clause* não mais é construída a priori, mas descoberta durante o processo de especialização de uma cláusula, fazendo com que o espaço de busca deixe de ser exponencial e passe a ser linear.

Concluímos a partir dos experimentos realizado no sistema FORTE_BETH que, apesar deste resultar numa redução significativa do tamanho da *Bottom Clause*, o tempo gasto para gerar *Bottom Clauses* durante o processo de revisão de um teoria pode ser muito pequeno quando comparado ao tempo total de treinamento. Para que esse tempo faça alguma diferença (significativa) no tempo total de treinamento do processo de revisão, é preciso que a diferença no tamanho da *Bottom Clause* seja realmente grande, e isso só é conseguido se combinarmos os seguintes parâmetros, que tornam a *Bottom Clause* exponencial: profundidade de variável i grandes, muitos argumentos $+Type$ nas declarações de modos do corpo e $-Type$ nas declarações de

modos da cabeça, e o uso de predicados *built-in*, como os usados quando *variable splitting* é considerado.

É importante ressaltar que a redução de tamanho da *Bottom Clause* obtida, assim como o ganho em eficiência no processo de construção da mesma, não afetam a capacidade de generalização do sistema. No caso, a caracterização do espaço de busca e operadores de refinamento feita no Capítulo 4 para o FORTE_MBC também é válida para o sistema FORTE_BETH. Houve uma redução no tamanho do espaço de busca de cláusulas (devido à redução no tamanho da *Bottom Clause*), porém as características deste espaço continuam as mesmas (mesma linguagem e ordem de generalidade).

6.1 Trabalhos Futuros

Como trabalho futuro, nós pretendemos implementar no FORTE_MBC os novos operadores de refinamento e as soluções propostas na Seção 4.4, de forma a termos operadores ideais e menos restritos. Também pretendemos analisar o sistema YAV_FORTE [32]. YAV_FORTE foi construído a partir do FORTE e do FORTE_MBC com algumas modificações: ele inclui, entre outras coisas, a busca local estocástica para buscar por pontos de revisão, por literais a serem adicionados a uma cláusula e pela revisão a ser implementada. Ele também contribui com: 1) suporte para negação por falha, para incluir literais negados no corpo de cláusulas e para revisar pontos na teoria que falharam por conta de literais negados; 2) suporte para abdução em três momentos distintos do processo de revisão [28]. Para analisar a busca local estocástica introduzida no YavFORTE seguiremos o estudo feito em [43]. Este adaptou a ordem de generalidade em cláusulas e os conceitos de operadores de refinamentos para lidar com busca estocástica, introduzindo o conceito de operadores de refinamento estocásticos.

Finalmente, pretendemos analisar o impacto de se usar no FORTE_MBC a abordagem ARMGs (*Asymmetric Relative Minimal Generalisation*) definida em [29]. Esta abordagem explora uma variante assimétrica do RLGG de Plotkin, e é baseada na *subsumption order* relativa a uma *Bottom Clause* (como definida em [42]). Visamos com isso propor um novo operador de generalização de teorias que é capaz de gerar o RLGG de duas cláusulas, baseando-se na *subsumption* proposicional relativa a uma *Bottom Clause* definida neste Tese.

Apêndice A

Algoritmo A.1 Algoritmo para construção da *Bottom Clause* no FORTE_MBC (baseado no algoritmo de construção da *Bottom Clause* do Aleph\Progol [24])

Seja $\overrightarrow{C} = h \leftarrow l_1, ..., l_n$ a cláusula a ser revisada.
Seja e um exemplo positivo provado por $\overrightarrow{C}$.
$hash : Terms \rightarrow N$ é uma função que univocamente mapeia termos em variáveis distintas.

1: Inicialize a *Bottom Clause* $\overrightarrow{\perp}$ a partir de $\overrightarrow{C}$ e e.
2: $InTerms$ = todas os termos da $\overrightarrow{\perp}$ inicial que são do tipo $+Type$ na cabeça, ou $+Type$ ou $-Type$ no corpo.
3: **para cada** declaração *modeb* de corpo b **faça**
4: **para toda** substituição possível θ de argumentos correspondendo a tipo $+Type$ por termos no conjunto $InTerms$ **faça**
5: **repita**
6: **se** Prolog tem sucesso no objetivo b com substituição θ' **então**
7: **para cada** v/t em θ e θ' **faça**
8: **se** v corresponde a tipo $\sharp$ **então**
9: substitua v em b por t
10: **senão**
11: substitua v em b por v_k onde $k = hash(t)$
12: **se** v corresponde a tipo $-Type$ **então**
13: adicione t ao conjunto $InTerms$
14: Adicione b a $\overrightarrow{\perp}$
15: **até que** se tenha executado o loop *Recall* vezes
16: Incremente a profundidade de variável
17: Vá para o passo 3 se a profundidade máxima de variável não foi alcançada
18: Retorne $\overrightarrow{\perp}$ variabilizada usando-se a função $hash$.

Apêndice B

Declaração de Modos para o domínio Alzheimer:

modeh(1,great_ne(+a,+a)).	modeb(*,x_subst(+a,-n,-b)).
modeb(*,alk_groups(+a,-n)).	modeb(*,r_subst_1(+a,-l)).
modeb(*,r_subst_2(+a,-m)).	modeb(*,r_subst_3(+a,-n)).
modeb(*,ring_substitutions(+a,-n)).	modeb(*,ring_subst_1(+a,-b)).
modeb(*,ring_subst_2(+a,-b)).	modeb(*,ring_subst_3(+a,-b)).
modeb(*,ring_subst_4(+a,-b)).	modeb(*,ring_subst_5(+a,-b)).
modeb(*,ring_subst_6(+a,-b)).	modeb(*,polar(+b,-c)).
modeb(*,size(+b,-d)).	modeb(*,flex(+b,-e)).
modeb(*,h_doner(+b,-f)).	modeb(*,h_acceptor(+b,-g)).
modeb(*,pi_doner(+b,-h)).	modeb(*,pi_acceptor(+b,-i)).
modeb(*,polarisable(+b,-j)).	modeb(*,sigma(+b,-k)).
modeb(*,n_val(+a,-n)).	modeb(*,gt(+n,-n)).
modeb(*,great_polar(+c,-c)).	modeb(*,great_size(+d,-d)).
modeb(*,great_flex(+e,-e)).	modeb(*,great_h_don(+f,-f)).
modeb(*,great_h_acc(+g,-g)).	modeb(*,great_pi_don(+h,-h)).
modeb(*,great_pi_acc(+i,-i)).	modeb(*,great_polari(+j,-j)).
modeb(*,great_sigma(+k,-k)).	

Declaração de Modos para o domínio UWCSE:

modeh(*,advisedby(+student,+professor)).
modeb(*,taughtby(+course,-professor,-date)).
modeb(*,taughtby(-course,+professor,-date)).
modeb(*,courselevel(+course,-level)).
modeb(*,courselevel(+course,#level)).
modeb(*,position(+professor,-faculty)).
modeb(*,position(+professor,#faculty)).
modeb(*,position(-professor,+faculty)).
modeb(*,projectmember(+project,-professor)).
modeb(*,projectmember(-project,+professor)).
modeb(*,projectmember(+project,-student)).
modeb(*,projectmember(-project,+student)).
modeb(*,phase(+student,-phase)).
modeb(*,phase(+student,#phase)).
modeb(*,phase(-student,+phase)).
modeb(*,tempadvisedby(-student,+professor)).
modeb(*,tempadvisedby(+student,-professor)).
modeb(*,yearsinprogram(+student,-years)).
modeb(*,yearsinprogram(+student,#years)).
modeb(*,yearsinprogram(-student,+years)).
modeb(*,ta(+course,-student,-date)).
modeb(*,ta(-course,+student,-date)).
modeb(*,ta(-course,-student,+date)).
modeb(*,professor(+professor)).
modeb(*,student(+professor)).
modeb(*,publication(+ref,-professor)).
modeb(*,publication(-ref,+professor)).
modeb(*,publication(+ref,-student)).
modeb(*,publication(-ref,+student)).
modeb(1,geq(+number,#number)).
modeb(1,geq(+number,+number)).
modeb(1,diff(+student,+student)).
modeb(1,diff(+professor,+professor)).
modeb(1,diff(+project,+project)).
modeb(1,diff(+course,+course)).
modeb(1,diff(+ref,+ref)).
Regra no BK: diff(X,Y) :- X $\neq$ Y.

Declaração de Modos para o domínio Link_Discovery:

```
modeh(1, ckmurder(+murderForHire)).
modeb(1,eMailSending(+eMailSending)).
modeb(1,from_Generic(-wireTransferOfFunds,+bankAccount)).
modeb(1,from_Generic(+wireTransferOfFunds,-bankAccount)).
modeb(1,to_Generic(-wireTransferOfFunds,+bankAccount)).
modeb(1,to_Generic(+wireTransferOfFunds,-bankAccount)).
modeb(1,wireTransferOfFunds(+wireTransferOfFunds)).
modeb(1,receiverNumber(-makingAPhoneCall,+phoneNumber)).
modeb(1,receiverNumber(+makingAPhoneCall,-phoneNumber)).
modeb(1,callerNumber(-makingAPhoneCall,+phoneNumber)).
modeb(1,callerNumber(+makingAPhoneCall,-phoneNumber)).
modeb(1,makingAPhoneCall(+makingAPhoneCall)).
modeb(1,orgHitman(-mafiyaGroup_Russian,+person)).
modeb(1,orgHitman(+mafiyaGroup_Russian,-person)).
modeb(1,orgMiddleman(-mafiyaGroup_Russian,+person)).
modeb(1,orgMiddleman(+mafiyaGroup_Russian,-person)).
modeb(1,vor(-mafiyaGroup_Russian,+person)).
modeb(1,vor(+mafiyaGroup_Russian,-person)).
modeb(1,mafiyaGroup_Russian(+mafiyaGroup_Russian)).
modeb(1,vor(+mafiyaGroup_Russian,-person)).
modeb(1,bankAccount(+bankAccount)).
modeb(1,person(+person)).
modeb(1,employees(-business,+person)).
modeb(1,employees(+business,-person)).
modeb(1,relatives(-person,+person)).
modeb(1,relatives(+person,-person)).
modeb(1,ceo(-business,+person)).
modeb(1,ceo(+business,-person)).
modeb(1,hasMembers(-industry_Localized,+business)).
modeb(1,hasMembers(+industry_Localized,-business)).
modeb(1,hasMembers(-business,+person)).
modeb(1,hasMembers(+business,-person)).
modeb(1,hasMembers(-mafiyaGroup_Russian,+person)).
modeb(1,hasMembers(+mafiyaGroup_Russian,-person)).
modeb(1,business(+business)).
modeb(1,operatesinRegion(-industry_Localized,+russianSubregion)).
modeb(1,operatesinRegion(+industry_Localized,-russianSubregion)).
```

modeb(1,operatesinRegion(-business,+russianSubregion)).
modeb(1,operatesinRegion(+business,-russianSubregion)).
modeb(1,operatesinRegion(-mafiyaGroup_Russian,+russianSubregion)).
modeb(1,operatesinRegion(+mafiyaGroup_Russian,-russianSubregion)).
modeb(1,industry_Localized(+industry_Localized)).
modeb(1,agentPhoneNumber(-person,+phoneNumber)).
modeb(1,agentPhoneNumber(+person,-phoneNumber)).
modeb(1,accountHolder(-bankAccount,+person)).
modeb(1,accountHolder(+bankAccount,-person)).
modeb(1,opponents(-mafiyaGroup_Russian,+mafiyaGroup_Russian)).
modeb(1,opponents(+mafiyaGroup_Russian,-mafiyaGroup_Russian)).
modeb(1,gangWar(+gangWar)).
modeb(1,hitContractor(-murderForHire,+person)).
modeb(1,hitContractor(+murderForHire,-person)).
modeb(1,mediators(-murderForHire,+person)).
modeb(1,mediators(+murderForHire,-person)).
modeb(1,hitman(-murderForHire,+person)).
modeb(1,hitman(+murderForHire,-person)).
modeb(1,murderForHire(+murderForHire)).
modeb(1,deliberateActors(-planningTodoSomething,+person)).
modeb(1,deliberateActors(+planningTodoSomething,-person)).
modeb(1,planningTodoSomething(+planningTodoSomething)).
modeb(1,subEvents(-premeditatedMurder,+observing)).
modeb(1,subEvents(+premeditatedMurder,-observing)).
modeb(1,subEvents(-premeditatedMurder,+murder)).
modeb(1,subEvents(+premeditatedMurder,-murder)).
modeb(1,subEvents(-murderForHire,+premeditatedMurder)).
modeb(1,subEvents(+murderForHire,-premeditatedMurder)).
modeb(1,subEvents(-murderForHire,+makingAPhoneCall)).
modeb(1,subEvents(+murderForHire,-makingAPhoneCall)).
modeb(1,subEvents(-murderForHire,+eMailSending)).
modeb(1,subEvents(+murderForHire,-eMailSending)).
modeb(1,subEvents(-murderForHire,+paying)).
modeb(1,subEvents(+murderForHire,-paying)).
modeb(1,subEvents(-murderForHire,+meetingTakingPlace)).
modeb(1,subEvents(+murderForHire,-meetingTakingPlace)).
modeb(1,subEvents(-murderForHire,+planningTodoSomething)).
modeb(1,subEvents(+murderForHire,-planningTodoSomething)).
modeb(1,subEvents(-gangWar,+murderForHire)).

modeb(1,subEvents(+gangWar,-murderForHire)).
modeb(1,premeditatedMurder(+premeditatedMurder)).
modeb(1,eventOccursAt(-murder,+city)).
modeb(1,eventOccursAt(+murder,-city)).
modeb(1,deviceTypeused(+murder,+constant)).
modeb(1,crimeVictim(-murder,+person)).
modeb(1,crimeVictim(+murder,-person)).
modeb(1,crimeVictim(-premeditatedMurder,+person)).
modeb(1,crimeVictim(+premeditatedMurder,-person)).
modeb(1,crimeVictim(-murderForHire,+person)).
modeb(1,crimeVictim(+murderForHire,-person)).
modeb(1,murder(+murder)).
modeb(1,perpetrator(-observing,+person)).
modeb(1,perpetrator(+observing,-person)).
modeb(1,perpetrator(-murder,+person)).
modeb(1,perpetrator(+murder,-person)).
modeb(1,perpetrator(-premeditatedMurder,+person)).
modeb(1,perpetrator(+premeditatedMurder,-person)).
modeb(1,objectsObserved(-observing,+person)).
modeb(1,objectsObserved(+observing,-person)).
modeb(1,observing(+observing)).
modeb(1,moneyTransferred(-wireTransferOfFunds,+amount)).
modeb(1,moneyTransferred(+wireTransferOfFunds,-amount)).
modeb(1,toPossessor(-paying,+person)).
modeb(1,toPossessor(+paying,-person)).
modeb(1,payer(-paying,+person)).
modeb(1,payer(+paying,-person)).
modeb(1,paying(+paying)).
modeb(1,iteillocutionaryForce(+eMailSending,+constant)).
modeb(1,recipientOfinfo(-eMailSending,+person)).
modeb(1,recipientOfinfo(+eMailSending,-person)).
modeb(1,senderOfinfo(-eMailSending,+person)).
modeb(1,senderOfinfo(+eMailSending,-person)).
modeb(1,socialParticipants(-meetingTakingPlace,+person)).
modeb(1,socialParticipants(+meetingTakingPlace,-person)).
modeb(1,socialParticipants(-gangWar,+mafiyaGroup_Russian)).
modeb(1,socialParticipants(+gangWar,-mafiyaGroup_Russian)).
modeb(1,dateOfEvent(-makingAPhoneCall,+date)).
modeb(1,dateOfEvent(+makingAPhoneCall,-date)).

```
modeb(1,dateOfEvent(-wireTransferOfFunds,+date)).
modeb(1,dateOfEvent(+wireTransferOfFunds,-date)).
modeb(1,dateOfEvent(-eMailSending,+date)).
modeb(1,dateOfEvent(+eMailSending,-date)).
modeb(1,meetingTakingPlace(+meetingTakingPlace)).
modeb(1,geographicalSubRegions(-constant,+city)).
modeb(1,geographicalSubRegions(+constant,-city)).
modeb(1,geographicalSubRegions(-russianSubregion,+city)).
modeb(1,geographicalSubRegions(+russianSubregion,-city)).
```

Referências Bibliográficas

[1] H. Adé, B. Malfait e L. D. Raedt. "RUTH: an ILP theory revision system", In: *Proc. 8th Int. Symposium on Methodologies for Intelligent Systems (ISMIS-94) Berlin. Springer Verlag*, 1994.

[2] H. Arimura. "Learning acyclic first-order horn sentences from entailment"In: M. Li e A. Maruoka (Eds.), *Algorithmic Learning Theory*, v. 1316, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 432–445, 1997.

[3] L. Badea. "A Refinement Operator for Theories", In: *Proceedings of Inductive Logic Programming, 11th International Conference, ILP 2001*, pp. 1–14, 2001.

[4] L. Badea e M. Stanciu. "Refinement Operators Can Be (Weakly) Perfect", In: *Proceedings of the 9th International Worshop on Inductive Logic Programming*, pp. 21–32, 1999.

[5] I. Bratko. "Refining complete hypotheses in ILP", In: *Proceedings of the 9th International Conference on ILP, LNAI 1634,Springer*, pp. 44–55, 1999.

[6] M. A. Casanova, F. Giorno e A. Furtado. *Programação em Lógica e a Linguagem Prolog*. Edgard Blücher, 1987.

[7] Costa V., Damas L., Reis R., Azevedo R. "Yap Prolog", .

[8] L. De Raedt, N. Lavrac e S. Dzeroski. "Multiple Predicate Learning", In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp. 1037–1043, 1993.

[9] A. L. Duboc. *Utilizando a Cláusula mais Específica e Declaração de Modos na Revisão de Teorias de Primeira Ordem a partir de Exemplos*. Dissertação de Mestrado, COPPE - UFRJ, Rio de Janeiro, RJ, 2008.

[10] A. L. Duboc, A. Paes e G. Zaverucha. "Using the bottom clause and mode declarations in FOL theory revision from examples", *Machine Learning*, v. 76, n. 1, pp. 73–107, 2009.

[11] F. Esposito, N. Fanizzi, S. Ferilli e G. Semeraro. "A Generalization Model Based on OI-implication for Ideal Theory Refinement", *Fundam. Inform.*, v. 47, n. 1-2, 2001.

[12] F. Esposito, G. Semeraro, N. Fanizzi e S. Ferilli. "Multistrategy Theory Revision: Induction and Abduction in INTHELEX", *Machine Learning*, v. 38, n. 1-2, pp. 133–156, 2000.

[13] N. Fanizzi, S. Ferilli, N. D. Mauro e T. M. A. Basile. "Spaces of Theories with Ideal Refinement Operators", In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pp. 527–532, 2003.

[14] R. D. King, M. J. E. Sternberg e A. Srinivasan. "Relating Chemical Activity to Structure: An Examination of ILP Successes." *New Generation Computing*, v. 13, n. 3-4, pp. 411–433, 1995.

[15] R. Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection", In: *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*, pp. 1137–1145, 1995.

[16] N. Lavrac e S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.

[17] P. Liberatore. "Redundancy in logic III: Non-monotonic reasoning", *Artif. Intell.*, v. 172, n. 11, pp. 1317–1359, 2008.

[18] J. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987.

[19] F. E. P. e. M. C. M. M. Ferro. *O Sistema de Programação Lógica Indutiva Aleph - Características e Funcionamento*. Relatório Técnico 300, Universidade de São Paulo, http://www.icmc.usp.br/~biblio/index.php?destino=relatorios_tecnicos.php, 2007.

[20] H. Midelfart. "A Bounded Search Space of Clausal Theories", In: *Proceedings of Inductive Logic Programming, 9th International Workshop, ILP-99*, pp. 210–221, 1999.

[21] R. Mooney, P. Melville, L. Tang, J. Shavlik, I. Dutra, D. Page e . V. S. Costa. "Relational Data Mining with Inductive Logic Programming for Link Discovery", In: *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining, Baltimore*. AAAI/MIT Press, 2002.

[22] S. Muggleton. "Inductive logic programming", *New Generation Computing*, v. 13, n. 4, pp. 245–286, 1991.

[23] _____. *Inductive logic programming.* Academic Press, New York, 1992.

[24] _____. "Inverse Entailment and Progol", *New Generation Computing Journal*, v. 13, pp. 245–286, 1995.

[25] S. Muggleton. "Inductive Logic Programming", In: *The MIT Encyclopedia of the Cognitive Sciences (MITECS), MIT Press*, 1999.

[26] S. Muggleton e L. De RAEDT. "Inductive Logic Programming: Theory and Methods", *Journal of Logic Programming*, v. 19, n. 20, pp. 629–679, 1994.

[27] S. Muggleton e C. Feng. "Efficient induction of logic programs", In: *S. Muggleton, editor, Inductive Logic Programming. Academic Press, London*, pp. 281–298, 1992.

[28] S. Muggleton, A. Paes, V. S. Costa e G. Zaverucha. "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples", In: *Revised Papers of the 19th International Conference on Inductive Logic Programming.*, v. 5989, *Lecture Notes in Computer Science*, pp. 123–130. Springer, 2010.

[29] S. Muggleton, J. C. A. Santos e A. Tamaddoni-Nezhad. "ProGolem: A System Based on Relative Minimal Generalisation", In: *ILP*, pp. 131–148, 2009.

[30] C. Nadeau e Y. Bengio. "Inference for the Generalization Error", *Machine Learning*, v. 52, n. 3, pp. 239–281, 2003.

[31] S.-H. Nienhuys-Cheng e R. de Wolf. *Foundations of Inductive Logic Programming*, v. 1228, *Lecture Notes in Computer Science.* Springer, 1997.

[32] A. Paes. *On th Effective Revision of (Bayesian) Logic Programs from Examples.* Tese de Doutorado, Federal Unviersity of Rio de Janeiro, Rio de Janeiro, Brazil, 2011.

[33] A. Paes, G. Zaverucha e V. S. Costa. "Revising First-order Logic Theories from Examples through Stochastic Local Search", In: *17th Annual International Conference on Inductive Logic Programming (ILP-2007), LNAI 4894, Springer*, pp. 200–210, 2008.

[34] J. Quinlan. "Learning logical definitions from relations", *Machine Learning*, v. 5, pp. 239–266, 1990.

[35] L. D. Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[36] L. D. Raedt e M. Bruynooghe. "Interactive concept learning and constructive induction by analogy", *Machine Learning*, v. 8, pp. 107–150, 1992.

[37] B. L. Richards e R. J. Mooney. "Automated Refinement of First-Order Horn-Clause Domain Theories", *Machine Learning*, v. 19, pp. 95–131, 1995.

[38] ———. "Learning Relations by Pathfinding", In: *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pp. 50–55, 1992.

[39] M. Richardson e P. Domingos. "Markov logic networks", *Machine Learning*, v. 62, n. 1-2, pp. 107–136, 2006.

[40] E. Y. Shapiro. *Algorithmic Program Debugging*. ACM Distinguished Doctoral Dissertations. The MIT Press, Cambridge, Mass., 1983.

[41] A. Srinivasan. *The Aleph manual*. Relatório técnico, Oxford University, http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html, 2001.

[42] A. Tamaddoni-Nezhad e S. Muggleton. "The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause", *Machine Learning*, v. 76, n. 1, pp. 37–72, 2009.

[43] A. Tamaddoni-Nezhad e S. Muggleton. "Stochastic Refinement", In: *Proceedings of Inductive Logic Programming - 20th International Conference, ILP 2010*, 2010.

[44] L. R. Tang, R. L. Mooney e P. Melville. "Scaling Up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism", In: *Proceedings of the KDD-2003 Workshop om Multi-Relational Data Mining. Washington, DC*, pp. 107–121, 2003.

[45] P. R. J. van der Laag e S.-H. Nienhuys-Cheng. "Existence and nonexistence of complete refinement operators", In: *ECML-94: Proceedings of the European conference on machine learning on Machine Learning*, pp. 307–322. Springer-Verlag New York, Inc., 1994.

[46] J. Wogulis. “Revising relational domain theories”, In: *Proceedings of the Eighth International Workshop on Machine Learning. San Mateo, CA: Morgan Kaufman*, pp. 462–466, 1991.

[47] S. Wrobel. “First-order Theory Refinement”In: L. D. Raedt (Ed.), *Advances in Inductive Logic Programming*, IOS Press, pp. 14–33, 1996.

[48] _____. “Concept formation during interactive theory revsion”, *Machine Learning*, v. 14, pp. 169–191, 1994.

[49] G. Zaverucha. “Apostila de Lógica - Notas de aula - material distribuido nos cursos Cos230 e cos705”, 2008.