ARTEMIO LUDWIG

λ-PROLOG

INTERPRETADOR E UNIFICAÇÃO DE ORDEM SUPERIOR

Tese de Doutorado apresentada ao Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Wagner C. Amaral
caraderi do

Departamento de Engenharia de Computação e Automação Industrial
Faculdade de Engenharia Elétrica - FEE
Universidade Estadual de Campinas - UNICAMP
Campinas - Outubro de 1992

À minha esposa,
Márcia,
companheira.

A minhas filhas,
Fabíola(13) e
Débora(12),
fontes de incentivo.

A meus pais,
Armindo e Irlanda

AGRADECIMENTOS

Este trabalho foi desenvolvido graças à política de treinamento da Universidade Federal de Viçosa, à estrutura de ensino e pesquisa da Universidade Estadual de Campinas (UNICAMP) e ao apoio financeiro da CAPES e FAPEMIG. Nestas três entidades encontram-se pessoas que merecem meu reconhecimento pela maneira desinteressada e amiga de conduzir suas atividades. Com a certeza de que muitos mais poderiam constar da lista, quero nominar:

Professor Wagner C. do Amaral, orientador e antes de tudo um grande motivador e exemplo. Sua serenidade me proporcionou a segurança do reconhecimento profissional e autoconfiança suficiente para levar a termo a tarefa.

Professor José Luiz Braga pelo acompanhamento, recomendações, sugestões e destacado interesse.

Professores do IMECC/UNICAMP que com amizade e sabedoria me transmitiram seus conhecimentos. No mesmo sentido ficam arrolados os professores do Departamento de Computação e Automação da FEE que fomentaram a formação acadêmica necessária para permitir a atividade científica.

Professores da FEE, Márcio L. Andrade Netto e Rafael Santos Mendes, e Prof. João Nunes de Souza da Universidade Federal de Uberlândia, membros da banca de avaliação, pelas sugestões ao trabalho, ensinamentos e estímulo.

Professor Leo Pini, que com entusiasmo me aceitou em seu programa de treinamento.

Mauro Carnassalle, Antonio A. C. Marcelo e outras pessoas do mesmo grupo que permitiram que minha passagem por Campinas não fosse apenas oportunidade de treinamento mas que contivesse momentos agradáveis de outra natureza.

Finalmente quero prestar uma homenagem às pessoas de meu refúgio. Minha esposa Márcia e minhas duas filhas Fabíola e Débora. Suportaram meus maus momentos, não raros, e se satisfizeram com o pouco que pude retribuir durante este período. Representaram e ainda representam a maior fonte de incentivo.

A todos aqueles cujos nomes não citei e que de alguma maneira contribuiram para a finalização deste trabalho expresso meus sinceros agradecimentos.

RESUMO

A implementação de interpretadores para a linguagem da Lógica de Ordem Superior (LOS) constitui-se num desafio ainda não vencido. Pode-se dividi-lo em duas partes: (a) tornar a linguagem mais *amigável* permitindo estimular sua adoção e (b) dotar as implementações com um desempenho que não sacrifique sua usabilidade.

A linguagem de programação para a LOS utiliza os conceitos do λ-cálculo e os recursos de tipificação de Russel. Sua sintaxe é portanto mais complexa do que aquela da Lógica de Primeira Ordem (LPO). A técnica de derivação de algum conhecimento em uma base de conhecimento, formalizada conforme suas regras, assemelha-se com aquela da LPO que é chamada de *resolução.* Qualquer sistema de provas em LOS deve atender a restrições de uma linguagem fortemente tipada e a unificação, que apresenta problemas de incompletude, pode gerar mais do que um unificador. Isto torna a pesquisa não-determinística e faz com que a derivação contenha mais uma fonte de retroencadeamento, quando comparada com modelo de programação em LPO.

Este trabalho expõe e analisa a implementação de um interpretador para a LOS sujeita a sentenças definidas positivas as quais contêm propriedades adequadas à mecanização semelhantes àquelas das cláusulas de HORN da LPO. Problemas de indecidibilidade são contornados e os vários aspectos computacionais são descritos como a formação da base de conhecimento, sua gramática e a P-derivação, que é o método de obtenção de provas sobre a base.

Central a estes procedimentos encontra-se a unificação que apresenta complexidade elevada e cujo processamento é razão de depauperação na qualidade do sistema. Uma proposta de enfoque alternativa é oferecida com o propósito de atenuar os efeitos sobre a lentidão do sistema. O algoritmo resultante indica ganhos no desempenho e apresenta uma interpretação mais facilitada do mecanismo da unificação.

ABSTRACT

Efficient Interpreter implementations for Higher Order Logic (HOL) programming language still remain as a challenge that can be splitted into two classes: (a) to make the language friendlier in order to stimulate the user to adopt it, and (b) to provide the interpreter with a performance that does not damage its usability.

This paper resulted from an HOL interpreter prototypation, called λ-PROLOG. Only the positive definite sentençes are used in the language. They are similar to those of the First Order Logic (FOL) programing language and present some apropriate mechanization properties.

One kind of implementation is discribed and its computational aspects are analysed. They include the knowledgement base formation, the grammar of the sentences and the P-derivation, which is the method of deriving proofs over the base.

The unification is the most important interpreter procedure and has high complexity. Its processing is one of the reazons for the depaupering of system quality. In this paper the analisies of the MATCH and SIMPL algorithms that were developed for this purpose are used to supply a new unification algorithm which, in contrast to MATCH, shows initial performance gains and an easier procedural interpretation of the unification mechanism.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Esta introdução expõe a motivação para o desenvolvimento deste trabalho, seus antecedentes, sua importância, e o seu conteúdo.

A principal tarefa para quem se dispõe resolver um problema computacionalmente é encontrar uma maneira de representar o problema. A representação requer algumas propriedades cujo atendimento conduzem à segurança com relação à correção dos resultados obtidos quando trabalhada. Por outro lado, deseja-se que se somem a estas propriedades, características que a tornem assimilável pelo usuário final e que atendam à mecanização.

Há várias tendências quanto à melhor maneira de representar e manipular conhecimento. Uma delas, e com muita influência, advoga a representação usando notações formais que usam o formalismo da lógica de primeira ordem [Hofs80]. Há ainda algumas linhas de pesquisa que trabalham com representação do conhecimento em ambientes de incerteza, com base probabilística.

A lógica tem sido usada como linguagem para suprir esta necessidade. Notadamente, desde que J. A. Robinson [Robi65] introduziu um procedimento completo de refutação para o cálculo de predicados de primeira ordem, esforços tem sido colocados com o propósito de refinar a idéia inicial para tornar o procedimento mais eficiente, pelos menos em alguns contextos [Andr71].

No entanto a lógica de primeira ordem não se mostra adequada para representar com simplicidade e naturalidade alguns conceitos matemáticos e de outras áreas de conhecimento. A axiomatização da teoria dos conjuntos é um exemplo, onde [Free83] falhou. Especialmente onde o conhecimento versa sobre conjuntos e suas propriedades, tem-se problemas de representação com a lógica de primeira ordem. Há, nestes casos, uma motivação para o uso de um simbolismo mais adequado que incorpore a métodos de classificar os termos conforme seus tipos. Em contraste a lógica de ordem superior dispõe de recursos que ampliam este poder de representação.

A maior diferença entre os dois paradigmas reside no uso dos *termos* do λ-cálculo, nos quais os predicados e funções podem aparecer como variáveis. Nessa lógica há uma distinção explícita entre as expressões, as quais denotam diferentes tipos de objetos, favorecendo a unificação. Estes termos podem representar relacionamentos lógicos entre classes de objetos dentro de tal linguagem.

Os relacionamentos [Hatc68] fornecem um modelo para a teoria dos tipos que é a hierarquia. O modelo parte de um conjunto de objetos de um dado domínio chamados de indivíduos, ou átomos, ou objetos elementares, que estariam no nível mais baixo da hierarquia. Em seguida são considerados os conjuntos de indivíduos e relações entre eles. A coleção de todos estes conjuntos formam a segunda classe da hierarquia. Na sequência são considerados os conjuntos e relações sobre a hierarquia anteriormente formada que forneceria uma nova classe hierárquica. Com a iteração deste processo, obtém-se uma estrutura hierárquica de conhecimento fundamentada no conjunto elementar.

Esta percepção apresenta o atrativo imediato da expansão do universo a ser representado, desde que devidamente formalizado, além de espelhar realidades frequentes do mundo real.

Partindo-se da teoria de tipos de Russel [Russ03] e dos fundamentos do sistema lógico definido por Church [Chur40] com versões apresentadas por diversos pesquisadores ([Andr71], [Henk63]) definiu-se o *λ-cálculo tipado.* Este, por sua vez, foi utilizado como ferramenta principal na formalização da lógica de ordem superior.

De um paradigma já bem estudado que é o da Lógica de Primeira Ordem (LPO), passou-se a um novo formalismo que exigiu estudos de decidibilidade e completude dentre outros [Huet75]. Investigada com maior intensidade, são reflexos desta preocupação os vários relatos encontrados na literatura sobre o assunto ([Lucc72], [Nada87], [Huet73], [Gold81]).

A escolha da Lógica de Ordem Superior (LOS) para a representação e processamento do conhecimento, mesmo que em situações especiais, não obteve unanimidade em sua aprovação. Os fatores desfavoráveis são de dois tipos principais: (a) teóricos, como a incompletude da lógica de ordem superior e a equivalência da programação lógica de primeira ordem à máquina de Turing, dispensando a lógica de ordem superior; (b) práticos, como as dificuldades de implementação encontradas na construção de interpretadores e a ausência de referências sobre o assunto. Soma-se a isto a argumentação feita em [Warr82], onde se questiona a necessidade de extensões de ordem superior para a linguagem PROLOG. Estes argumentos contrastam com vantagens inegáveis do sistema que incluem a eliminação de paradoxos [Hofs80], a representação de meta-conhecimento [Mill85] e seu possível uso em diversas áreas como em Sistemas Especialistas.

Algumas das dificuldades acima provocaram o surgimento de novas propostas de sistemas que, para contornar os problemas, limitam o universo das sentenças usadas pela lógica de ordem superior, ou detectam as circunstâncias causadoras de indecidibilidade, sugerindo que pode haver um atendimento dos requisitos de exequibilidade sem um comprometimento irrecuperável das exigências de correção. Isto tornaria o sistema aproveitável computacionalmente. Neste contexto, embora com críticas quanto seu desempenho operacional e pouco ilustrados do ponto de vista computacional, alguns interpretadores para programação lógica de ordem-superior foram desenvolvidos ([Huet75], [Nada87] e [Nada89]).

O sistema formal baseado na lógica de ordem superior apresentado por [Nada90] tem servido de base para a implementação de sistemas experimentais . No entanto nada se tem publicado a respeito de técnicas de implementação da linguagem λ-Prolog resultante, exceção feita a [Nada89]. Tal sistema se baseia nas fórmulas chamadas sentenças definidas

de ordem superior, que são uma extensão, para a ordem superior, das bem conhecidas cláusulas de HORN.

Em razão do exposto permanecem ainda não devidamente esclarecidos alguns tópicos vinculados principalmente ao desejo de se explorar a metodologia para resolver problemas práticos e de melhor entender as vantagens de uma boa implementação. As implementações apresentam ainda problemas não resolvidos de compartilhamento de espaço, tempo de resposta, interface para orientação e entendimento do usuário, flexibilidade dos λ-termos sujeitos a substituição e posterior volta à estrutura inicial.

Correlacionadas à implementação estão a otimização das formas de declaração de tipos que são restritivas no contexto computacional e o controle da unificação que, em certas ocasiões, pode requerer a intervenção do usuário e pode ainda merecer do implementador uma indicação dos unificadores que ele prefere ver primeiro utilizados.

Problemas centrais, segundo Nadathur [Nada90], são a questão de se encontrar estruturas de dados adequadas para representar os λ-termos e o controle da unificação, que é um processo bem mais complicado do que aquele da primeira ordem.

Estes preliminares indicam que há uma suposta potencialidade para a programação de ordem superior cuja adaptação e aproveitamento não foram devidamente explorados. A despeito de que ainda sejam obscuras as vantagens práticas do aproveitamento das características funcionais desta representação e da não clareza na base lógica [Nada89] da linguagem, e ainda que em termos de aplicações a teoria ainda não tenha encontrado sua contrapartida prática, o interesse por tal formalismo continua, na busca de situações em que seu uso aproveite de fato as vantagens não contestadas da representação proposta.

Paralelamente é necessário tornar a teoria um pouco mais acessível ao usuário para que se torne aproveitável e não fique restrita a um pequeno universo de pesquisadores capazes de entendê-la. Nas mais recentes literaturas, os grupos interessados na área levantam problemas de dificuldades em tratar os "mistérios" que envolvem a ordem superior [Nada89], dada sua complexidade sabidamente maior. É difícil escrever programas que usam a unificação de ordem superior. A maneira de pensar é diferente e somente depois de algum tempo é que o pesquisador ou usuário começa perceber suas vantagens. No entanto já existem várias experiências de uso e esta atividade vai fornecendo argumentos para mostrar que vale a pena continuar com a técnica [Mill85].

Independentemente desta situação, a implementação de um interpretador constitui num objetivo interessante por dois motivos:

1. Em primeiro lugar é um excelente exercício de aplicação de toda a teoria que constitui o sistema. O desenvolvimento do interpretador contempla ainda as características não consideradas na formalização da linguagem. A exequibilidade computacional, incluindo aí o uso da memória, tempo de processamento, interação com o usuário, etc, só serão testadas e melhor entendidas com sua implementação. Há que se propor modelos que abriguem a forma de desenvolver o interpretador visando a obter as qualidades desejáveis, especialmente de velocidade e de interação com o usuário.

2. Em segundo lugar, uma vez desenvolvido, representa importante ferramenta para o próprio estudo de possíveis aplicações.

A continuidade de tais estudos é freada pela ausência de um interpretador para a linguagem. Este permitiria que testes fossem executados e conjecturas fossem testadas com maior desenvoltura, evitando-se trabalhos manuais infindáveis, haja vista o caráter altamente recursivo do processo. Sua existência aumentaria a quantidade de pesquisas relacionadas ao aproveitamento da proposta.

A disponibilidade de um interpretador para um estudo do uso da lógica de ordem superior somente torna-se possível graças a uma análise pormenorizada de sua fundamentação.

A fundamentação passa pelo entendimento da aplicação e das teorias envolvidas. Compõe este quadro a lógica de primeira ordem ([Klee67], [Lloy84] ,[Ster86] e [Shoe67]), a teoria dos tipos ([Andr71], [Chur40]), o lambda cálculo ([Bare85],[Hind86]), o lambda cálculo tipado [Henk63] e a formalização de sistemas de lógica de ordem superior usando o λ-cálculo tipado ([Huet73], [Nada87]). A restrição destes sistemas a subconjuntos de fórmulas [Nada90] e a mecanização de teorias sob tais sistemas [Huet75] representam o estágio sobre o qual se assenta a área no momento e, a partir de onde se pretende avançar na possibilidade do uso da programação de ordem superior com o presente trabalho.

O objetivo principal deste trabalho foi desenvolver um interpretador para a programação em lógica de ordem superior chamado λ-PROLOG. Além de passar por caminhos intermediários, que incluem a interpretação e o entendimento da linguagem lógica proposta, ênfase é colocada na descrição de uma proposta de estrutura de implementação e na detecção dos problemas existentes e que devem ser resolvidos.

Como marco inicial de pesquisa o trabalho contemplou os aspectos gerais do interpretador contrastando, quando possível, com a metodologia usada no PROLOG padrão. Os itens de caráter mais específico foram relatados e indicados como pontos de possível desenvolvimento.

Desta forma são enfatizadas as duas tarefas que diferenciam substancialmente o provador de ordem superior daquele da lógica de primeira ordem: a unificação, em si mesma, e a maneira de se tratar a nova forma de retroencadeamento [1] proveniente da provável existência de vários unificadores.

Como uma primeira consequência do esforço da implementação, e já cumprindo um dos objetivos para os quais foi desenvolvida, propõe-se uma nova abordagem para a unificação de termos de ordem superior. A proposta é formalizada como um algoritmo que apresenta vantagens significativas em termos de desempenho.

Inicialmente é feita uma apresentação da Lógica de Ordem Superior conduzida de modo a passar o entendimento necessário para a compreensão dos mecanismos que possibilitaram sua mecanização. Buscou-se, neste estágio do trabalho, esclarecer aspectos pouco explorados e por isso ainda um pouco obscuros, dada a sua complexidade (como a formação de termos, aplicações e manuseio de tipos), em sua formalização, e que aumentam as dificuldades de sua manipulação e de seu uso computacional.

Os detalhes da tipificação e formação de sentenças foram expostos e ilustrados didaticamente com a intenção de torná-las mais facilmente assimiláveis. Com a mesma preocupação

[1] Retroencadeamento é o retorno de uma pesquisa em árvore para o nó em que se deixou um ramo que pode ainda ser testado.

são vistos os mecanismos de λ-conversão, que permitem a manipulação de fórmulas pertencentes a uma mesma classe de equivalência.

Num segundo estágio são levantados os vários aspectos da unificação dos termos de ordem superior e da P-resolução que é a denominação dada ao mecanismo que resolve consultas sobre uma base de conhecimento formalizada com uma linguagem de ordem superior.

A semelhança do procedimento da P-derivação com a resolução na programação lógica de primeira ordem é examinada. Dado um modelo geral de resolução de consulta em primeira ordem, e considerando-se a unificação um procedimento à parte que devolve um conjunto de substituições, a P-derivação torna-se a resolução com um acréscimo de um controle a mais para o retroencadeamento sobre todas as substituições possíveis. O modelo geral de implementação do PROLOG é investigado sob a ótica de que a inclusão das noções de ordem superior não comprometem o modelo. No entanto, delineam-se as dificuldades existentes na unificação e percebe-se a existência de segmentos que demandam pesquisas para otimização.

Detalhes do mecanismo de controle da pesquisa na base são fornecidos com o objetivo de contribuir para a melhoria dos processos que otimizem o sistema em termos de tempo e de recursos de memória.

Em seu contexto mais amplo o trabalho se apresenta como uma base inicial onde se fundamenta a programação lógica de ordem superior e um relato das experiências obtidas com o desenvolvimento do interpretador. É um elemento de encorajamento para pesquisas subsequentes e contribui fornecendo caminhos menos árduos para seu aproveitamento. Contribuição específica é dada no sentido de melhorar o desempenho de interpretadores oferecendo um algoritmo para a unificação com características diferentes daquelas usadas até o momento.

Em seguida é apresentado um esboço do trabalho com uma descrição sucinta de cada capítulo.

Uma descrição resumida da lógica de primeira ordem que contém o que se acredita necessário para o entendimento dos capítulos posteriores é feita no capítulo 2. No capítulo 3 apresenta-se a lógica de ordem superior fazendo-se a interligação entre fórmulas bem formadas e a tipificação dos termos. Em seguida, capítulo 4, restringe-se o universo destas fórmulas para um subconjunto chamado de fórmulas definidas de ordem superior, mais adequadas para o tratamento computacional, à semelhança de suas congêneres de primeira ordem. No capítulo 5 passa-se para a programação lógica de ordem superior que contempla os aspectos de unificação e P-derivação que são os mecanismos básicos para a implementação do interpretador. Neste capítulo mostra-se uma proposta geral de modelo de implementação, contrastando-a com o modelo geral do PROLOG padrão abordando-se as dificuldades encontradas na implementação e as respectivas opções escolhidas para contorno ou solução do problema. No capítulo 6 formaliza-se o novo algoritmo para a unificação que, em contrastes iniciais, apresenta vantagens computacionais significativas.

Conclusões e perpectivas são apresentadas no capítulo 7. O texto contém 2 apêndices: o primeiro com o rastreio da P-resolução para um exemplo de consulta e o segundo com uma unificação de duas fórmulas de ordem superior usadas para melhor acompanhar o

desenvolvimento do algoritmo proposto.

Capítulo 2

Lógica Proposicional e de Predicados

2.1 Introdução

Existem diferentes maneiras de representar qualquer tipo de conhecimento. Alguns fatores interferem na escolha do formalismo escolhido. Para cada *fato* (ou verdade) de determinado universo existe uma *representação*. A *manipulação do conhecimento* se dá, indiretamente, por meio da *manipulação da representação* que foi escolhida para o mesmo. Há portanto um mapeamento entre o fato e sua representação que garante que esta possa ser usada e que o resultado da operação sobre ela resulte em verdades ou fatos do mundo em questão.

As sentenças da linguagem natural constituem uma certa representação coloquial. Outro formalismo representacional é a lógica matemática, e é sobre esta maneira particular de representar os fatos que se volta a atenção neste trabalho. Neste capítulo, em especial, descreve-se a linguagem da lógica proposicional e da lógica de primeira ordem. Elas representam um arcabouço, já bastante estudado em seus aspectos formais ([Klee67], [Lloy84]), que suporta avanços para uma lógica de ordem ω, ou lógica de ordem superior (a ser explicada mais adiante) que, por sua vez permite representar de modo mais direto do que o das primeiras, certos conhecimentos, conservando suas características. Além disto, a mecanização deste simbolismo, usado para explorar bases de conhecimento com objetivo de derivar novos conhecimentos, tem obtido sucessos crescentes a partir de 1972, com a implementação da lógica como linguagem de programação. Como o interesse é explorar a possibilidade de mecanização da lógica de ordem superior, experiências na lógica de primeira ordem sugerem técnicas que podem ser aproveitadas.

Inicia-se a discussão apresentando o formalismo da representação da lógica, descrito basicamente pela definição de fórmulas bem formadas; segue-se analisando um universo de formas padrão em que o mecanismo operacional do procedimento de provas é menos complexo, e formalmente completo ao se tomar as cláusulas definidas, para ao final discutir os algorítmos da unificação e da resolução que são os principais procedimentos envolvidos

na mecanizaçào desta lógica com fins de manipulação do conhecimento, especialmente de busca de provas. As definições, os conceitos, e explicaçoes mais detalhadas podem ser encontradas em livros textos como [Kowa79], [Chan73], [Lloy84] e [Klee67].

Este capítulo oferece uma macro visão de uma base de conhecimento pertencente a um sistema de produção, que usa a lógica como forma de representação, e sobre o qual se deseja atuar. Para isso necessita-se de algumas regras e de um mecanismo de inferência.

Espera-se que este resumo sobre alguns tópicos da lógica e programação de primeira ordem facilitem o entendimento dos capítulos seguintes, que tratarão de lógica de ordem superior, nos quais aparecerão comparações entre as programações nas duas lógicas. Descreve-se ainda as cláusulas de Horn, a interpretação procedural de Kowalski [Kowa79] e mostra-se a resolução como regra de inferência. Na última seção é feita uma explanação geral de como são implementados os interpretadores para a programação lógica de primeira ordem.

2.2 A Lógica Proposicional

Na lógica proposicional, os elementos básicos da representação do conhecimento são os *átomos* por meio dos quais constroem-se as fórmulas usadas para expressar idéias mais complexas.

Um *átomo* representa uma sentença declarativa que pode ser ou verdadeira ou falsa, mas não ambos. O átomo é tratado como uma unidade simples. Existem muitos fatos que podem ser tratados pela lógica proposicional. Por exemplo:

Fato	Atomo
Esta chovendo	Chovendo
E inverno	Inverno
Esta frio	Frio

A lógica proposicional permite que outros fatos sejam representados usando operadores sobre átomos. Para isto requer-se a definição de como se formam tais sentenças, utilizando-se os símbolos lógicos que são : $\sim$ (negação), $\wedge$ (conjunção), $\vee$ (disjunção), $\Rightarrow$ (implicação) e $\Longleftrightarrow$ (bi- condicional). Uma vez que a lógica proposicional é limitada para representar alguns fatos, cita-se apenas alguns exemplos de fórmulas bem formadas na lógica proposicional :

$$\text{Chovendo} \wedge \text{Inverno}$$
$$\sim \text{Inverno}$$
$$(\text{Frio} \vee \text{Inverno}) \Rightarrow \text{Chovendo}$$

A dedução de algum conhecimento de um conjunto de fórmulas da lógica proposicional depende do valor verdade destas fórmulas e não apresenta problemas de decisão, já que

o número de interpretações é finito. No entanto alguns fatos não podem ser bem representados nesta lógica. Especialmente os fatos ou regras que envolvem quantificadores e variáveis. Por exemplo, não há como representar na lógica proposicional a sentença,

$$\forall x \; maior(x, 5)$$

cuja leitura é "para todo x, x é maior do que 5 ".

Dois fatos que apresentam alguma correlação, teriam uma afinidade em termos de recuperação se fossem representados de outra forma. Por exemplo, usando a lógica de primeira ordem,

fato1 : Ligada (Prensa)
fato2 : Ligada (Caldeira)

representam que os dois equipamentos estão ligados. No entanto não é possível recuperar a informação de que os dois equipamentos ligados são prensa e caldeira, na lógica proposicional, pois a representaçàо dos fatos seria por exemplo:

fato1 : LigadaPrensa
fato2 : LigadaCaldeira

e não há ligação entre os dois fatos, em termos de representação.

2.3 A Lógica de Predicados ou de Primeira Ordem

Na formalização da linguagem lógica de primeira ordem os átomos são construídos usando quatro tipos de símbolos:

i) aqueles usados para identificar variáveis

ii) aqueles usados para identificar constantes

iii) aqueles usados para representar funções e

iv) símbolos para representar predicados.

Três noções lógicas importantes são usadas para definir as fórmulas bem formadas: termos, predicados e quantificadores.

Termo é definido recursivamente como segue:

i) Uma constante é um termo;

ii) Uma variável é um termo;

iii) Se f é um símbolo funcional de n argumentos, e $t_1, \cdots, t_n$ são termos, então $f(t_1, \cdots, t_n)$ é um termo;

iv) Todos os termos são criados por meio das regras acima.

Adotando-se a convenção de que *variáveis* são expressas pelas letras *x, y, z*, funcões por *f, g, h* ou cadeias de caracteres minúsculos e *constantes* são representadas por *a, Pedro e 11*, por exemplo, são termos: *x, a, 11, Pedro* por serem constantes ou variáveis. São termos ainda *f(11, x), g(f(11), x)* e outras definições funcionais.

Predicado é um mapeamento de uma lista de constantes em Verdadeiro, Falso. Por convenção, na Lógica de Primeira Ordem os predicados são expressos por cadeias de caracteres maiúsculos, PAI(João) expressa o mapeamento PAI, o predicado, aplicado a um elemento de um conjunto cujo nome é João sobre {verdade, falso}. PAI é um predicado que é aplicado a uma lista com 1 elemento. Quando a lista tiver n elementos, o predicado será n-ário, ou de aridade n.

Se P é um símbolo para um predicado n-ário e $t_1, \ldots, t_n$ são termos então $P(t_1, \ldots, t_n)$ é um átomo.

PAI(João) é um átomo assim como TIO(José, Ricardo). TIO é um predicado 2-ário.

Com os átomos pode-se então elaborar fórmulas bem formadas que serão usadas para descrever o conhecimento. Para a definição de fórmula são usados os conectivos lógicos já vistos na lógica proposicional $\wedge, \vee, \sim, \Rightarrow$ e $\Leftrightarrow$ e os quantificadores universal ($\forall$) e existencial ($\exists$).

São fórmulas bem formadas:

i) Um átomo é uma fórmula

ii) Se F e G são fórmulas, então $(\sim F), (F \vee G), (F \vee G), (F \Rightarrow G)$ e $(F \Leftrightarrow G)$ são fórmulas.

iii) Se F é uma fórmula e x é uma variável livre em F, então $(\forall x)F$ e $(\exists x)F$ são fórmulas.

iv) Fórmulas são geradas somente por um número finito de aplicações de (i), (ii) e (iii).

Os símbolos especiais $\forall$ e $\exists$ têm a interpretação semântica de "para todo" e "existe" respectivamente.

Estabelece-se assim uma linguagem para expressar fatos e regras do universo. São exemplos de sentenças bem formadas:

$\forall x\ ROMANO(x) \Rightarrow LEALA(x, Cesar)$
$\exists x\ \forall y\ AMA(x, y)$
HOMEM(João)
$\forall x\ FUNCIONA(x) \Rightarrow NOVO(x) \vee CONCERTADO(x)$
$MATOU(x, y) \Rightarrow$ DETESTA(x,João) $\wedge\ AMA(x, y)$

A conversão desta representação para frases em português é conhecida da maioria dos leitores. Vale ressaltar neste ponto que os valores aceitos para as variáveis nos mapeamentos funcionais e nos predicados devem pertencer a um domínio conhecido para que se possa conhecer o valor verdade das fórmulas bem formadas. A esta especificação de domínio e

correspondente mapeamento de funcões e predicados aos seus respectivos contradomínios chama-se *interpretação* (geralmente simbolizada por I).

Na lógica proposicional, a interpretação é uma associação de valores verdade a átomos. Na lógica de primeira ordem, desde que existem variáveis envolvidas, este processo não é direto.

Uma interpretação de uma fórmula F em lógica de primeira ordem consiste de um *domínio D* não vazio, e uma associaçào de valores para cada constante, símbolo funcional e símbolo predicativo que ocorrem em F como segue:

i) Para cada constante, é associado um elemento em D

ii) Para cada símbolo funcional n-ário, associa-se um mapeamento de D^n em D. (Note que $D^n = \{(x_1, \cdots, x_n)$ tal que $x_1 \in D, \cdots, x_n \in D\}$).

iii) Para cada símbolo n-ário de predicado, associa-se um mapeamento de D^n a {V, F}.

A interpretação de fórmulas se dá sobre um domínio D. Assim, quando temos $(\forall x)$ a fórmula será interpretada como "para todos os elementos x em D" , e $(\exists x)$ como "existe um elemento x em D". Para toda interpretação de uma fórmula sobre um domínio D, a fórmula pode ser avaliada em V ou F de acordo com os seguintes regras:

1. Se os valores verdade das fórmulas G e H estão disponíveis, então os valores verdade das fórmulas $\sim G$, $(G \wedge H)$, $(G \vee H)$, $(G \Rightarrow H)$, e $(G \Leftrightarrow H)$ são avaliados, conforme tabela verdade.
2. $(\forall x)\, G$ é avaliado ser V se o valor verdade de G é V para todo x em D. De outra maneira, ele é F.
3. $(\exists x)\, G$ é avaliado ser V se o valor verdade de G é V para pelo menos um x em D. De outro modo, ele é F.

Uma variável é livre se não estiver ligada a um dos quantificadores. Note que uma fórmula contendo variáveis livres não pode ser avaliada. Uma fórmula G é *consistente* se e somente se existe uma interpretação I na qual G é avaliada V. Se uma fórmula G é V em uma interpretação I, diz-se que I é um modelo de G e I satisfaz G. Uma fórmula G é inconsistente se e somente se não existe uma interpretação que satisfaça G. Ela será válida se e somente se toda interpretação satisfaz G.

Uma fórmula G é consequência lógica da fórmulas $F^1, F^2, \cdots, F^n$ se e somente se para toda interpretação I, quando $F^1 \wedge F^2 \wedge \cdots \wedge F^n$ for verdade, G também é verdade. Este é o *teorema da dedução* [Chan73].

A lógica de primeira ordem pode ser considerada como uma extensão da lógica proposicional. Quando uma fórmula em lógica de primeira ordem não contém variáveis e quantificadores, ela pode ser tratada como uma fórmula em lógica proposicional.

Em lógica de primeira ordem, por existir número potencialmente infinito de domínios, existirá um número também potencialmente infinito de interpretações para uma fórmula.

Um conjunto finito de fórmulas da lógica de primeira ordem é denominada de *base de conhecimento*. Sob o enfoque da teoria de primeira ordem ela representa o conjunto de axiomas e quando suas fórmulas obedecem a uma forma padrão chamada de *cláusula*, a ser definida abaixo, é um programa lógico.

O procedimento que realiza a operação de *provar* uma declaração sobre uma base de conhecimento requer que as fórmulas sejam convertidas numa forma padrão conveniente, chamada forma clausal. Esse procedimento, chamado de *resolução*, produz provas por *refutação*, ao demonstrar que a *negação* da declaraçào produz uma contradição com as declarações da base (portanto não pode ser satisfeita).

Para simplificar os procedimentos de prova e atingir a linguagem de cláusulas, deve-se estabelecer inicialmente a forma normal prenex de uma fórmula. Uma fórmula F em lógica de primeira ordem é dita estar na forma normal *prenex* se e somente se a fórmula F está na forma:

$$(Q_1 x_1) \cdots (Q_n x_n)(M)$$

onde todo $Q_i x_i$, para $i = 1, \cdots, n$ é ou $(\forall x_i)$ ou $(\exists x_i)$, e M é uma fórmula que não contém quantificadores. $(Q_1 x_1), \cdots, (Q_n x_n)$ são chamados de *prefixo* e M é chamado de *matriz* da fórmula F.

Existe um algorítmo que transforma qualquer fórmula bem formada em uma fórmula na forma normal prenex. O mesmo algorítmo pode ser aumentado para transformar estas fórmulas em uma forma normal conjuntiva, na qual cada fórmula tem sua matriz M formada por um conjunção de disjunções.

A fórmula F abaixo é uma conjunçào de disjunções:

$$\begin{aligned}&(\text{CASADO(x)} \vee \text{JAPONES(x)})\\ &\wedge (\text{DEMOCRATA(x)} \vee \text{LIBERAL(x)})\\ &\wedge (\text{CAPAZ(x)})\end{aligned}$$

No exemplo acima, não há quantificadores.

Após a prefixação os quantificadores existenciais são eliminados por meio da utilização das funções de Skolem e a fórmula conjuntiva (prenex) resultante fica apenas com quantificadores universais. Forma-se um conjunto com todas as fórmulas que estão unidas pelo conectivo $\wedge$. Se todas estas fórmulas são universalmente quantificadas os quantificadores são dispensáveis. Pode-se portanto ver um conjunto de fórmulas ou base de declarações como uma conjunto universalmente quantificado de disjunções. No exemplo

$$\begin{aligned}\text{F} : \{&(\text{CASADO(x)} \vee \text{JAPONES(x)}),\\ &(\text{DEMOCRATA(x)} \vee \text{LIBERAL(x)}),\\ &(\text{CAPAZ(x)})\}\end{aligned}$$

as fórmulas que são uma disjunção de literais (átomos ou negações de átomos) são chamadas de *cláusulas*. Após estas transformações está-se apto a explorar o procedimento da resolução para provar declarações.

2.4 A Unificação e a Resolução na Lógica de Primeira Ordem

Church e Turing ([Chur36], [Turi36]) mostraram, independentemente, que não existe um procedimento geral de decisão para verificar a validade de fórmulas em lógica de primeira ordem. Este é o teorema da decidibilidade da lógica. No entanto existem procedimentos de prova que podem verificar se uma fórmula é válida quando realmente ela é válida. Para fórmulas não válidas, estes procedimentos em geral nunca terminarão.

Por definição, uma fórmula *válida* é verdadeira sob todas as interpretações. Em 1930, Herbrand [Herb30] desenvolveu um algoritmo para encontrar uma interpretação que pode falsificar uma fórmula dada. No entanto, se a fórmula dada for realmente válida não existirá interpretação falsa e o algorítmo para após um número finito de tentativas. O método de Herbrand é a base para o moderno procedimento automático de provas.

O procedimento de prova por resolução em vez de produzir uma dedução de uma fórmula a partir da base inferindo sua validade, prova que a negação de uma fórmula é inconsistente com o conjunto de axiomas que descreve o problema. Esse procedimento é aplicável à base de conhecimento cujas sentenças obedecem às "formas padrões" de fórmulas, vistas na secção 2.3 e chamadas de cláusulas.

Quando se tem um conjunto S de declarações, todas na forma de cláusulas, pode-se considerar a conjunção de todos os elementos de S como uma fórmula F. Provar a inconsistência do conjunto de declarações (base de conhecimento) é o mesmo que provar a inconsistência de F (uma fórmula bem formada) na forma normal conjuntiva. Por exemplo, o conjunto (S) de declarações abaixo contém cláusulas que assume-se universalmente quantificadas:

$$\begin{aligned} &\text{PAR(8)} \\ &\sim PAR(x) \vee \text{Q(f(x))} \\ &\sim \text{Q(f(a))} \end{aligned}$$

e constitue uma fórmula F que é

$$\text{PAR(8)} \wedge (\sim\text{PAR(x)} \vee \text{Q(f(x))} \wedge \sim\text{Q(f(a))}$$

na forma normal conjuntiva.

Responder a uma consulta (representada por G) sobre S ou F corresponde a acrescentar ao conjunto S a negação da declaração que representa a consulta e provar que há uma interpretaçàо em que a fórmula G' (G acrescida da negação) é falsa.

Isto é equivalente a dizer que G' é inconsistente.

Se G' for inconsistente então para alguma interpretação ela não é válida. Pelo teorema da dedução ([Chan73]), dadas fórmulas $F^1, \cdots, F^n$ e uma fórmula G, G será uma consequência lógica de $F^1 \wedge \cdots \wedge F^n$ se e somente se $(F^1 \wedge \cdots \wedge F^n \wedge \sim\text{G})$ for inconsistente. G, no teorema, é a declaração considerada como consulta. A prova da finitude do processo bem como da existência da interpretação, caso G seja inconsistente, considera apenas um conjunto de cláusulas, chamado de cláusulas de Herbrand, na lógica de primeira ordem. Na lógica proposicional o universo das interpretações é finito e o processo é decidível.

Portanto no processo de resolução, apenas são consideradas as interpretações sobre o conjunto de cláusulas pertencentes ao universo de Herband.

O princípio da resolução pode ser aplicado diretamente a um conjunto S de cláusulas para testar se uma proposição é satisfeita em S. A idéia do uso do princípio da resolução é checar se uma cláusula vazia pode ser obtida de S acrescido da negação da proposição. Se ela puder ser obtida por meio do uso da resolução, então S não satisfaz a proposição. Pode-se visualizar o princípio da resolução como uma regra de inferência que pode ser usada para gerar novas cláusulas de S.

Sejam as seguintes cláusulas:

$$C_1 : \mathrm{P}$$
$$C_2 : \sim P \vee Q$$

A regra de um literal consiste em examinar se existe um par complementar de literais (por exemplo, P em C_1 e $\sim P$ em C_2) e então deletar C_1 e C_2 para obter a cláusula C_3 que é Q. Mais formalmente, o princípio da resolução diz que para duas cláusulas C_1 e C_2, se existe um literal L_1 em C_1 que é complementar a um literal L_2 em C_2, então delete L_1 e L_2 de C_1 e C_2, respectivamente, e construa a disjunção das cláusulas remanescentes em C_1 e C_2.

A cláusula construída é a *resolução de* C_1 *e* C_2. Prova-se que uma resolução C de C_1 e C_2 é uma consequência lógica de C_1 e C_2.

Dado um conjunto de cláusulas, deduzir C de S corresponde a encontrar uma sequência finita $C_1, C_2, \cdots, C_k$ de cláusulas tais que cada C_i ou é uma cláusula em S ou uma resolução de cláusulas que precedem C_i, e $C_k = C$. A dedução de uma *cláusula vazia* de S é uma *refutação* ou *prova* em S.

Na lógica proposicional, o princípio da resolução é aplicado a cláusulas que não contêm variáveis. Na lógica de primeira ordem, o processo é mais complicado. Nestes casos deve-se fazer uma substituição da variável por um termo de modo a permitir que os literais se tornem complementares. O procedimento para encontrar uma substituição chama-se *unificação.*

Uma instância de uma variável em uma cláusula consiste na substituição da variável por um termo da base de conhecimento. Com esta possibilidade, o princípio pode ser aplicado. Por exemplo, sejam:

$$C_1\text{: } P(x) \vee Q(x)$$
$$C_2\text{: } \sim P(f(x)) \vee R(x)$$

C_1 e C_2 não possuem literais complementares. Substituindo-se x por f(x) em C_1 tem-se:

$$C_1\text{'} : P(f(x)) \vee Q(f(x))$$

C_1' é uma instância de C_1. Neste ponto C_1' e C_2 possuem literais complementares que, pelo principio da resolução, podem ser usados para se obter a chamada *cláusula resolvente*:

$$C_3 = Q(f(x)) \vee R(x)$$

Uma *substituição* é um conjunto finito da forma $\{t_1/v_1, \cdots, t_n/v_n\}$, onde v_i é uma variável, todo t_i é um termo diferente de v_i, e nenhum par de elementos no conjunto possui

a mesma variável após o símbolo /. A substituição que não contém elementos é chamada de substituição vazia. Se $\theta = \{t_i/v_i, \cdots, t_n/v_n\}$ é uma substituição e E é uma expressão, então $E\theta$ será uma expressão obtida de E substituindo-se simultaneamente cada ocorrência da variável $v_i, 1 \leq i \leq n$, em E pelo termo t_i. E(θ) é chamada uma instância de E por θ.

A composição de duas substituições θ e $\lambda, segundo$[Chan73],, onde $\theta = \{t_1/x_1, \cdots, t_n/x_n\}$ e $\lambda = \{u_1/y_1, \cdots, u_n/y_n\}$ é a substituição obtida do conjunto:

$$\{t_1\lambda/x_1, \cdots, t_n\lambda/x_n, u_1/y_1, \cdots, u_n/y_n\}$$

por meio da deleção de todo elemento $t_j\lambda/xj$ onde $t_j\lambda = x_j$ e de todos os elementos u_i/y_i nos quais $y_i, \in \{x_1, x_2, \cdots, x_n\}$.

Uma substituição θ é chamada de um *unificador* para um conjunto de expressões $\{E_1, \cdots, E_k\}$ se a aplicação do *unificador* a cada expressão as torna idênticas. Um conjunto de expressões é dito ser unificável se existe um *unificador* para ele. Um *unificador* σ para um conjunto $\{E_1, \cdots, E_k\}$ de expressões, é o unificador mais geral se e somente se para cada unificador θ para o conjunto existe uma substituição λ tal que $\theta = \sigma\lambda$

A unificação permite então verificar se dois literais podem ser iguais e qual substituição seria necessária para isto. A noção de composição de substituições é necessária pois a resolução é uma sequência de eliminação de literais que se unificam e a sequência dos unificadores, do início do processo de resolução até o seu final, fica retida na composição, a cada passo, do unificador anterior com o atual. O unificador final, obtido por meio destas composições, representará a substituição que deverá ser feita nos axiomas acrescidos de $\sim$G para se mostrar a inconsistência da base assim formada.

Alem disto, a substituição necessária para que os dois literais sejam eliminados é aplicada, a cada passo, ao restante da base de dados, para que na sequência do desenvolvimento, as substituições anteriores sejam consideradas. Este processo assemelha-se a assumir premissas para que o procedimento possa prosseguir. Alguns aspectos deste procedimento de pesquisa são mostrados na seção seguinte.

2.5 Aspectos Procedurais da Resolução

A prova de teoremas na lógica de primeira ordem teve uma implementação bastante eficiente de um tipo de resolução chamada SL. A nomenclatura provem das palavras Linear e Seleçao presentes no nome do método chamado de resolução linear com função de seleção mais tarde acrescida de um D (SLD) simbolizando que atuava sobre cláusulas definidas. A implementação resultou no PROLOG em 1972 que trabalha com um subconjunto de fórmulas conhecido como cláusulas de HORN.

Como foi visto na secção 2.4 a forma de cláusulas é aquela onde a fórmula é escrita como uma disjunção de literais (átomos positivos ou negativos). Aproveitando as propriedades comutativa e associativa das conjunções separa-se os literais positivos dos negativos obtido da fórmula $L_1 \vee \cdots \vee L_n \vee \sim A_1 \vee \cdots \vee \sim A_m$ por meio da lei de De Morgan obtendo-se $(L_1 \vee \cdots \vee L_n) \vee \sim (A_1 \wedge \cdots \wedge A_m)$ e, pela substituição de $(A \Rightarrow B)$ por $(\sim A \vee B)$, obtem-se então

$$(L_1 \vee \cdots \vee L_n) \Leftarrow (A_1 \wedge \cdots \wedge A_m)$$

com a fórmula da esquerda sendo o *consequente*, ou *cabeça* da cláusula, e a parte da direita o *antecedente* ou *cauda*.

O processo de resolução de Robinson é aplicado às expressões na forma de cláusulas. Trata-se de um procedimento que combina duas cláusulas tratando de eliminar fórmulas em comum, desde que uma esteja no consequente e a outra esteja no antecedente das cláusulas em questão. Neste processo entra o mecanismo da unificação que permite o cancelamento apenas em situações particulares, descritas anteriormente.

Assim, dado um conjunto de cláusulas, se deseja-se provar uma cláusula, basta acrecentar sua negação ao conjunto e verificar se consegue-se gerar, por meio da resolução, a cláusula vazia.

As cláusulas de HORN apresentam sua cabeça constituída por, no máximo, uma fórmula positiva. Tem-se portanto como cláusulas de HORN

forma de implicação	forma de disjuncão de literais
$A \Leftarrow B \wedge C \wedge D$	$A \vee \sim B \vee \sim C \vee \sim D$
$A \Leftarrow$	A
$\Leftarrow B \wedge C$	$\sim B \vee \sim C$

As duas primeiras, que têm exatamente um literal positivo são chamadas de cláusulas definidas (ou regra e fato respectivamente) e a terceira será vista como fórmula objetivo.

Portanto espera-se que a base de conhecimentos esteja representada na forma de cláusulas de HORN.

Quando aplicada a um conjunto de cláusulas de HORN a resolução tem um comportamento procedural que simula um programa, apesar de não determinístico.

Para que se possa estabelecer algumas comparações entre a lógica de primeira ordem e a lógica de ordem superior com relação aos mecanismos utilizados na mecanização das teorias nelas envolvidas, descreve-se ainda alguns aspectos de sua implementação.

A visão prodedural da resolução é percebida de maneira mais clara quando se observa a forma de implicação do conjunto de axiomas. Considere $\sim B \vee \sim C$ uma fórmula objetivo, que se deseja provar. A escolha de uma fórmula para que se possa realizar um cancelamento do tipo L e $\sim L$ permite algumas opções. Em primeiro lugar, pode-se querer trabalhar com a fórmula $\sim B$ deixando-se a opção $\sim C$ para uma segunda tentativa, caso haja falha com ~B ou se queira mostrar a possibilidade de se provar a fórmula objetivo de mais de um modo.

Neste ponto o procedimento não é determinístico e algumas heurísticas são geralmente usadas para melhorar o processo de busca. O processo de busca consiste em escolher uma fórmula que apresente (ou possa apresentar) um literal complementar que permita o cancelamento. É possível que não se encontre tal fórmula, encontre-se uma, ou mais de uma fórmula seja candidata. Novamente haverá um não-determinismo caso haja mais de uma possibilidade de escolha.

A visão procedural consiste em imaginar que o literal da fórmula objetivo é uma chamada de procedimento, possivelmente com parâmetros, e que o antecedente da cláusula escolhida é a subrotina, com seus parâmetros formais. Há um "casamento" de parâmetros, realizado basicamente pela unificação, e a subrotina será executada, tomando-se sua parte resolvente, como novas chamadas a procedimentos. Desta forma a "passagem de parâmetros" pode ser vista como uma "passagem por nome". Observe o exemplo abaixo:

Base de dados :

1. $A \Leftarrow B \wedge C$
2. $A \Leftarrow C$
3. $B \Leftarrow D \wedge E$
4. $B \Leftarrow F$
5. $C \Leftarrow$
6. $C \Leftarrow F \wedge D$
7. $D \Leftarrow$
8. $F \Leftarrow$

Consulta ou fórmula objetivo :

$$\Leftarrow (A \wedge C) \text{ ou } \sim A \vee \sim C$$

A escolha do literal $\sim A$ para unificar em primeiro lugar, deixa ainda a opção $\sim C$ para uma oportunidade futura. Ao mesmo tempo permite que se escolha a fórmula 1 ou 2 para serem resolvidas. Se as fórmulas são unificáveis, na possível escolha da fórmula 1, a visão procedural aponta para a resolução de uma nova fórmula objetivo que é $B \wedge C$, com as substituições estabelecidas pela unificacão de A e $\sim A$.

Se ao final o procedimento falhar em conseguir a fórmula vazia, pode-se tentar a prova utilizando-se o literal $\sim C$, ou outro que tenha sido deixado para trás. Este mecanismo é chamado de *retroencadeamento*. De modo semelhante, $\sim C$ e outros literais que tenham sido deixados para trás podem ser usados para mostrar uma outra solução para a prova da fórmula $\sim A \vee \sim C$.

2.6 O Interpretador PROLOG

O PROLOG foi implementado pela primeira vez entre 1970 e 1972, como uma aplicação das idéias de programação lógica, por [Colm73]. Dado um programa lógico, conforme definido por [Ster86], o interpretador abstrato para tal programa é descrito na figura 2.1

O interpretador abstrato da figura 2.1 encobre detalhes de implementação que interferem no mecanismo de execução, na medida em que é definida uma ordem para a escolha quer das cláusulas dentro do programa, quer das cláusulas dentro da consulta G.

O PROLOG padrão segue a orientação de selecionar sempre a fórmula mais à esquerda da consulta e orientar a pesquisa por uma cláusula unificável no programa (base) sequencialmente, o mesmo ocorrendo no retroencadeamento.

Entrada: Um programa lógico P e uma consulta G

Saída: $G\theta$, se houve uma instância de G deduzida de P, ou **Falha** se ocorreu falha na dedução.

Algoritmo: Inicialize o resolvente como sendo G.

Enquanto o resolvente não for vazio escolha uma fórmula A do resolvente e uma cláusula
..... $A\prime \Leftarrow B_1, B_2, \cdots, B_n,\ n \geq 0$
de P, tal que A e $A\prime$ tenham θ como unificador. Se não existir tal cláusula, saia do sistema. Remova A e adicione
..... $B_1, B_2, \cdots, B_n$
ao resolvente. Aplique θ ao resolvente e a G.

Se o resolvente for vazio, imprima G. Caso contrário imprima **Falha**.

Figura 2.1: Interpretador abstrato para um programa de primeira ordem

Existem muitas implementações do PROLOG. Alterações de sintaxe, de facilidades e de procedimentos as diferenciam.

Em termos de implementação, maiores avanços foram conseguidos por Warren D. H. a partir de 1977, descritos em vários trabalhos até 1980 [Hogg84]. As características de suas implementações levaram ao estabelecimento de modelos que incluem métodos de indexação, compartilhamento de estruturas, compiladores parciais e outras particularidades que objetivam à otimização tanto do consumo de memória, quanto do tempo de processamento.

[Hogg84] apresenta um resumo geral das técnicas utilizadas na implementação de interpretadores PROLOG. Há basicamente a preocupação de controlar o ambiente de execução: o assinalamento de valores às variáveis e o progresso da computação, entendida como os passos executados para a busca das soluções. Isto é feito por meio de uma estrutura de pilha que armazena a história de toda a execução e possibilita a recuperação de ambientes anteriores para o mecanismo de retroencadeamento e para justificar e exibir os resultados de uma computação.

Em 1983, um artigo de D. H. Warren [Warr83] estabeleceu técnicas de implementação da linguagem , que passaram a ser chamadas de WAM (Warren Abstract Machine) e se tornaram padrão para implementação de PROLOG compilado. A "máquina" é composta de uma arquitetura de memória e um conjunto de instruções. A despeito de seu grande valor o trabalho original foi feito de modo pouco claro, com poucos detalhes e muito foi deixado de falar. Em maior ou menor grau, muitos dos métodos nele contidos foram aproveitados nas implementações existentes comercialmente.

Descrição desta máquina abstrata foi feita em forma de tutorial por Hassan Ait-Kaci, [Ait-90], em 1990 numa forma detalhada que conduz o leitor à sua implementação. A robustez da máquina reside no bom uso de uma estrutura de dados em forma de árvore para os termos e o estabelecimento de instruções compiladas para manipulá-las.

A pesquisa e implementações comerciais tornaram o PROLOG uma ferramenta bem divulgada e aceita para resolução de problemas que se ajustam à linguagem lógica. Estudos teóricos fundamentaram sua aplicação por meio de análises de correção e completude. Os choques originados destas análises ao se buscar a exequibilidade dos modelos geraram algumas "permissividades" que enfraquecem a robustez teórica das implementações em detrimento da eficiência. Um exemplo disto é a ausência do *teste de ocorrência* em alguns interpretadores e a presença do operador *cut*. Há que se lembrar ainda que a resolução SLD se aplica apenas a conjuntos de cláusulas de Horn e seu uso não permite deduzir negações.

Em termos de aproveitamento das suas potencialidades foi grande a variedade de aplicações mesmo em situações onde as características do problemas eram mais procedurais do que declarativas. Além da implementação de predicados chamados por [Ster86] de "extra-logicos" e que se referem a entradas e saídas, interface com o sistema operacional e predicados para acessar e manipular o programa, técnicas foram desenvolvidas para atender a manipulação de conjuntos e suas propriedades que pertencem a chamada segunda-ordem. Assim surgiram alguns predicados já conhecidos como o *apply, set-off, map-list*, etc.

Já em 1971, Andrews [Andr71] questionava a lógica de primeira ordem como "unfortu-

nate" uma vez que muitas sentenças matemáticas e de outras disciplinas seriam expressas mais simples e naturalmente por meio da lógica de ordem superior, que apresentava ainda vantagens de distinguir sintaticamente expressões que denotavam tipos diferentes de objetos.

Paralelas ao uso crescente do PROLOG, pesquisas foram conduzidas em direção à formalização e operacionalização de modelos agora dentro deste contexto.

Nos capítulos subsequentes (3 e 4) são apresentados estes dois aspectos da lógica de ordem superior, que buscam atender algumas das deficiências citadas.

Capítulo 3

Lógica de Ordem Superior

3.1 Introdução

A Lógica de Ordem Superior (LOS) possui uma linguagem que admite predicados e símbolos funcionais como variáveis. Desta forma, em LOS, é possível ter sentenças como:

HOMEM (x)
INTELIGENTE (x)

que são predicados para a variável x, e se fazer a consulta *p(x)* que significaria "*quais os predicados que estão associados à variável x.*"

Um exemplo da representação de conhecimento em lógica de ordem dois, pode ser o seguinte: suponha uma base de dados que contenha conhecimento sobre as qualidades de diversas pessoas. Suponha que as qualidades informadas para as pessoas sejam a *caridade, honestidade, lealdade e fé.* Como qualidades não desejáveis tem-se: a *desonestidade e a ganância.* A base de conhecimento conteria o seguinte conhecimento expresso em lógica de primeira ordem:

Leal(Paulo)
Leal(Luiz)
Honesto(Mário)
Ganancioso(José)
Fé(Maria)
Leal(Roberto)
Desonesto(Luiz)
Fé(Paulo)
Ganancioso(Pedro)
Honesto(Márcio)

Usando a linguagem da lógica de ordem superior, pode-se informar quais predicados são virtudes e quais são defeitos:

Virtude(Fé)

Virtude(Honesto)
Virtude(Leal)
Defeito(Desonesto)
Defeito(Ganancioso)

Observe-se que neste nível as relações usam os nomes de predicados do nível de primeira ordem, ou nível *objeto*, como argumentos. Uma possível consulta sobre esta base seria

Candidato(x) :- Virtude(f), Virtude(g), f(x), g(x), Honesto(x)

na qual se pretende encontrar na base de conhecimento um candidato x que seja *honesto* e que possua duas virtudes f e g.

Na lógica de ordem superior, o domínio D das funções e predicados é constituído por um conjunto de relações arbitrárias. Relembrando, na lógica de primeira ordem tem-se:

funções : $D^n \rightarrow D$ e
predicados: $D^n \rightarrow$ { T,F }

Na LOS, esse conjunto de relações arbitrárias possui como elementos de seu domínio:

- *átomos*, que são os elementos básicos da linguagem
- *funções*, com sintaxe "f x" onde a variável x pode por sua vez ser outra função ou um átomo. Dependendo do tipo de x a função pode ser um predicado.

Todos os elementos referenciáveis em sentenças *(fórmulas)* da LOS fazem parte deste domínio. Isto significa que é possível fazer consultas sobre esta base de conhecimentos, na qual qualquer elemento desse conjunto pode ser tratado como uma variável, incluindo *funções* e *predicados*. É esta ampliação do domínio das funções e predicados que diferencia a linguagem da Lógica de Ordem Superior daquela da Primeira Ordem.

Na LOS todos os elementos do alfabeto da linguagem estão associados a símbolos para *tipo*, uma vez que se deve explicitar a que *tipo* de relação cada elemento pertence.

Para formalizar a lógica de ordem superior usa-se princípios do λ-cálculo, com suas noções de aplicação e abstração. A eles é incorporada a noção de tipos, permitindo-se a distinção sintática explícita entre termos que se referem a diferentes tipos de objetos.

A finalidade principal da associação de tipos aos elementos do alfabeto é diferenciar átomos, predicados e funções. Na *resolução* é necessário fazer casamento ("matching") entre as variáveis de uma fórmula e os elementos da base de conhecimento. As etapas de *substituição* e de *redução* são efetuadas com base nas variáveis. Isto é feito impondo-se inicialmente a restrição semântica de igualdade de tipos das expressões a unificar.

No λ-cálculo tipado, há algumas condições a serem satisfeitas para que se possa aplicar uma expressao F a uma expressão A. É necessário que F e A sejam de tipos apropriados e então a aplicação $F(A)$ tem sentido. A aplicação $F(F)$ é sempre sem sentido, porque pelas regras de formação, um argumento não pode ter o mesmo tipo da função. Essencialmente o λ-cálculo usa uma expressão do tipo (λx.F) para designar a abstração de uma função F com parâmetro x e corpo F. (F_1F_2) é a aplicação da função F_1 ao argumento F_2, que

é uma notação usada pelo λ-cálculo, diferente da $F_1(F_2)$ usual. Assim (λu.v(xy)) denota uma função definida por f(u) = v(x(y)). O uso do operador λ é feito para descrever, dentro da gramática, a propriedade da λ-expressão e poder atribuí-la a alguma propriedade (predicado) de um nível superior [Carn58].

A existência de diferentes elementos, pertencentes a diferentes conjuntos, sobre os quais se quer descrever conhecimento, e, por sua vez, a possibilidade de se descrever conhecimento sobre conhecimentos anteriormente estabelecidos, de maneira recursiva, cria uma hierarquia entre os símbolos associados aos elementos do alfabeto. A pesquisa na base de conhecimento se dá gradualmente, passando-se do nível mais alto na hierarquia até o mais baixo, pesquisando os argumentos até que um átomo da base possa substituir um ou mais argumentos.

Deve-se lembrar que a aparente liberdade de uso de relações e conjuntos, na hierarquia de tipos, não é completa. Relações e conjuntos de qualquer nível da hierarquia são constituídos apenas de conjuntos e relações que já faziam parte da hierarquia quando se atinge aquele determinado nível.

O objetivo deste capítulo é fornecer o formalismo básico da tipificação e da formação da linguagem da lógica de ordem superior. A linguagem necessita da definição de um alfabeto e das regras de formação de fórmulas bem formadas. Isto é feito nas seções 3.3 e 3.4, onde procura-se ilustrar com exemplos a mecânica da formação de expressões. Na seção 3.5 oferece-se uma interpretação para as fórmulas bem formadas e na seção 3.6 são vistas e ilustradas as definições que permitem a conversão de fórmulas, possibilitando o manuseio e operacionalização da linguagem. Grande parte da teoria coberta neste capítulo pode ser encontrada, a menos de diferenças na forma de expressar, em livros de λ-cálculo [Bare85] e em alguns trabalhos que investigam o uso da linguagem da lógica de ordem superior como [Henk63],[Andr86], [Mill86] e [Nune89], dentre outros. Em especial usou-se a linguagem encontrada em [Nada90].

3.2 Símbolos para Tipos

A teoria dos tipos pretende eliminar os paradoxos da teoria dos conjuntos introduzindo uma hierarquia de tipos e impedindo a formação de certos tipos de conjuntos. Aspectos desta hierarquização, independentes da formalização que se possa dar-lhes, são abordados em [Hofs80], onde a interação entre seus diversos níveis é objeto de análise.

A seguir é apresentada a definição do universo (infinito) de tipos para símbolos.

Seja F um conjunto com pelo menos dois elementos e e t, que não são n-tuplas ordenadas, onde $n \geq 2$. O simbolo t tem significado especial e está sempre presente em F. Seja também um conjunto C de pares ordenados $\langle c, n \rangle$, onde c é um símbolo qualquer e n pertence aos naturais. Para cada c existe no máximo um par $\langle c, n \rangle$ tal que $\langle c, n \rangle$ pertença a C. Os conjuntos F e C são denominados *conjunto de tipos fundamentais* e *conjunto construtor de tipos* , respectivamente.

O conjunto dos símbolos para tipos é o menor conjunto T tal que:

i) F está contido em T.

ii) Se $\langle c, n \rangle \in$ C e $a_1 \cdots a_n \in$ T, então $(c\, a_1 \cdots a_n) \in$ T.

iii) Se $\alpha \in$ T então $(\alpha) \in$ T.

iv) Se α e $\beta \in$ T então $(\alpha \rightarrow \beta) \in$ T.

Exemplos[1] :
Considere F = { *e*, *t*, *int* } e C = $\{\langle list, 1 \rangle\}$ então os seguintes símbolos para tipos são válidos:

a) $(e \rightarrow t)$: de (i) e (iv);

b) (e) : de iii);

c) $(list\ int)$: de ii);

d) $((e \rightarrow t) \rightarrow (e))$: de i), ii) e iv);

e) $((list\ int) \rightarrow (int \rightarrow int))$.

Dados os conjuntos F e C é possível descrever o conjunto T e verificar se um símbolo para tipo pertence ou não a T.

Considere o conjunto $A = \{e_1, e_2, \cdots, e_n\}$. Considere ainda que os elementos de A estão associados ao símbolo para tipo α (α é um tipo qualquer). Então A está associado ao símbolo para tipo (α). Note que todos os elementos de A possuem o mesmo símbolo para tipo.

Na lógica de ordem superior os *predicados* possuem símbolo para tipo $(\beta \rightarrow t)$, onde t pertence ao conjunto F (fundamental) e representa o conjunto de expressões cujas interpretações são um mapeamento sobre o conjunto {verdade, falso}.

As funções possuem símbolo para tipo da forma $(\alpha \rightarrow \beta)$ onde α e β podem ser quaisquer tipos válidos no conjunto T definido a partir de F e C (isto é, resultado das regras anteriores).

Os elementos do domínio da função possuem símbolo para tipo α e os do contradomínio β. O predicado é um caso especial de função onde $\beta = t$.

A *ordem* de um símbolo para tipo fundamenta o estudo do nível de complexidade em que a fórmula está em relação aos elementos fundamentais. Ela fornece a distância hierárquica da fórmula aos elementos fundamentais. Isto é importante porque uma fórmula de *ordem* n tem o processo de resolução satisfeito por fórmulas nos níveis $n, n-1 \cdots$ até o nivel elementar. Na verdade cada expressão bem formada na linguagem da lógica de ordem superior possui um único tipo, que indica, além de outras coisas, sua posição dentro de uma hierarquia funcional.

As relações de *complexidade* entre fórmulas da LOS foram analisadas em [Nune89] onde se apresenta um conjunto de condições necessárias para a dedução de conhecimento de

[1]Outras notações encontradas na literatura para $(\alpha \rightarrow \beta)$ são $(\alpha\beta)$, $(\alpha)\beta$ ou $F_{\alpha\beta}$. A notação $\alpha_1, \cdots, \alpha_n; \beta$ é uma abreviação para $(\alpha_1 \rightarrow (\alpha_2 \rightarrow (\cdots \rightarrow \alpha_n(\rightarrow \beta))))$. Outras vezes encontra-se a notação $(\alpha_1 \cdots \alpha_n; \beta)$ para a regra ii de geração de tipos

uma base. Tais condições são estabelecidas a partir das relações de complexidade entre fórmulas.

A *ordem* de um tipo é a profundidade do aninhamento dos parênteses daquele tipo acrescida de 1. A *ordem* de uma linguagem para uma base de conhecimento é a *ordem* máxima de suas variáveis e constantes.

Dado o conjunto de tipos elementares F e o construtor C, o conjunto de tipos T é uma linguagem livre de contexto sobre $T \bigcup \{(,), \rightarrow\}$.

A hierarquia observada entre os diferentes símbolos que denotam *tipos* é transferida também para os *conceitos* ou *conhecimentos* correspondentes, nos quais são usados.

A *ordem* de um símbolo para tipo é definida recursivamente como se segue. Dados α e β, a *ordem* do tipo α, indicada por *ordem*[α], tem a seguinte avaliação:

i) Se α pertence a F, então ordem[α] = 0. A ordem de um tipo fundamental é zero.

ii) Se $\alpha = (C\ a_1 \cdots a_n)$, tal que $\langle c, n \rangle \in C$, $a_i \in T$, para todo i, então ordem[α]= k+1 onde k = máx{ordem[a_i]}, $1 \leq i \leq n$.

iii) Se $\alpha \in T$ e ordem[α]= k, então ordem[(α)]= k + 1

iv) Se o tipo for $\gamma = (\alpha \rightarrow \beta)$, então ordem[$\gamma$]= k + 1 onde k = max{ordem[α], ordem[β]}

Uma hierarquia de tipos consiste da coleção de todos as relações tipadas sobre um conjunto de símbolos tipados D.

Uma relação tipada baseada em um conjunto D não vazio é uma relação que tem seu tipo definido recursivamente como se segue. Seja α pertencente a F o símbolo para tipo associado aos elementos de D.

i) Se x pertence a D, então x é uma relação que está associada ao símbolo para tipo α.

ii) Se $A = \{x_1, x_2, \cdots, x_n\}$, tal que $x_i \in D, i = 1, \cdots, n$, então A é uma relação a qual está associado o símbolo para tipo (α).

iii) Se B é uma lista, árvore, ou outra estrutura formada por elementos pertencentes a D, então B é uma relação que está associada ao símbolo para tipo (c α), em que "c" é um identificador para B. Isto é, "c" identifica se B é uma lista, uma àrvore, etc.

iv) Uma relação R associada ao símbolo para tipo $(\alpha_1 \rightarrow \alpha_2)$ é um conjunto R de pares ordenados tais que: $\langle r_1, r_2 \rangle$ pertence a R se e somente se r_i é uma relação associada ao símbolo para tipo α_i, $1 \leq i \leq 2$.

3.3 Alfabeto da Lógica de Ordem Superior

Uma vez estabelecido um universo para a definição de símbolos para tipos, deve-se estabelecer o alfabeto para que possa usar adequadameente a linguagem para expressar o conhecimento. A cada constante ou variável está associado um símbolo para tipo. Da mesma forma as constantes lógicas e quantificadores são associadas a tipos.

As seguintes classes de símbolos definem o alfabeto do sistema lógico.

i) Símbolos próprios.

(a) Para cada tipo $\alpha \in$ T existe um conjunto enumerável de variáveis do tipo α.

(b) Para cada tipo $\alpha \in$ T também existe um conjunto enumerável de constantes do tipo α, disjunto do conjunto de símbolos das variáveis.

ii) Símbolos impróprios. Como símbolos impróprios tem-se os parênteses,) e (, e o símbolo λ.

Pertencentes ao conjunto *i.b)* acima existem já algumas constantes lógicas:

Constantes	Tipo
$\wedge$	$(t \rightarrow t \rightarrow t)$
$\vee$	$(t \rightarrow t \rightarrow t)$
$\Rightarrow$	$(t \rightarrow t \rightarrow t)$
$\sim$	$(t \rightarrow t)$
T	t
$\exists_\alpha$	$((\alpha \rightarrow t) \rightarrow t)$
$\forall_\alpha$	$((\alpha \rightarrow t) \rightarrow t)$

Observe-se que os símbolos $\exists$ e $\forall$ exigem um predicado do tipo $(\alpha \rightarrow t)$ como argumento.

3.4 Fórmulas Bem Formadas

As fórmulas bem formadas da LOS são constituidas de átomos, aplicações e abstrações. O conjunto das fórmulas do tipo α $\{F_\alpha, \alpha \in T\}$, indicado tambem por $FORM_\alpha$, é definido indutivamente pelas seguintes regras:

i) *Elementos básicos*: Variáveis ou constantes do tipo α são fórmulas do tipo α

ii) *Abstração*: Seja x_α uma variável e H_β uma fórmula, onde os índices estão indicando os tipos, então $(\lambda x_\alpha H_\beta)$ é uma fórmula do tipo $(\alpha \rightarrow \beta)$

iii) *Aplicação*: Sejam as fórmulas $F_{\alpha\to\beta}$ e G_α, então $(F_{(\alpha\to\beta)}G_\alpha)$ é fórmula do tipo β.

Intuitivamente, uma *abstração* significa extrair de uma fórmula a propriedade ligada a variável que está sendo abstraida. Verbalmente, $(\lambda x_\alpha H_\beta)$ a "propriedade de x tal que H" ou ainda, " a classe dos elementos x tal que H". Trata-se de uma generalização. A *aplicação*, por sua vez, restringe a fórmula pois corresponde à eleição de um elemento.

O conjunto de fórmulas pode então ser visto como o *conjunto potência* de todas as variáveis e constantes (átomos), fechado sob aplicação e abstração. A relação *subfórmula de* é definida como sendo o fecho transitivo e reflexivo de :

F é subfórmula de F
F e G são subfórmulas de (FG)
F é subfórmula de $\lambda x\ F$

Diz-se no terceiro caso, que a abstração é ligada pela variável x. Se a variável não liga qualquer abstração de alguma subfórmula de F diz-se que ela é livre em F.

Neste estágio da formulação das sentenças de ordem superior é necessário observar com maiores detalhes a formação das fórmulas e sua interpretação. Alguns exemplos de fórmulas são dados a seguir sem a preocupação de interpretá-las:

Exemplo-3.4.1 Considere a seguinte fórmula de ordem superior:

(igual x 4)

em que x é uma variável, 4 e igual são constantes. Cada variável ou constante tem seu tipo apropriado: x é do tipo *int*, 4 é do tipo *int* e igual é do tipo $(int \to int \to t)$. Um elemento do conjunto potência de {4, x, igual} é $igual\ x\ 4$. Na verdade ele foi obtido ao se tomar a fórmula (constante) *igual*, acrescentar a ela por meio de uma aplicação, a fórmula x, pelo que se obtem $(igual\ x)$. Em seguida, por meio de outra aplicação obtém-se $((igual\ x)\ 4)$. Partiu-se de uma fórmula do tipo $(int \to int \to t)$ e obteve-se (observe a regra acima) uma fórmula do tipo t. O leitor deve observar que usa-se a notação abreviada para tipos: de $(\alpha_1 \to (\alpha_2 \to \cdots (\alpha_n \to \beta)\cdots))$ para $(\alpha_1 \to \alpha_2 \to \cdots \alpha_n \to \beta)$. Da mesma forma simplifica-se $(\cdots((F_1G_1)G_2)\cdots G_n)$ para $F_1G_1G_2\cdots G_n$. Portanto vale a associatividade à direita para simbolos de tipos. Alem disto, a menos que altere a interpretação, os parênteses podem ser omitidos. Para uma fórmula, vale a associatividade à esquerda.

Exemplo-3.4.2 Considere ainda a fórmula do exemplo-3.4.1. Sabe-se que seu tipo é t. Com uma abstração de x obtém-se

λx (igual x 4)

conforme regra ii) de formação de fórmulas. Esta fórmula, ainda conforme regra de formação, tem o tipo $(tipo_x \to t)$. Se x tiver tipo *int* então a nova fórmula formada tem tipo $(int \to t)$. Observe que a abstração está ligada por x. No exemplo 1, x estava livre.

Exemplo-3.4.3 Sejam F e G duas fórmulas definidas usando os conceitos de formação de fórmulas, conforme exemplos 3.4.1 e 3.4.2 acima: G = (Pai y José) e F = λy (Primo y Maria). Mostra-se a seguir como se forma indutivamente a fórmula

$$\wedge \text{ (Pai y José) } (\lambda y \text{ (Primo y Maria))}$$

Esta fórmula é uma representação prefixada da fórmula G $\wedge$ F. Neste, como nos casos anteriores, tem-se variáveis { y } e constantes {Pai, Primo, Maria, José e $\wedge$ }. Note-se que $\wedge$ é uma *constante lógica* predefinida de tipo $(t \rightarrow t \rightarrow t)$. As constantes *José* e *Maria* e a variável *y* têm tipo *pes*. As constantes *Primo* e *Pai* têm tipo $(pes \rightarrow pes \rightarrow t)$. O tipo da subfórmula G é *t*. Logo pode-se aplicá-la a $\wedge$ obtendo ($\wedge$ (Pai y José)). Uma nova aplicação à fórmula assim formada que é do tipo $(t \rightarrow t)$, exige, conforme regra (iii) de formação, uma fórmula do tipo *t*. A fórmula F tem tipo $(pes \rightarrow t)$. Portanto a fórmula acima não está correta. Algum conhecimento poderia ser representado por

$$\lambda y \wedge \text{ (Pai x José) (Primo y Maria)}$$

que está correta. É interessante observar que x é uma variável livre nesta fórmula.

Exemplo-3.4.4 Suponha que a fórmula H tenha tipo *int* e seja definida como (mais x 2). A constante *mais* tem tipo $(int \rightarrow int \rightarrow int)$. A fórmula H está bem formada. Pode-se então abstrair H de x fazendo

$$\lambda x \text{ H} = \lambda x \text{ (mais x 2)}$$

Pode-se aplicar quaisquer dos quantificadores a qualquer fórmula contendo variáveis. Basta que sejam obedecidas as regras de formação de fórmulas. Assim, supondo $tipo_x = int$ como a fórmula acima tem tipo $(int \rightarrow int)$ e o tipo de $\exists$ e $\forall$ é $((\alpha \rightarrow t) \rightarrow t)$ ela não permite a aplicação de qualquer dos quantificadores. Ficam então incorretas as duas fórmulas:

$$\exists x(\lambda x \text{ (mais x 2))}$$
$$\forall x(\lambda x \text{ (mais x 2))}$$

Se, no entanto substituir-se a constante mais por maior, com tipo $(int \rightarrow int \rightarrow t)$ tem-se as duas seguintes fórmulas corretas:

$$\exists x(\lambda x \text{ (maior x 2))}$$
$$\forall x(\lambda x \text{ (maior x 2))}$$

Neste caso a relação tornou-se predicativa e tomando-se o dominio dos naturais, por exemplo, uma aplicação da abstração a uma constante permite uma avaliação dentro do conjunto dos valores verdade. A interpretação e a aplicação são discutidas nas seções seguintes.

Exemplo-3.4.5 O conjunto construtor C contém o elemento $\langle cj, 1 \rangle$. Desta forma pode-se ter a fórmula $(cachorro\ gato)_{(cj\ animal)}$, que é uma constante. A partir dela forma-se a fórmula

(Domésticos (cachorro gato))

por exemplo, em que Domesticos é uma constante do tipo $((cj\ animal) \rightarrow t)$. O conjunto C contem elementos que são construtores de estruturas.

3.5 Interpretação

As fórmulas do tipo $(\alpha \rightarrow \beta)$ são interpretadas como funções cujo domínio é formado por objetos do tipo α e o contradomínio por objetos do tipo β. Já as fórmulas $F_{(\alpha \rightarrow \beta)}G$ são interpretadas como a aplicação de um símbolo funcional a uma fórmula.

A análise formal da interpretação das fórmulas deste sistema lógico é dada por [Nune89]. onde as fórmulas tomam a semântica indicada acima, ou seja, respectivamente de uma função (abstração) e de uma aplicação funcional (aplicação). A seguir são dados alguns exemplos que ilustram tal semântica.

Exemplo-3.5.1 A abstração da fórmula (igual x 4) por x resulta em λx (igual x 4). Sua semântica torna-se aquela de uma função em x. Observe que seu domínio é aquele de x (de tipo *int*) e seu contradominio é de valores de tipo t, o que está dito no tipo da fórmula (função) assim formada que é $int \rightarrow t$. Já a fórmula obtida da aplicação de λ x (igual x 4) a 3 resulta em

(λx (igual x 4) 3)

que é o mesmo que aplicar uma f(x) a 3. Neste ponto deve-se observar que a aplicação resulta no valor Falso, mas para isso deve-se considerar a transformação de fórmulas que será vista adiante. Tem-se então, duas formas de representar a fórmula. Como no cálculo real se $f(x) = x^2$ então pode-se representar aplicação de f(x) a 3 como f(3) ou como $3^2 = 9$.

Exemplo-3.5.2 Uma estrutura pode conter em seu interior variáveis ligadas ou não. Suponha que a estrutura seja uma lista, abstraida por x da seguinte forma

λx (append 3 (x 8 6) (3 x 8 6))

A interpretação é a de que pode-se acrescentar um elemento a uma lista contendo x que o resultado será aquele expresso na última lista. Toda a fórmula poderia ser aplicada a 6, resultando em

(λx (append 3 (x 8 6)(3 x 8 6))6)

cujo resultado depende de uma λ-conversão, para transformar a fórmula para uma forma adequada.

Exemplo-3.5.3 Observa-se que não há um extremo rigor na colocação dos parênteses. Outras liberdades notacionais são assumidas. Quando há mais de uma variável na abstração a forma correta seria algo do tipo $(\lambda x(\lambda y(\cdots$ que é abreviada como $\lambda xy\cdots$. De forma semelhante $(\lambda x_1(\lambda x_2\cdots$ tem sido representado por $\lambda\bar{x}$. Se a "função" $(\lambda x$((mais x) 2)) sofresse a aplicação de uma outra função $(\lambda y$ ((execute y) 3)), a aplicação resultaria em

$$((\lambda y\ ((\text{execute y})\ 3)\ \lambda x\ (\text{mais x}\ 2)))$$

Observe que a constante *execute* aceita como parâmetro y uma função do tipo $(int \rightarrow int)$. Seu tipo é $((int \rightarrow int) \rightarrow int \rightarrow int)$. O tipo da fórmula acima é *int*, admitindo que 2, 3, x tenham tipo *int*.

3.6 λ–Conversões

Nos exemplos acima nota-se que a operação de aplicação tem levado à fórmulas não resolvidas. Chama-se de fórmulas não resolvidas quando estas são o resultado de uma aplicação de uma abstração a uma fórmula de tipo adequado. Tem-se uma equivalência à situação do cálculo, quando coloca-se f(2) e não o valor desta aplicação. Outras situações apresentam fórmulas que pos suem equivalência, e a conversão de uma fórmula em outra se chama λ- conversão. A operação de λ-conversão elimina redundâncias na representação de conhecimento e permite determinar uma representante para um conjunto de fórmulas equivalentes, que é única a menos de renomeação de variáveis, e para todo conjunto de fórmulas equivalentes é possível determiná-la. Para se formalizar a operação de λ-conversão são necessários os conceitos de *substituição* e de *liberdade de uma variável para substituição.*

Seja G uma fórmula cujo tipo é igual ao tipo de x e F uma fórmula arbitrária. A notação

$$S_G^x F$$

é usada para representar o resultado da substituição de todas as ocorrências livres de x em F por G. Esta operação tem a seguinte definição recursiva:

i) Se F é
   a) uma variável igual a x então $S_G^x F$ = G
   b) uma constante ou variável diferente de x, então $S_G^x F$ = F

ii) Se F é uma abstração de H por y, ou seja λ y H
   a) Se y = x então $S_G^x F$ = F
   b) Se y $\neq$ x então $S_G^x F = \lambda$ y $(S_G^x H)$

iii) Se F = (BH) então $S_G^x F = (S_G^x$ B$)(S_G^x$ H) onde B, H e F são fórmulas.

Naturalmente as substituições são possíveis apenas quando há compatibilidade de tipos entre x e G. Considere a seguinte fórmula

$$\text{F} = (\lambda z\ (\lambda x\ (\ \text{maior z x})))$$

Suponha G = (mais y 5) do tipo *int*. Na fórmula F o tipo de x e de z é *int*. No entanto, nem x, nem z podem ser objeto de substituição por não estarem livres em F. Suponha agora que se tem a constante mais do tipo (*int* $\rightarrow$ *int* $\rightarrow$ *int*) na fórmula

$$\text{F} = (\lambda z\ (\lambda x\ (\ \text{maior z (mais x w)})))$$

Neste caso a variável w é passível de substituição por um termo r do tipo *int*. Assim, tomando-se G como expresso acima tem-se

$$S_G^w F = (\lambda z(\lambda x\ (\ \text{maior z (mais x (mais y 5)))))).$$

Deve-se notar que a variável y que era livre em G continua livre em $S_G^x F$.

A substituição pode, à vezes, tornar ligada uma variável que era livre. [Bare85] analisa os efeitos de substituições em variáveis livres e ligadas e contrasta dois metodos de tratá-los: o primeiro é atribuido a [Curr58], e usa o conceito de variável livre que será fornecido abaixo. O segundo, de De Bruijn [DeBr72], transforma cada termo em uma estrutura de árvore em que o nome das variáveis não é requerido. Com a eliminação dos nomes das variáveis por meio de termos chamados *termos sem nomes*, o problema de *confusão de variáveis* inexiste. O autor aponta esta representação como mais adequada para o tratamento computacional.

Para contornar este fato e tornar a substituição invariante quanto à ligação das variáveis, considera-se a seguinte definição:

G é *livre para x em F*, sendo x e G do mesmo tipo, se e somente se F não posssui uma subfórmula (λy C) tal que x seja livre no escopo da abstração (λy C) e a variável y ocorra livre em G.

As operações de λ-conversão são usadas para se obter fórmulas equivalentes entre si. Citou-se anteriormente um caso de equivalência. Outro caso ocorre quando duas fórmulas são iguais a menos de uma renomeação de variáveis.

Tem-se cinco operações de conversão: α- conversão, β-redução, β-expansão, η-redução e η-expansão:

- α-conversão : é a substituição de (λx F) por ($\lambda y\ (S_y^x F)$) ou a substituição de (AB) por ($S_y^x F$ A)($S_y^x F$ B). Esta operação tem apenas o efeito de renomear variáveis.
- β-redução: é a substituição de ((λx F) G) por $S_G^x F$. De modo informal, trata-se de substituir x por G na fórmula F.
- β-expansão: é a operação inversa à operação realizada na β-redução. O retorno da forma reduzida para a forma anterior à redução é o resultado desta operação.
- η-redução : é a substituição da fórmula ($\lambda x(F\ x)$) por F , se F tem tipo $\alpha \rightarrow \beta$ e x tem tipo α. Observa-se que há manutenção de tipos para as fórmulas. Adota-se portanto a equivalência entre λx *(impar x)* e *impar*, por exemplo, onde ambas têm tipo *int* $\rightarrow$ *t*.

- η-expansão: é o inverso de η-redução. Isto é substitui-se uma fórmula F por ($\lambda x(Fx)$) desde que haja compatibilidade de tipos entre F e x.

Em todas as operações de λ-conversão deve-se observar a liberdade de substituição, conforme definida acima. As duas últimas conversões (η-redução e η-expansão) caracterizam o axioma da extensionalidade dado em [Andr86], que assegura que se dois predicados têm a mesma *extensão*, seus significados são os mesmos.

Redex é um sufixo para fórmulas passíveis de redução. Assim tem-se fórmulas η-redex e β-redex. Alguns exemplos de conversões são dados a seguir:

Exemplo-3.6.1 Seja F definido como ((λy (maior y 3)) 4). Então F é uma β-redex que pode ser convertida em (maior 4 3). Observa-se que há uma manutenção de tipos nas duas fórmulas. Pois se, por exemplo, maior tem tipo ($int \rightarrow int \rightarrow t$), y e 3 têm tipo *int*, λy(maior y 3) tem tipo $(int \rightarrow t)$ e F terá tipo *t*. (maior 4 3) terá tipo *t*.

Exemplo-3.6.2 A α-redução é apenas a renomeação de variáveis. Assume-se como equivalentes as fórmulas cuja diferença reside apenas nos nomes das variáveis. Se F é definida como ($\lambda x(\lambda z$ (gosta x z))) então (λ y (λ r (gosta y r))) é uma α-conversão de F.

Exemplo-3.6.3 Seja F definida como 16, de tipo int. Não se pode fazer a aplicação de 16 a qualquer fórmula devido ao tipo de F. Suponha, no entanto que F seja (λ y (Pai y João)). Deseja-se aplicar a η-redução a F. Ainda neste caso F não se conforma com a definição de η-redução. Caso a fórmula seja ((λx (Pai y João)) x), pode-se substituí-la por (Pai y João) usando a η-redução.

Exemplo-3.6.4 Seja Pai uma constante do tipo $(p \rightarrow p \rightarrow t)$. As seguintes operações de aplicação, abstração e redução sobre ela estão válidas, desde que A, B e x sejam do tipo p

$Pai_{(p\rightarrow p\rightarrow t)}$	$Pai_{(p\rightarrow p\rightarrow t)}$	$Pai_{(p\rightarrow p\rightarrow t)}$
$(Pai\ x)_{(p\rightarrow t)}$	$(Pai\ B)_{(p\rightarrow t)}$	$(\lambda x Pai)_{(p\rightarrow p\rightarrow p\rightarrow t)}$
$((Pai\ x)A)_t$	$((Pai\ B)x)_t$	$(\lambda y(\lambda x Pai))_{(p\rightarrow p\rightarrow p\rightarrow p\rightarrow t)}$
$((\lambda x(Pai\ x)A))_{(p\rightarrow t)}$	$((\lambda x((Pai\ B)x))_{(p\rightarrow t)}$	$((\lambda y(\lambda x Pai))B)_{(p\rightarrow p\rightarrow p\rightarrow t)}$
$((\lambda x((Pai\ x)A))B)_{(t)}$	$((\lambda x((Pai\ B)x))A)_{(t)}$	$(\lambda x Pai)_{(p\rightarrow p\rightarrow p\rightarrow t)}$
$((Pai\ B)A)_{(t)}$	$((Pai\ B)A)_{(t)}$	$((\lambda x Pai)A)_{(p\rightarrow p\rightarrow t)}$

No último exemplo da secção anterior o parâmetro execute indica que se deseja executar a função y aplicando-se a ela o valor 3. No entanto, esta aplicação é pertencente à metalinguagem que interpreta a fórmula acima. A aplicação da função

(λy ((execute y) 3)

à formula

$(\lambda x$ (mais x 2))

resultou em

$(\lambda y$ ((execute y) 3) λx (mais x 2))

que após β-redução será

(((execute (λx ((mais x 2))) 3)

que não mais permite β-redução. Se fosse usada a notação simplificada

λy (execute y 3) λx(mais x 2)

a β-redução seria indicada por

(execute λx(mais x 2) 3).

À primeira vista a fórmula contém uma sub-fórmula β-redex que é λx(mais x 2) 3. No entanto, não se pode efetuar a redução. Ela mudaria toda a estrutura dos argumentos de execute. O tipo de execute requer um argumento funcional e a redução, alem de tornar o primeiro argumento uma constante, eliminaria o segundo argumento de execute. A diferenciação entre o que é um argumento e o que é uma aplicação sujeita a redução é vista melhor abaixo.

Exemplo-3.6.5.1 Sejam $Par_{(int \to t)}$ e $Produto_{(int \to int \to int)}$ duas fórmulas. Após alguns desenvolvimentos chega-se , por exemplo, a λx (Par x) cujo tipo é $(int \to t)$ e λy (Produto 2 y) 3, cujo tipo é *int*. Logo pode-se aplicar Par a Produto obtendo-se a seguinte fórmula β-reduzida: (Par λy (produto 2 y) 3). Neste caso, há uma sub-fórmula passível de redução. Está estabelecida uma fonte de confusão.

Uma fórmula pode ser convertida para outra por meio de uma sequência de λ-conversões. Diz-se que uma fórmula está na forma λ-normal se ela tem a forma :

$$\lambda x_1 \cdots \lambda x_n (H\ t_1 \cdots t_m)$$

onde H é uma constante ou variável, $n, m \geq 0$, isto é o vetor de abstrações pode ser vazio e/ou o conjunto de argumentos é vazio. H é denominado de **cabeça** da forma normal e os termos t_i são os argumentos que por sua vez também devem ter a forma λ-normal. As formas λ-normais são obtidas por uma série de reduções, até que nada mais existe para reduzir. As formas λ-normais têm então uma das formas abaixo:

i) F = $(H\ t_1 \cdots t_m)$

ii) F = $\lambda x_1 \cdots \lambda x_n (H\ t_1 \cdots t_m)$

iii) F = H

iv) F = $\lambda x_1 \cdots \lambda x_n$H

A fórmula $\lambda x_1 \cdots \lambda x_n(Ht_1 \cdots t_m)$ representa $(\lambda x_1(\cdots(\lambda x_n(Ht_1)\cdots)t_m)$ e nela os parênteses são omitidos, conforme já foi mostrado no exemplo 3.5.3. Demonstra-se ([Andr71], [Fort83]) a unicidade da forma λ-normal de uma fórmula.

3.7 Resumo

A lógica de ordem superior foi apresentada como uma forma de representação de conhecimento. Além da gramática comum à primeira ordem, nela são usados os recursos de tipificação e suas expressões usam o operador λ como forma de abstração. As operações do λ-cálculo para conversões de fórmulas são incorporadas para transformação de fórmulas equivalentes.

Estabelecidas as regras de tipificação e de formação de sentenças um sistema formal pode ser estabelecido fornecendo-se alguns axiomas e tomando-se algumas operações como regra de inferência. [Nada87] fornece o seguinte sistema formal:

- Axiomas:

  1. T

  2. $p \wedge q \Rightarrow p$

  3. $p \wedge q \Rightarrow q \wedge p$

  4. $p \Rightarrow (q \Rightarrow (p \wedge q))$

  5. $p \vee p \Rightarrow p$

  6. $p \Rightarrow p \vee q$

  7. $p \vee q \Rightarrow q \vee q$

  8. $p \Rightarrow q \Rightarrow [r \vee p \Rightarrow r \vee q]$

  9. $fx \Rightarrow \exists x.fx$ (dependendo de escolha do tipo α de f.)

- Regras de inferência:

  Substituição, λ-conversões, *Modus Ponens* e Regra do Existencial.

[Andr71] também fornece um sistema formal um pouco diferente do acima exposto.

A restrição das sentenças deste sistema a um subconjunto de sentenças que generalizam as cláusulas definidas de primeira ordem permite conservar as propriedades computacionais da primeira ordem. É este subconjunto que apresenta interesse para a definição de um interpretador para a linguagem, e será objeto de análise do próximo capítulo.

Capítulo 4

Unificação e P-derivação

4.1 Introdução

Como a Lógica de Ordem Superior fornece um formalismo mais natural e com mais facilidades para a expressão do conhecimento, há um acentuado interesse em elaborar uma teoria que permita a construção de provas em bases de conhecimento que usem este tipo de representação. Definida a tipificação, as fórmulas bem formadas e as regras de conversão de fórmulas daquele sistema, explicita-se neste capítulo um subconjunto de fórmulas que permite a construção de provadores .

Algumas pesquisas têm mostrado que a LOS possui várias propriedades similares àquelas da primeira ordem que tornam possível sua mecanização em termos de busca de provas. Assim, existe um teorema de Herbrand equivalente para a LOS, a unificação foi desenvolvida e algumas implementações de provadores de LOS foram propostas e desenvolvidas, embora alguns problemas ainda sejam levantados e por isso não se tem informação de sistemas que executem a tarefa sem restrições ([Nada90],[Nada89]).

A limitação destes sistemas está no compromisso entre ser completo e ao mesmo tempo atrativo do ponto de vista computacional. Quanto mais se exige da segunda qualidade, mais se compromete a primeira. O problema se liga principalmente à existência na LOS de predicados variáveis que podem ocorrer extensionalmente em fórmulas desta lógica. Uma variável predicativa ocorre extensionalmente numa fórmula se ela for o símbolo mais à esquerda da fórmula L em L, $\sim$L ou ainda quando L é extensional e está em $\vee$ L L, ou em $\wedge$ L L. Nestes casos a substituição da variável predicativa por qualquer termo pode produzir uma fórmula com estruturas proposicional e quantificacional diferentes.

Exemplo 4.1.1 Seja F definida como (Virtude Honesto) $\Rightarrow$ (y Honesto). A variável y ocorre extensionalmente em F. Como a gramática não aceita o símbolo $\Rightarrow$ a sentença será ($\vee$ $\sim$(Virtude Honesto) (y Honesto)). A constante Virtude, bem como a variável y têm tipo $((pes \rightarrow t) \rightarrow t)$. Portanto y pode ser substituido por qualquer fórmula daquele tipo. Por exemplo[1]

[1]Interpretação: z é uma virtude tal que para todo x que possue aquela virtude z, este x tambem é Leal.

$$\lambda z\ (\wedge\ (\text{Virtude z})\ (\forall x\ (\text{z x}) \Rightarrow (\text{Leal x}))$$

cuja substituição em y, seguida de uma β-redução resulta em:

$$(\text{Virtude Honesto}) \Rightarrow \wedge\ (\text{Virtude Honesto})$$
$$(\forall x\ (\text{Honesto x}) \Rightarrow (\ \text{Leal x})).$$

[Mill85] analisa a situação de variáveis predicativas intensionais quando estas aparecem dentro dos termos de uma fórmula objetivo e extensionais, quando são a cabeça destas fórmulas. Tais predicados podem ser determinados por meio da unificação, como funções, quando aparecem intensionalmente . Quando são extensionais estes predicados são essencialmente "executados".

O método tradicional de construir uma prova para uma fórmula em lógica com quantificação pode, no sentido geral, ser imaginado como sendo a substituição de variáveis existencialmente quantificadas por termos e a posterior investigação para ver se a fórmula bem formada é uma tautologia. Na lógica de primeira ordem isto ocorre sem problemas pois a substituição de tais variáveis não interfere na estrutura proposicional das sentenças. Isto não é verdade na lógica de ordem superior, onde variáveis predicativas podem ocorrer extensionalmente.

A substituição aplicada a variáveis assim quantificadas pode alterar profundamente a estrutura das fórmulas. Como ficará então o processo de construção de provas onde o procedimento pode alterar a estrutura das proposições que dão suporte à prova ? É necessário então escolher a forma correta de fazer a substituição, ou a maneira correta de alterar a estrutura proposicional. Nenhum método tem mostrado completude para desenvolver esta tarefa. Em tais casos, a construção da prova envolve encontrar a maneira "correta" de mudar a estrutura proposicional. Tem-se indicações de técnicas úteis, como em [Bled79]. Outros provadores não fazem a procura para tais substituições [Andr86], enquanto outros as executam exaustiva e indiretamente como em [Huet73].

Embora este problema tenha sua relevância, a restrição da base de conhecimento a um conjunto de fórmulas com estrutura pré-fixada permite que um procedimento completo de prova seja descrito. Ele usa a unificação, como na lógica de primeira ordem e é implementado por uma rotina de aplicação. Consegue-se assim uma grande motivação para trabalhar na LOS : segue-se o modelo de mecanização da lógica de primeira ordem, e consegue-se ao mesmo tempo uma representação mais livre com poder mais abrangente de expressão do conhecimento, permitindo inclusive ter funções e predicados como variáveis.

A argumentação anterior introduz a expectativa de que se pode trabalhar com uma base de conhecimento sem envolver alguns tipos de fórmulas e seguir com as vantagens da LOS na representação. De forma resumida, as formas extensionais causam problema quando são incluidas como consulta. Nestes casos, as possíveis soluções seriam genéricas a ponto de terem seu interesse diminuido ou até mesmo inútil. Um tratamento especial é dispensado a elas quando isto ocorre, e é mostrado na seção 4.4 deste capítulo. Espera-se

com isso suplantar as dificuldades advindas da substituição de algumas variáveis extensionais. Além disto, também da argumentação anterior, espera-se que algum tipo de atitude não convencional esteja para ser assumida na unificação .

Neste capítulo é definido o subconjunto de fórmulas chamado de fórmulas definidas de ordem superior. Sua definição e exemplos estão contidas na seção 4.2. Com uma base de conhecimento assim formada o procedimento de derivação chama-se de P-derivação e conserva grande parte das características da resolução das cláusulas definidas de primeira ordem. Na seção 4.3 são analisadas algumas destas propriedades e descrevem-se os procedimentos principais da programação lógica de ordem superior, notadamente a unificação (subseção 4.3.1) e a P-derivação (subseção 4.3.2).

A unificação pode ser vista como o núcleo central do processo de derivação, permitindo deixá-lo ou não prosseguir, abrindo-se um leque de possíveis opções de derivação, de acordo com o número de unificadores encontrados. Para a P-derivação aborda-se sobremaneira a redução da consulta e a pesquisa de regras e fatos na base de conhecimento.

4.2 Fórmulas Definidas de Ordem Superior

Restringe-se o estudo a um subconjunto de fórmulas $\mathcal{T}$, chamado de conjunto de fórmulas definidas. Para defini-lo é preciso do conceito de fórmula positiva. Estas são fórmulas em que não ocorre o simbolo de negação ($\sim$).

A classe de fórmulas positivas (FP) é o menor conjunto de fórmulas de $\mathcal{T}$ que satisfaz as seguintes propriedades :

a) Cada variável e constante, exceto $\sim$, estão em FP.

b) Se F, G e H pertencem FP então λx F e (G H) também pertencem.

Cada fórmula positiva é tipificada e segue as regras de tipificação.

Universo Positivo de Herbrand (H^+) é a coleção de todas as fórmulas positivas λ-normais, isto é, expressas em suas formas normalizadas, como foi definido no capítulo 3. Base de Herbrand (HB) é a coleção de todas as fórmulas fechadas (instanciadas) em H^+ .

As definições de Universo e Base seguem os mesmos conceitos das definições equivalentes na primeira ordem.

Então a Base de Herbrand (HB) é o conjunto de todas as fórmulas que estão instanciadas por qualquer das constantes da linguagem.

Fórmula objetivo é uma fórmula de tipo t em H^+. Ela é uma proposição sem o símbolo $\sim$. Intuitivamente, fórmulas objetivo são sentenças que possuem um valor verdade. Podem, inclusive, ser compostas com as constantes lógicas. Veja exemplo 4.2.1.

Átomo positivo é uma fórmula objetivo positiva (com tipo t, portanto com valor verdade,) atômica e átomo **rígido** positivo é uma fórmula objetivo que tem um parâmetro como sua cabeça. Assim uma proposição sem $\sim$ é um átomo positivo e é ou $\top$ (a constante $\top$) ou uma fórmula da forma (A $t_1 \cdots t_n$) onde os t_i representam os "n" argumentos

aplicados a A, e A é do tipo $\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow t$ e cada t_i é uma fórmula λ-normal do tipo α_i. A proposição em que A é um parâmetro é *rígida.*

Indutivamente as fórmulas objetivo podem ser definidas como :

i) um átomo é uma fórmula objetivo (átomo é uma proposição cujo símbolo mais à esquerda é uma variável ou um parâmetro).

ii) (A $\vee$ B) e (A $\wedge$ B) são fórmulas objetivo se A e B o forem.

iii) $\exists x \lambda x$ P é uma fórmula objetivo se P pertence ao universo Positivo de Herbrand (H^+) e tem tipo $(\alpha \rightarrow t)$.

Exemplo 4.2.1 As seguintes fórmulas são fórmulas objetivo :

a) $\top$, porque é um parâmetro do tipo t.

b) (Pai José João) onde Pai é do tipo $(p \rightarrow p \rightarrow t)$ e tipo de José = tipo de João = p. Neste caso a fórmula é uma proposição. Da mesma forma seria (Pai x João) com tipo de x = p e (f José João) com tipo de f = $(p \rightarrow p \rightarrow t)$. No último caso, trata-se de uma fórmula objetivo flexível pois f é variável e está na cabeça da fórmula.

c) $\vee$ (Pai x João) (Surdo x) é uma fómula objetivo. Tem-se neste caso, a disjunção de duas proposições.

d) $\exists x$ (λx (Pai x João)) pois, (λx (Pai x João) é um predicado do tipo $(\alpha \rightarrow t)$ onde α = tipo de x = p como no exemplo b .

e) (Inverno maio Brasil)

f) $\vee$ (Pai x José) (Padre x)

g) $\wedge$ (Padre y) (Filho y João)

h) $\wedge$ (Sucessor x 20) (Quadr (sucessor x y) 405)

i) $\exists x \lambda x$ (sucessor x 32)

Observa-se que o domínio das fórmulas objetivo continua sendo H^+, o Universo Positivo de Herbrand, onde se definem as fórmulas objetivo.

Com estes conceitos estabelece-se a noção de sentença definida de ordem superior, que tem a forma

$$\forall \bar{x}\ \mathrm{G} \Rightarrow \mathrm{A},$$

em que G é uma fórmula objetivo qualquer, A é um atomo rígido e $\bar{x}$ representa uma lista de variáveis livres em G ou A.

Adota-se que sua representação correta é

$$\forall \bar{x}(\sim \mathrm{G} \vee \mathrm{A})$$

Assume-se que sentença definida seja universalmente quantificada em todas as variáveis livres de G e A e que tenha a interpretação de uma implicação lógica de uma fórmula objetivo (G) sobre um átomo rígido positivo (A). Observa-se então que as sentenças definidas

podem conter os conectivos $\vee$, $\wedge$ e $\exists$ na sua parte esquerda (isto é, em G) além de poderem ser $\top \Rightarrow$ A. Esta última forma representa o fato A.

Como foi dito anteriormente, pode-se retirar os conectivos, como na lógica de primeira ordem, por meio de transformação em sua forma normal conjuntiva. Já que as substituições podem introduzir disjunções e existenciais, é melhor deixá-los explícitos.

Como na lógica de primeira ordem, pode-se interpretar computacionalmente e mais convenientemente as sentenças definidas de ordem superior. Ao se agrupar um conjunto de sentenças definidas, P, pode-se pensar em uma consulta G sobre este conjunto. Deseja-se responder a uma consulta G, que é uma fórmula objetivo, com respeito a esse programa P.

A base de conhecimento para a programação lógica de ordem superior (interpretada computacionalmente como um programa P) constitui-se portanto de sentenças definidas de ordem superior, como as que se seguem:

$$\forall f \ \top \Rightarrow (\text{ mapfun f nil nil })$$
$$\forall x \, \forall f \, \forall l_1 \, \forall l_2 \, ((\text{mapfun f } l_1 \, l_2) \Rightarrow (\text{mapfun f } (\text{ cons } x \, l_1) \; (\text{cons } (\text{ f x }) \, l_2))$$

A possibilidade de se ter funções como argumento permite a este formalismo emular linguagens funcionais como o LISP. As fórmulas acima definem um predicado chamado *mapfun* correspondente à função *maplist* do LISP, cuja finalidade é obter uma lista com os resultados da aplicação de uma função f aos elementos de outra. A primeira sentença diz que a aplicação de f aos elementos de uma lista vazia *nil* gera outra lista vazia.

A segunda sentença espera uma lista não vazia l_1 contendo os elementos sobre os quais se aplica f. Assim *mapfun* é aplicado a uma lista cujo primeiro elemento é x e o restante é l_1, tendo como resultado l_2 a junção do resultado de $(f \, x)$ à lista l_2, obtida por meio da função *cons*, também homônima do LISP.

Para os mecanismos de unificação e resolução que serão usados sobre a base, não importa o símbolo usado para representar $\Rightarrow$. Interessam mais a correção gramatical de G e de A. Assume-se também a forma clausal das sentenças definidas, ou seja, sem o(s) quantificador(es) universal. Assim as cláusulas definidas de ordem superior têm as seguintes formas indutivas:

i) $\sim$ G $\vee$ A

ii) $\sim \exists \lambda x \sim$ D em que D é uma cláusula definida e $\exists$ tem o tipo $((\alpha \rightarrow t) \rightarrow t)$

Na prática, as cláusulas não serão colocadas desta forma na base de conhecimento. As seguintes sentenças definidas,

$\top \Rightarrow$ (relprim mother)
$\top \Rightarrow$ (relprim wife)
$\forall r$ ((relprim r) $\Rightarrow$ (rel r))
$\forall r \forall s$ ((relprim r $\wedge$ relprim s) $\Rightarrow$ (rel $\lambda x \lambda y \exists z$ ((r x y) $\wedge$ (s z y)))
$\top \Rightarrow$ (mother jane mary)

$$\top \Rightarrow \text{(wife john jane)}$$

onde *relprim* expressa uma relação familiar primária e *rel* expressa uma relação familiar qualquer entre duas pessoas (jane, mary, john, z, y e x), tiradas de [Nada87], são transcritas na base de dados numa forma análoga àquela do PROLOG, onde as cláusulas são representadas na forma:

A :- G

Nos casos em que G é a constante $\top$, abrevia-se a cláusula para A e assume-se que as variáveis em G e A são universalvente quantificadas. Denota-se ainda os simbolos $\wedge$ e $\vee$ por vírgula e ponto e vírgula, respectivamente, reservando-se os rótulos Sigma, \ e Ver respectivamente para $\exists$, λ e $\top$. O conhecimento acima será expresso por:

```
relprim mother
relprim wife
rel r :- relprim r
rel \x\y (sigma \ z (r x z , s z y)) :- relprim r , relprim s
mother jane mary
wife john jane
```

O processo de resposta a uma consulta G passa, naturalmente, pela unificação, substituição e resolução, e é neste contexto que prossegue a discussão. Tome-se uma substituição $\sigma = \{\langle x_1, t_1\rangle, \langle x_2, t_2\rangle \cdots \langle x_n, t_n\rangle\}$ com o tipo de x_i = tipo de t_i. A aplicação da substituição a uma fórmula, por exemplo B, resulta na seguinte forma λ-normal :

$$\sigma(\text{B}) = (\ \lambda x_1 \cdots x_n\ (\text{B}\ t_1 \cdots t_n\ )\ ).$$

Por exemplo a aplicação da substituição { x/λy (Pai y) , z/2 } à formula (Pai x z) resulta em

$$\lambda xz\ (\text{Pai x z})\ (\lambda y\ (\text{Pai y}))\ 2$$

que β-reduzida se converte em

$$(\text{Pai}\ (\lambda y\ (\text{Pai y}))\ 2)$$

Suponha que y é uma lista de todas as variáveis livres da consulta G que se quer aplicar sobre um programa genérico P. G é respondida afirmativamente se $\text{P} \vdash \exists y\ \text{G}$, ou seja, se há uma substituição σ tal que $\sigma(G)$ é teorema de P.

A resposta desejada requer a confirmação da existência de y e o conhecimento de seu valor. Em lógica de ordem superior esta confirmação pode não ser possível para um conjunto arbitrário de fórmulas. No entanto, se P for um conjunto de cláusulas definidas esta pesquisa é possível. Este resultado é assegurado pelo teorema abaixo cuja prova pode ser vista em [Nada87].

Teorema : Seja uma substituição positiva aquela em que os termos t_i pertencem a H^+ e substituição fechada aquela onde os termos t_i são fórmulas fechadas. Seja P um conjunto de fómulas definidas e representa-se por |P| uma instanciação do programa P. |P| é formado pelas fórmulas $\sigma(G \Rightarrow A)$ em que $\forall x$ G $\Rightarrow$ A é uma fórmula de P.

Assuma ainda que σ é uma substituição positiva e fechada para x. Seja G pertencente a H^+ uma fórmula objetivo. Então :

a) Se G é $G_1 \wedge G_2$ então P $\vdash$ G se e somente se P $\vdash G_1$ e P $\vdash G_2$.

b) Se G é $G_1 \vee G_2$ então P $\vdash$ G se e somente se P $\vdash G_1$ ou P $\vdash G_2$.

c) Se G é $\exists x$ B em que B é do tipo $\alpha \rightarrow t$ então P $\vdash$ G se e somente se existe uma fórmula fechada u $\in H^+$ tal que P $\vdash$ λ-normal (B u), isto é, uma aplicação de B a u.

d) Se G é um átomo então P $\vdash$ G se e somente se existe uma fórmula $G_1 \Rightarrow$ G $\in$ |P| tal que P $\vdash G_1$.

Uma consequência do teorema acima é a possibilidade de estabelecer uma visão procedural para as cláusulas e, na sequência, a mesma visão para provas de fórmulas objetivo sobre um conjunto de sentenças definidas.

Numa cláusula definida $\forall x$ (G $\Rightarrow$ A), G pode ser $\top$ ou uma fórmula que contenha conjunções, disjunções e/ou quantificadores existenciais. Se G é T, então tem-se $\forall x$ A como um fato, caso contrário, ela é interpretada como uma declaração de procedimento, na qual a cabeça do átomo A é o nome do procedimento que está sendo definido e G é o procedimento que está dentro de A. Assim, num exemplo sem preocupação com tipos, sejam as cláusulas G = (pai x) e A = (casado x) e $\forall x$ (G $\Rightarrow$ A).

A expressão

casado x :- pai x

é interpretada como um procedimento *casado* que para ser executado chama o procedimento *pai*.

Intuitivamente, para saber se x é casado pergunta-se se x é pai (pois há um conhecimento na base que diz que todo individuo que é pai, é casado)

O teorema assegura que é preciso considerar substituições positivas a fim de estabelecer uma prova para um conjunto de cláusulas definidas.

Aliando-se a este fato, a observação de que a substituição positiva em H^+ produz um resultado ainda em H^+, pode-se definir, mesmo na presença de variáveis predicativas, um provador de teoremas para este subconjunto de sentenças, usando exatamente a visão procedural dada acima para as cláusulas definidas.

A descrição do modelo de pesquisa para atender a uma consulta requer o conhecimento da unificação (que é interpretada como uma passagem de parâmetro por nome) e da derivação, que serão tratados em 4.3.1 e 4.3.2. Suscintamente, uma consulta G que é uma fórmula objetivo, deve ser unificada com A (passagem de parâmetro) que pode ativar o procedimento G', caso exista em P uma cláusula $\forall x$ (G' $\Rightarrow$ A).

O tipo de ramificação da árvore de pesquisa fica determinado pelos possíveis conectivos em G, que podem ser $\wedge$, $\vee$ ou $\exists$. No primeiro caso é uma árvore "e", no segundo uma árvore não determinística "ou" e no último uma árvore infinita com parâmetros em HB.

Exemplo 4.2.1 Um conjunto P, tem as seguintes cláusulas definidas :

$$\top \Rightarrow \forall f \text{ (mapfun f nil nil)}$$

$$\forall f\ \forall l_1\ \forall l_2(mapfun\ f\ l_1\ l_2)) \Rightarrow (mapfun\ f\ (cons\ x\ l_1\ )\ (cons\ (f\ x)))$$

onde os tipos são os seguintes :

vari 'avel ou parâmetro	tipo
f	$int \rightarrow int$
l_1,l_2	$(list\ int)$
mapfun	$(int \rightarrow int) \rightarrow (list\ int) \rightarrow (list\ int) \rightarrow int$
cons	$int \rightarrow (list\ int)$

Este é um programa que tem o mesmo significado daquele homônimo "mapfun", no LISP. Suponha a seguinte consulta:

$$\exists l \text{ (mapfun } \lambda x \text{ (g x 1) (cons 1 (cons 2 nil)) } l \text{)}$$

cuja interpretação é a seguinte: existe lista l que 'e o resultado da aplicação de λx (g x 1) aos membros da lista (1 , 2) ? Seguramente l é a lista ((g 1 1) (g 2 1)) ou (cons (g 1 1) (cons (g 2 1) nil)).

A consulta poderia ser feita de outra forma, ou seja, existe função f (e qual) que aplicada a determinada lista fornecerá outra determinada lista, como abaixo:

$$\exists f \text{ (mapfun f (cons 1 (cons 2 nil))}$$
$$\text{(cons (g 1 1) (cons (g 1 2) nil)))}$$

Aqui a resposta à consulta deverá ser dada encontrando-se a substituição {f/λx (g 1 x) } que está um tanto intuitiva, e que tambem é possível obter em LOS.

4.3 Programação Lógica de Ordem Superior

A mecanização lógica de ordem superior consiste na implementação de mecanismos que permitam a derivação de um conjunto de informações, a partir de uma certa base de conhecimento e de uma consulta feita a esta base . A obtenção destas informações permite concluir sobre a existência ou não de uma prova sob a teoria para o fecho existencial de uma fórmula objetivo, que é a consulta, e um conjunto de sentenças definidas que é a base de conhecimento.

Pressupõe-se que o sistema que a ser construido contenha procedimentos que permitam acessar rapidamente a base de conhecimento, verificar a adequação dos tipos e inferir o tipo de novas expressões geradas. Um dicionário da base deve estar disponível de modo a facilitar a consulta em termos de nomes de predicados, fatos, variáveis e posicionamento dos predicados dentro da base. As informações referentes à tipificação de variáveis e constantes da consulta fazem parte do processo e são tratadas antes da derivação.

O mecanismo da derivação consiste de um enumerador de todas as possíveis sequências de derivação relativas a uma determinada base de conhecimento e consulta. As sequências são direcionadas pela consulta fornecida.

Um passo de derivação pode consistir de uma das seguintes opções:

a) Quebra da consulta tomando-se uma parte da mesma para prosseguir na derivação. É preciso lembrar que uma consulta G é uma fórmula objetivo e, pela definição, está sujeita a conter os conectivos $\exists$, $\wedge$ e $\vee$;

b) Busca de uma regra ou fato na base de conhecimento que tenha possibilidade de ser unificada com a consulta ou parte dela

c) Unificação de duas expressões caso a parte (b) gere um conjunto discórdia (definido na subseção 4.3.1) de tipo apropriado.

A segunda opção da derivação (b) pode ser vista como uma noção generalizada da noção da resolução SLD que existe na maioria das discussões sobre cláusulas definidas de primeira ordem e que foram introduzidas por K. R. Apt (1982) [Apt:82], e que aqui são utilizadas para a ordem superior.

O processo todo é chamado aqui de *P-derivação* e não mais de *resolução*. Pretende-se obter uma prova para fórmula objetivo G, sujeita a conter os parâmetros $\wedge$, $\vee$ e $\exists$, sobre uma base de conhecimento P. A fórmula objetivo G é a consulta.

Para que uma prova seja obtida pode-se requerer uma sequência de unificadores entre os termos na sequência de derivação. Tais unificadores são mantidos e combinados entre si (por meio da composição como na lógica de primeira ordem) para se obter ao final não apenas a comprovação da existência ou não da prova para G, mas também indicar em que condições ou seja, com quais substituições isto é possível.

A seguir descreve-se-se como é feita a derivação. Para isto, é necessário controlar a estrutura de G (item a) de modo que que toda ela seja atendida, lembrando que cada conectivo requer tratamento diferenciado. O processo de pesquisa (letra b acima) e a unificação (letra c) devem ser processados de modo a permitir o retroencadeamento, desta vez mais complexo que aquele da primeira ordem devido à possibilidade de geração de mais de um unificador entre dois termos. Os itens b) e c) apresentam-se mais trabalhosos e fazem parte da pesquisa propriamente dita. Para melhor visualização e entendimento são apresentados em primeiro lugar (subseção 4.3.1). O item a), por se tratar de mecanismo de controle, é discutido posteriormente na subseção 4.3.2 onde foi incluido no fechamento do procedimento de pesquisa.

4.3.1 Unificação de Ordem Superior

A tarefa de unificar duas fórmulas bem formadas de ordem superior tem sido estudada por diversos pesquisadores e dentre êles, de forma mais extensiva e detalhada, por [Huet75]. Um trabalho inicial para unificação de termos de segunda ordem foi apresentado por Tomasz Pietrzkowsky (1973) [Piet73]. O procedimento de unificação apresentado baseou-se em cinco regras: eliminação, imitação, projeção, repetição e identificação. O autor incorporou ainda algumas heurísticas com o objetivo de remover substituições supérfluas. O significado dos termos imitação e projeção é exposto adiante na apresentação do algoritmo que as usa.

[Huet75] apresentou um algoritmo de unificação para o λ-cálculo tipado de ordem ω (ou seja, sem limite superior para a ordem da lógica,) e provou sua correção. O algoritmo já não usa as regras de eliminação, repetição e identificação de Pietrzkowsky. No trabalho foram utilizadas duas rotinas principais para o algoritmo da unificação chamadas de SIMPL e a MATCH. A primeira cria e simplifica um conjunto de discórdia a partir dos termos a serem unificados e a segunda aplica as regras de imitação e projeção aos pares discordantes, quando possível.

O detalhamento destas rotinas, bem como a explicação do significado dos termos *imitação e projeção*, é feito adiante quando é abordado o algoritmo exposto por [Nada87] que também as usa com pequenas variações. No entanto, este algoritmo pressupõe-se que os dois termos a serem unificados sejam sentenças definidas e, além disso, que estejam na forma normal de unificação. Para melhor entendimento do algoritmo são necessárias algumas definições.

Um *par de discórdia* é um par de fórmulas bem formadas do mesmo tipo. Um *conjunto de discórdia* é então um conjunto finito, $\{\langle F^i, H^i\rangle\}$, $1 \leq i \leq n$ de pares de discórdia. Um *unificador* para o conjunto é uma substituição σ tal que, para $1 \leq i \leq n$, $\sigma(F^i) = \sigma(H^i)$. O problema da unificação de ordem superior torna-se então o de saber se há unificadores para um dado conjunto de discórdia e se houver, explicitá-los.

Exemplo 4.3.1.1 O conjunto discórdia {⟨x,3⟩, ⟨z, 4⟩} tem como unificador a substituição { x/3 , z/4 }. Este é um unificador para o conjunto. Em nível mais complexo {⟨ f / λu A ⟩ } é um unificador para {⟨ f(A) , A ⟩}. Na segunda parte do exemplo o conjunto discórdia é unitário.

Na lógica de ordem superior a questão da existência de um unificador para um conjunto de discórdia arbitrário é indecidível, o que não acontece na lógica de primeira ordem. A dificuldade encontra-se em que dados dois termos quaisquer é preciso encontrar uma substituição que aplicada a cada um dos termos os tornem iguais. Além disto, que qualquer outra substituição que provoque o mesmo resultado seja uma instância da primeira, ou seja, que a primeira seja mais geral . Tal substituição mais geral é conhecida como *mgu*. A noção de um *mgu* (most general unifier), bem conhecida em primeira ordem, não tem sua correspondência na lógica de ordem superior onde mais de um unificador pode ser

encontrado, nenhum dos quais pode ser obtido de outro por meio de uma composição com uma substituição.

A questão da decidibilidade da unificação na lógica de ordem superior está resumida em [Knig89]. Dele é o seguinte exemplo: Considere dois termos (f x b) e (A y) para os quais se procura um unificador, ou seja uma substituição para as variáveis *f, x e y*. Pode-se encontrar:

f/λuv.u, x/(A y), y/y ou
f/λuv.(A (g u v)), x/x, y/(g x b)

em que nenhuma delas é mais geral do que a outra. Naquele trabalho, Knight apresenta extensa bibliografia referente à prova da indecibilidade da unificação a partir da lógica de segunda ordem.

A mecanização da lógica de ordem superior, para que possa utilizar uma regra similar à resolução usada na primeira ordem, apresenta dificuldades de que em certos pares, pode existir uma infinidade de unificadores independentes. O reconhecimento de que dois termos apresentem uma instância comum pode então tornar-se indecidível.

Este problema é contornado por Nadatur [Nada87], seguindo Huet [Huet75], observando-se que em certos conjuntos de discórdia no mínimo um unificador pode ser facilmente encontrado, e que em outros tipos de conjuntos, nao há possibilidade de haver unificadores. Esta primeira classificação dos conjuntos facilita a tarefa de encontrar unificadores, quando existentes. Para classificar um conjunto em um dos tipos usa-se iterativamente duas rotinas (SIMPL e MATCH) sobre o conjunto de discórdia inicial. Assim, se houver unificadores, um procedimento poderá ser usado para encontrar alguns deles.

Antes de expor os dois algoritmos que executam a unificação é necessário definir uma forma normal adequada para trabalhar com as fórmulas.

Uma fórmula β-normal

$$\mathrm{F} = \lambda x^1 \cdots \lambda x^n (H A^1 \cdots A^m)$$

é uma *fórmula normal de unificação* se o tipo de F é da forma $\alpha_1 \rightarrow \cdots \rightarrow \alpha_m \rightarrow \alpha_t$ onde α_t é um tipo fundamental e para $1 \leq i \leq m$, A^i é também uma fórmula normal de unificação. Observe-se que toda fórmula pode ser transformada em sua forma normal de unificação. Em modo mais direto, esta forma explicita todos os argumentos de H, e dos A^i. Se os A^i não forem de tipos fundamentais, esta transformação não implica que na forma β-normal eles serão. Terão seus argumentos explicitados e ao mesmo tempo abstraidos. Esta transformação mantém o tipo.

Toda fórmula bem formada tem uma forma normal de unificação que pode ser obtida convertendo-se em primeiro lugar a fórmula bem formada para uma fórmula λ-normal e realizando-se posteriormente uma série de η-(conversões) expansões. Pode ser observado que tais formas são únicas a menos de um renomeamento das variáveis ligadas. Se F é uma fórmula bem formada, então $\tilde{F}$ é usado como notação para a forma normal de unificação de F. Note que se σ é uma substituição então $\sigma(F) = \sigma(\tilde{F})$.

Uma das justificativas para representar fórmulas bem formadas na forma normal de

unificação é que se pode analisar mais facilmente os efeitos das substituições e, consequentemente, tomar decisões a respeito dos unificadores.

Uma fórmula normal de unificação é *rígida* se H é uma constante ou é um elemento do conjunto $\{x^1, \cdots, x^n\}$ de variáveis de ligação e é *flexível* se não for rígida.

A base da primeira fase da simplificação na busca de unificadores para um dado conjunto de discórdia, onde as duas fórmulas *rígidas*

$$F^1 = \lambda x^1. \cdots x^n.(H^1 A^1 \cdots A^r) \text{ e}$$
$$F^2 = \lambda x^1. \cdots x^n.(H^2 B^1 \cdots B^s)$$

estão na forma normal de unificação e são do mesmo tipo é que: se σ for um unificador para $\{\langle F^1 , F^2 \rangle\}$ então

1) $H^1 = H^2$, e

2) σ é um unificador para $\{\langle \lambda x^1. \cdots x^n\ A^i, \lambda x^i. \cdots x^n\ B^i \rangle\}$, $(1 \leq i \leq r)$

Desta forma se F^1 e F^2 são rígidas, há meios de determinar que F^1 e F^2 não tem unificadores ou de reduzir o problema de encontrar unificadores para F^1 e F^2 ao problema de encontrar unificadores para os pares de argumentos destas fórmulas bem formadas.

Sem a preocupação da interpretação das fórmulas, esta primeira etapa diz que dadas, por exemplo, duas fórmulas

$$F^1 = \lambda x^1.\ (\text{ Ancestral (pai Pedro) } x^1\ ) \text{ e}$$
$$F^2 = \lambda y^1.\ (\text{ Ancestral } y^1 \text{ Maria }\ )$$

deve-se reduzir o problema a encontrar unificadores para os pares *(pai Pedro)* e y^1 e em seguida para x^1 e *maria*, já que as cabeças *(Ancestral)* são rígidas e iguais.

Esta *SIMPL*ificação efetuada sobre um par de discórdia é a tarefa da rotina SIMPL na unificação.

O algoritmo da unificação parte de um par de discórdia que se deseja unificar. Como ponto de partida, para se efetuar uma consulta, tem-se uma fórmula objetivo, que é a consulta, e deseja-se torná-la, ou parte dela, igual a um átomo A de uma fórmula $G \Rightarrow A$ da base de conhecimento que está na forma A := G. Para isso deve-se unificá-la com A. Este é o primeiro conjunto discórdia do procedimento global de prova ou resposta a uma consulta. Este par é constituido de duas fórmulas bem formadas do mesmo tipo (t), na forma normal de unificação.

Os passos iniciais podem ser observados na figura 4.1. A este par se aplica a rotina SIMPL que, atua sobre os pares rígido-rígido comparando suas cabeças e formando novos pares de discórdia com o emparelhamento dos argumentos. Se novos pares rígido-rígido aparecerem, sobre estes se repete o procedimento. O surgimento de um par rígido-rígido com cabeças diferentes determina a *falha* **F** na unificação. Se não houver unificação, a consulta não pode ser respondida e todo procedimento de consulta deve pesquisar na base se não há outra fórmula que poderia satisfazer a unificação. Caso haja novo fracasso nesta nova pesquisa, o procedimento de prova como um todo reporta a falha.

Algoritmo Para um Ramo da Árvore de Unificação : Seja o conjunto D de discórdia constituido de pares de fórmulas em sua forma normal de unificação.

1.1 Enquanto houver em D um par rígido-rígido
Aplique a ele a simplificação (SIMPL) .
Se há em D um par rígido-rígido de cabeças diferentes
então determine Falha **F** e retorna.
Se há em D apenas pares flexível-flexível ou D = $\emptyset$
então a unificação está pronta. Retorne.
1.2 Tome um dos pares flexível-rígido de D .
1.3 Procure os unificadores para o par.
1.4 Aplique um dos unificadores (a substituição) ao conjunto D.
Volte para 1.1

Figura 4.1: Macro para a unificação : Nesta visão global considera-se apenas um ramo da árvore de unificação. Os demais ramos são determinados pelos outros unificadores deixados no passo 1.4.

A rotina SIMPL deixa de ser aplicada quando o conjunto de discórdia resultante for vazio ou composto apenas de pares flexível-rígido ou flexível-flexível ou ambos. Na sequência do processo, é aplicada a rotina MATCH, que pesquisa para encontrar os unificadores e cujos detalhes estão ainda nesta seção.

Quando o procedimento da unificação encontra-se na situação acima, um par flexível-rígido é eleito[1] para, sobre ele, se aplicar a rotina MATCH. Esta rotina fornecerá os possíveis unificadores (se houver) para o par escolhido. A figura 4.1 fornece uma visão geral da rotina de unificação.

Neste ponto inicia-se a formação de uma árvore de unificação, cujo nó raiz é o par discordante (duas fórmulas normais de unificação do mesmo tipo) e cujos ramos são rotulados com os possíveis unificadores do par escolhido para MATCH. Os novos nós são formados aplicando-se o unificador, que rotula o ramo, ao conjunto discórdia de sua raiz. Repete-se o ciclo SIMPL-MATCH a cada nó assim formado. Veja figura 4.4.

Antes de passar à rotina SIMPL observa-se que na forma normal de unificação duas fórmulas do mesmo tipo têm o mesmo número de variáveis ligadas. Portanto o tamanho do vetor $\bar{x}$ em $\lambda\bar{x}$ é o mesmo para F^1 e F^2 . Exemplificando, seja Pai uma constante do tipo $(p \rightarrow p \rightarrow t)$. Abaixo são mostradas aplicações e reduções sobre o parâmetro Pai com A, B e x do tipo p .

$1.Pai_{(p\rightarrow p\rightarrow t)}$	$Pai_{(p\rightarrow p\rightarrow t)}$	$Pai_{(p\rightarrow p\rightarrow t)}$
$2.(Pai\ x)_{(p\rightarrow t)}$	$(Pai\ B)_{(p\rightarrow t)}$	$(\lambda x Pai_{(p\rightarrow p\rightarrow p\rightarrow t)}$

[1]Aqui está a primeira fonte de não determinismo da derivação.

3.$((Pai\ x)\ A)_t$	$((Pai\ A)\ x)_t$	$(\lambda y(\lambda x Pai))_{(p\to p\to p\to p\to t)}$
4.$(\lambda x((Pai\ x)\ A))_{(p\to t)}$	$(\lambda x((Pai\ B)\ x))_{(p\to t)}$	$((\lambda y(\lambda x Pai))\ B)_{(p\to p\to p\to t)}$
5.$((\lambda x((Pai\ x)\ A))\ B)_{(t)}$	$((\lambda x((Pai\ B)\ x))\ A)_{(t)}$	$(\lambda x Pai)_{(p\to p\to p\to t)}$
6.$((Pai\ B)\ A)_{(t)}$	$((Pai\ B)\ A)_{(t)}$	$((\lambda x Pai)\ A)_{(p\to p\to t)}$
7.$((\lambda y(\lambda x((Pai\ B)\ A)_{(p\to p\to t)}$		

As duas primeiras fórmulas na linha 4 têm o mesmo tipo $(p \to t)$ e estão na forma normal de unificação. Na coluna 1, observa-se que as fórmulas 1 e 7 têm o mesmo tipo, só não estão na forma normal de unificação. A sentença 1 seria

$$\lambda x \lambda y\ (Pai\ y\ x)_{p\to p\to t}$$

em sua forma normal de unificação. A fórmula 7 já se encontra na forma normal de unificação.

Observe-se ainda a diferença entre as duas fórmulas

$(\lambda x\ (F\ x\ 4))_{\alpha\to t}$ e
$((\lambda x(F\ x))\ 4)_{i\to t}$

Suponha que o tipo de F seja $\alpha \to i \to t$. Representa-se por $F_{\alpha\to i\to t}$. Os parâmetros x e 4 são do tipo α e *int*, respectivamente.

A primeira fórmula pode ser aplicada a Pedro por exemplo, resultando em $((\lambda x(F\ x\ 4))$ Pedro) que após a β-conversão equivale a *F Pedro 4*.

A segunda fórmula não está correta pois 4 não é do tipo α. Substituindo 4 por Pedro, para corrigir a inadequação de tipos, ter-se-ia $((\lambda$ x (F x) Pedro$)_{i\to t}$. Nesta situação a fórmula pode sofrer uma β-redução para $(F\ Pedro)_{i\to t}$. Tambem pode sofrer uma η-redução para $(F\ Pedro)_{i\to t}$.

Na segunda fórmula ocorreu o seguinte desenvolvimento errado:

$F_{\alpha\to i\to t}$
$(F\ x)_{i\to t}$
$\lambda x(F\ x)_{\alpha\to i\to t}$
$(\lambda x(F\ x)\ 4)_{\ldots\ldots i\to t}$

Esta última está errada porque o tipo não é adequado para a substituição, pois 4 é do tipo *i*.

No entanto se os dois tipos de argumentos fossem iguais,isto é, se F fosse do tipo $(i \to i \to t)$

$(\lambda x(F\ x\ 4))$ e $((\lambda x(F\ x))4$

estariam corretas e teriam o mesmo tipo e a mesma cabeça e não seriam a mesma fórmula.

O desenvolvimento para se chegar à primeira fórmula foi o seguinte:

$F_{\alpha\to i\to t}$
$(F\ x)_{i\to t}$
$(F\ x\ 4)_t$

Definição da Rotina SIMPL : Seja o conjunto D de discórdia constituido de pares de fórmulas em sua forma normal de unificação.

1) Se D = $\emptyset$ então SIMPL(D) = $\emptyset$

2) Se D = $\{\langle F^1, F^2\rangle\}$ e

a) F^1 é flexível então SIMPL(D) = D; caso contrário

b) Se F^2 é flexível então SIMPL(D) = $\{\langle F^2, F^1\rangle\}$;

c) De outro modo F^1 e F^2 são ambos rígidos. Sejam $\lambda\bar{x}.(C^1\ A^1 \cdots A^r)$ e $\lambda\bar{x}.(C^2\ B^1 \cdots B^s)$ as formas normais de unificação para F^1 e F^2. Se $C^1 \neq C^2$ SIMPL(D) = **F**, caso contrario SIMPL(D) = SIMPL($\{\langle \lambda\bar{x}.A^i, \lambda\bar{x}.B^i\rangle\ ,\ 1 \leq i \leq r\}$)

3) De outro modo D tem no mínimo dois membros. Seja D = $\{\langle F^i, G^i\rangle\}\ ,\ 1 \leq i \leq n\}$, onde n é o numero de pares do conjunto.

a) Se SIMPL($\{\langle F^i, G^i\rangle\}$) = **F** para algum i então SIMPL(D) = **F**

b) Senão SIMPL(D) = $\cup_{i=1}^{n}$ SIMPL($\{\langle F^i, G^i\rangle\}$).

Figura 4.2: Definição da rotina SIMPL

$$(\lambda x(F\ x\ 4)_{i \rightarrow t}$$

As formas normais de unificação, para as duas fórmulas, são

$$(\lambda x(F\ x\ 4)) \text{ e } ((\lambda x(F\ 4\ x))$$

respectivamente, que só se unificariam para x = 4 .

A Simplificação

Conforme foi dito anteriormente, o resultado da rotina SIMPL pode ser um insucesso (quando retorna **F**), um conjunto discórdia com pares flexível-rígido e flexível-flexível, um conjunto discórdia apenas com pares flexível-flexível ou um conjunto vazio. A situação de insucesso é provocada pela discordância de duas cabeças rígidas. Neste caso o par de discórdia inicial não possui unificador. No segundo caso, isto é, na existência de apenas pares flexível-flexível e flexível-rígido, o procedimento prossegue como será visto adiante. Nos dois últimos casos haverá sucesso na busca de um unificador inicial para um dos pares obtidos por meio da simplificação.

A rotina SIMPL que se aplica aos conjuntos de pares de discórdia D é definida na figura 4.2.

Exemplo 4.3.1.2 Suponha que se deseja aplicar a rotina SIMPL às seguintes fórmulas vistas em [Nada87]

$\langle$ (mapfun f_1 (cons x l_1) (cons (f_1 x) l_2)),
(mapfun f_2 (cons 2 nil) (cons (g 1 1) (cons (g 1 2) nil)))$\rangle$.

O resultado apresenta o seguinte conjunto discórdia:

$\{\langle f_1, f_2\rangle$, $\langle$ x , 2 $\rangle$, $\langle$ l_1, nil$\rangle$,$\langle$(f_1 x), (g 1 1) $\rangle$,
$\langle$ l_2, (cons (g 1 2) nil)$\rangle$ $\}$.

A rotina SIMPL é recursiva. No exemplo anterior, ao se cancelar as duas constantes *mapfun* obtem-se os pares

$\langle f_1, f_2\rangle$,
$\langle$ (cons x l_1), (cons 2 nil)$\rangle$,
$\langle$(cons (f_1 x) l_2), (cons (g 1 1) (cons (g 1 2) nil)))$\rangle$.

No segundo e terceiro pares pode-se eliminar as cabeças *cons* obtendo o resultado expresso acima.

A intenção de SIMPL é que o conjunto simplificado tenha os mesmos unificadores do conjunto fornecido. Consegue-se isto com um número finito de passos e pode ser enunciado da seguinte forma: SIMPL é uma função computável total sobre conjunto de pares de discórdia. Além disso, se D é um conjunto de pares de discórdia então σ pertence ao conjunto de unificadores de D se e somente se SIMPL(D) $\neq$ **F** e σ pertence ao conjunto de unificadores de SIMPL(D).

A primeira fase da unificação para um conjunto discórdia D consiste portanto em avaliar SIMPL(D). Se o resultado for **F** não há unificador para D (é o caso em que há cabeças de diferentes para as duas fórmulas rígidas emparelhadas). A pesquisa não obtém sucesso e os mecanismos de controle pesquisam outras possibilidades, por exemplo,o mecanismo de retroencadeamento ou escolhe outra sentença da base de conhecimento para formar novo conjunto de discórdia D.

Se o resultado for um conjunto vazio passou-se à SIMPL um conjunto vazio. Isto ocorre quando é encontrado um unificador para D e a aplicação deste unificador ao conjunto igualou os pares do conjunto. Nesta ocasião estes pares são eliminados do conjunto D, que pode porisso se tornar vazio.

Se D apresentar apenas pares flexível-flexível, no mínimo um unificador pode ser encontrado facilmente para o conjunto pode ser encontrado e ao final do procedimento de derivação deve-se considerar a substituição. Como caso de análise especial será visto na seção 4.4.

As situações de D vazio ou constituido apenas de pares flexível-flexível, representam casos em que houve ou pode haver solução para a unificação e este conjunto é chamado de *resolvido.*

Definição da rotina MATCH

Sejam V um conjunto de variáveis, F^1 uma fórmula bem formada flexível e F^2 uma fórmula bem formada rígida do mesmo tipo de F^1 e sejam $\lambda\bar{x}.(f\ A^1 \cdots A^r)$, e $\lambda x.(C\ B^1 \cdots B^s)$ as formas normais de unificação de F^1 e F^2. Além disto, seja $\alpha_1 \rightarrow \cdots \rightarrow \alpha_r \rightarrow \beta$ o tipo de f e para $1 \leq i \leq r$, seja w^i uma variável do tipo α_i

i) Se C é uma variável (isto é C aparece em $\bar{x}$), então IMIT(F^1, F^2, V) = $\emptyset$; Caso contrário (C é constante) sejam $h^i \notin$ V $\cup\{w^1, \cdots, w^r\}$ variáveis de tipos adequados para $1 \leq i \leq s$, isto é, do mesmo tipo de $B^1 \cdots B^s$ então IMIT(F^1, F^2, V) = $\{\langle f, \lambda w^1.\cdots\lambda w^r.(C(h^1 w^1 \cdots w^r)\cdots(h^s w^1 \cdots w^r))\rangle\}$.

ii) Para $1 \leq i \leq r$, se α_i não for da forma $\beta_1 \rightarrow \cdots \rightarrow \beta_t \rightarrow \beta$ então $PROJ_i(F^1, F^2,$ V) = $\emptyset$; Caso contrário, sejam $h^i \notin V \cup \{w^1, \cdots, w^r\}$ variáveis de tipos adequados para $1 \leq i \leq t$, então $PROJ_i(F^1, F^2, V)$ = $\{\langle f, \lambda w^1.\cdots\lambda w^r.(w^i\ (h^1\ w^1 \cdots w^r)\cdots(h^t\ w^1 \cdots w^r))\rangle\}$.

iii) MATCH(F^1, F^2, V) = IMIT(F^1, F^2, V) $\cup$ $\bigcup_{1\leq i\leq r} PROJ_i(F^1, F^2, V)$

Figura 4.3: Definição da rotina MATCH

Unificação Propriamente Dita (MATCH)

Se o conjunto tem no mínimo um par flexível-rígido, então é usada a rotina MATCH para prosseguir na procura de um unificador. A rotina MATCH é definida na figura 4.3.

Intuitivamente MATCH sugere maneiras pelas quais uma fórmula bem formada flexível pode se assemelhar a fórmula bem formada rígida. Uma maneira é fazer sua cabeça "imitar" (copiar) a cabeça da fórmula bem formada rígida. No contexto dos termos de primeira ordem, de fato, esta é a única maneira pela qual dois termos podem ser igualados. Se as fórmulas bem formadas são de ordem superior, no entanto, isto pode ser feito também pela "projeção" de um dos argumentos da fórmula bem formada flexível como cabeça e fazendo o termo resultante se assemelhar ao termo rígido. Este aspecto da unificação de termos de ordem superior torna ramificada a busca por um unificador.

O propósito de MATCH é sugerir um conjunto de substituições que podem formar "segmento iniciais" de unificadores e, dentro deste processo, conduzir a busca de um unificador mais próximo à resolução. Demonstra-se que MATCH atinge este objetivo [Nada87].

Como o propósito de MATCH é obter uma substituição que torne dois termos iguais, a *imitação* sugere que uma possível substituição deve ser um termo cuja cabeça é igual àquela do termo rígido do par de discórdia. A intenção é aplicar uma substituição para f de modo a manter a cabeça rígida aplicada a dois termos t_1 e t_2 e na sequência obter o

unificador para os pares de argumentos. No exemplo abaixo, onde

F^1 : Avo Pedro (Pai y) e

F^2 : f Pedro José

o unificador para f, por imitação, é uma função que aplicada a Pedro e José resulta em

Avo t_1 t_2

Para isso é preciso que t_1 tenha o mesmo tipo de Pedro e t_2 tenha o tipo de (Pai y). Isto é p, pois Avo e f têm tipo $p \rightarrow p \rightarrow p$. Assim os termos h_1 e h_2 são do tipo $\alpha_1 \rightarrow \alpha_2 \rightarrow p$ ou seja $p \rightarrow p \rightarrow p$. O unificador por imitação é então:

f / $\lambda w_1 w_2$(avo (h_1 w_1 w_2) (h_2 w_1 w_2).

cuja substituição por f na fórmula flexível β-reduz para:

Avo (h_1 Pedro José) (h_2 Pedro José) .

Aplicando-se SIMPL ao par formado por F^1 e esta fórmula obtem-se o seguinte conjunto discórdia D:

{⟨ h_1 Pedro José , Pedro ⟩, ⟨ h_2 Pedro José , (Pai y) ⟩} .

Tomando-se o primeiro para para dar sequência à unificação e considerando outra vez o unificador gerado por IMIT, obtem-se:

$h_1/\lambda w_1 w_2$ Pedro

cuja substituição seguida da β-redução torna o conjunto de discórdia D igual a:

{⟨ Pedro Pedro ⟩, ⟨ h_2 Pedro José ⟩}

no qual o primeiro par é eliminado. Na sequência tem-se

$h_2/\lambda w_1 w_2$(Pai (h_3 w_1 w_2))

que transforma D em:

{⟨h_3 Pedro José , y ⟩}

que é um par flexível/flexível, cujo unificador $\lambda w_1 w_2.y$ é explicado na secão 4.4. A composição dos unificadores resulta em

θ = $\lambda w_1 w2$ (Avo Pedro (Pai y))

para f, que posibilita tornar $F^1 = F^2$.

No espaço, observa-se que a procura do unificador vai gerando uma árvore. Chamando o conjunto discórdia inicial de D, obtem-se a árvore da figura 4.4.

Nela u_i representa o unificador obtido por imitação e u_{p_i} representa o i-ésimo unificador obtido pela projeção. Esta estrutura é chamada de *árvore da unificação* e D_1 representa o conjunto D após a aplicação de uma das substituições u.

Os unificadores obtidos por *projeção* projetam, após substituição e simplificação, um dos argumentos da fórmula flexível como cabeça da fórmula resultante. Pretende-se com

Formação da árvore de Unificação: Seja o conjunto D de discórdia constituido de pares de fórmulas em sua forma normal de unificação. A figura abaixo esquematiza o inicio de formação da árvore de unificação.

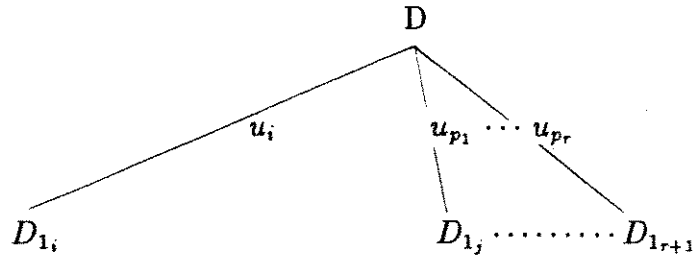


Figura 4.4: Árvore da unificação

isso que a cabeça da fórmula resultante seja igual à cabeça do argumento correspondente na fórmula rígida. Para que a unificação posterior seja possível, a variável w_i deve ter o mesmo número e tipo de argumentos do primeiro argumento da fórmula flexível, no exemplo a seguir, *Pai*. Daí a necessidade de se observar o tipo de α_i. Considere A e B duas constantes quaisquer, com de tipo p e Pai do tipo $p \rightarrow p \rightarrow p$ e as fórmulas

F^1 : Pai A B
F^2 : f Pai

onde F^2 não está em sua forma normal de unificação. Logo o par que formará o conjunto discórdia D é:

F^1 : Pai A B
F^2 : f ($\lambda x\ y$ (Pai x y))

com D igual a

$\{\langle$ Pai A B , f (λxy (Pai x y)) $\rangle\}$

O unificador por projeção é então

$\lambda w_1.w_1$ ($h_1\ w_1$) ($h_2\ w_1$)

com o tipo de $w_1 = \alpha_1 = p \rightarrow p \rightarrow p$. Portanto ($h_1\ w_1$) e ($h_2\ w_1$) têm tipo p e h_i têm tipo $p \rightarrow p$. A substituição aplicada a F^2 gera

$(\lambda w_1 w_2$ ($h_1\ w_1$) ($h_2\ w_1$)) (f ($\lambda x\ y$ (Pai x y)))

que β-reduzida é

(Pai (h_1 λxy Pai x y) (h_2 λxy Pai x y))

que então é SIMPLificável gerando os pares

$\{\langle$ (h_1 λxy Pai x y), A $\rangle$, $\langle$ (h_2 λxy Pai x y), B $\rangle\}$

A sistemática de pesquisa na árvore segue de maneira similar àquela mostrada no exemplo anterior, e no algoritmo da figura 4.1.

Dois exemplos de aplicação do algoritmo da unificação, para um conjunto discórdia com duas fórmulas, são analisados a seguir:

Exemplo 4.3.1.3 Suponha D = $\{\langle F^1, F^2\rangle\}$ =$\{\langle f(f(x)), quad(quad(\ 2)\rangle\}$, com os tipos de x e de 2 sendo *int* e o tipo de f e quad ($int \rightarrow int$). Se a função *quad* for interpretada como a função real x^2 (quadrática) então o resultado de quad(quad 2) pode ser interpretado como 16. Uma solução de unificação visível seria { f/quad, x/2 }, isto é, se esta substituição for aplicada a D, F^1 e F^2 se tornam iguais.

Aplicando-se a D o algoritmo SIMPL temos SIMPL(D) = SIMPL({ $\langle F^1, F2\rangle$}) = {$\langle$ (f (f x)),(quad(quad 2))$\rangle$ } pois F^1 é flexível e $F2$ é rígido, representando o caso 2.a do algoritmo da figura 4.2. Assim SIMPL(D) = D, isto é, SIMPL retornará o mesmo conjunto. Para rastrear o algoritmo MATCH necessita-se que as duas fórmulas estejam na forma normal de unificação (FNU). Observa-se que isto acontece com F^1 e com F^2. Para que se possa melhor acompanhar o rastreio as duas fórmulas são especificadas abaixo:

$$F^1 = \lambda\bar{x}\ (f\ A^1 \cdots A^r) = (\text{f}\ (\text{f x}))$$

com r = 1, A^1 = (f x), $\alpha_1 = int$ e $\beta = int$

$$F^2 = \lambda\bar{x}\ (C\ B^1 \cdots B^s) = (\text{quad}\ (\text{quad}\ 2))$$

com s = 1 e B^1 = (quad 2)

Por imitação obtem-se { $f/\lambda w(C\ (h^1\ w))$ ou seja $\{f/\lambda w(quad\ (h^1\ w))\}$ já que C não é variável. Aqui h^1 é do tipo ($int \rightarrow int$).

Na projeção é necessário observar atentamente o tipo dos argumentos de F^1 (a fórmula flexível). Para cada argumento, se seu tipo contiver um argumento funcional, não haverá projeção. Neste caso, o único argumento de f é (f x) que é do tipo *int*. Logo α_i é do tipo $\beta_1 \rightarrow \beta_2 \rightarrow \cdots \rightarrow \beta_t \rightarrow \beta$, com t = 0. Não haverá h^i, mas project = $\lambda w.w$.

Para unificação do primeiro par de discórdia tem-se aí dois unificadores:

$$f/\lambda w\ quad(h^1\ w) \text{ e } f/\lambda w.w$$

Tomando-se o primeiro[1] unificador e usando-o como substituição para o par de discórdia, tem-se

$$\langle\lambda w(quad(h^1\ w))(\lambda w(quad(h^1\ w))\ x, (\text{quad}\ (\text{quad}\ 2))\ \rangle,$$

que não está na forma normal e β-reduzida gera

[1]Aqui está a segunda fonte de não determinismo para a derivação

$\langle \lambda w(quad(h^1\ w))(quad(h^1\ x),$ (quad (quad 2)) $\rangle,$

que é novamente β-reduzida para

$\langle (quad(h^1\ (quad(h^1\ x)),$ (quad (quad 2)) $\rangle$

Simplificando, por meio da SIMPL, tem-se

$\langle h^1\ (quad(h^1\ x)),$ (quad 2) $\rangle.$

Aplicando-se novamente a rotina MATCH tem-se por imitação

$h^1/\lambda w_1(quad(h^2\ w_1))$

Neste caso foi necessário escolher uma variável (h_2) de um tipo que aplicada a w_{int} resultasse em algo cujo tipo fosse igual ao tipo de 2 que é (int). Então h é do tipo ($int \rightarrow int$). Portanto, não contém função como argumento. Não gera portanto qualquer h. Logo *project* gera a substituição $h^1, \lambda w.w$. Novamente tem-se dois resultados cuja substituição no conjunto discórdia permitirá continuar a pesquisa para encontrar um unificador.

Escolhendo novamente o primeiro par, resultante da imitação, e aplicando-o ao conjunto de discórdia como substituição obtem-se:

$\langle \lambda w_1(quad(h^2\ w_1))(quad\ \lambda w_1(quad(h^2\ w_1)\ x)), (quad 2) \rangle$ Prosseguindo com as β-reduções chega-se a

$\langle quad(h^2(quad(quad(h^2\ x)))), (quad\ 2) \rangle$

e finalmente, após nova aplicação de SIMPL, tem-se:

$\langle (h^2(quad(quad(h^2\ x)))), 2 \rangle$

Na sequência são obtidos os seguintes unificadores : $\lambda w.2$ por imitação e $\lambda w.w$ por projeção. Neste caso não há argumentos B em F^2, e o argumento h da projeção não existe porque o argumento de h^2, em F^2 não é funcional. Aplicando-se a primeira substituição obem-se o seguinte par de discórdia:

$\langle (\lambda w.2(quad(quad(\lambda w.2\ x)))), 2) \rangle$

que β-reduzido será

$\langle (\lambda w.2(quad(quad(2))), 2 \rangle$

$\langle (2, 2) \rangle$

o que representa um sucesso, isto é, por meio de todas as substituições escolhidas, o conjunto discórdia inicial pode ser unificado. Este fato é observado pois D é $\emptyset$. O unificador é a composição das substituições intermediárias escolhidas. Neste caso um unificador apresenta a substituição é $f/\lambda w_1$ (quad(quad 2)).

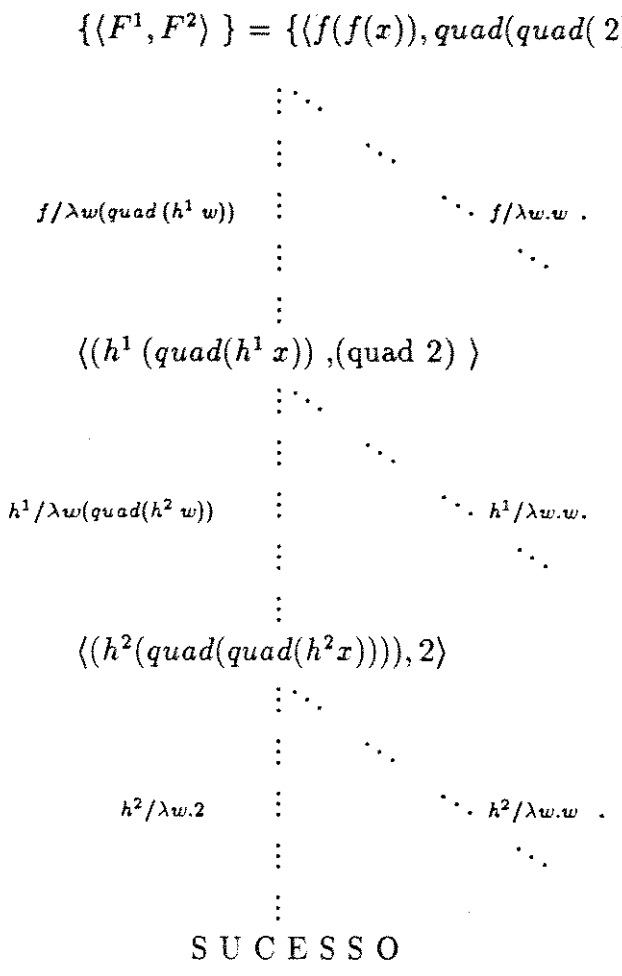


Figura 4.5: Árvore de unificação para $\{\langle f(f(x)), quad(quad(\ 2))\rangle\ \}$

O conjunto dos rótulos dos ramos da folha, da folha até à raiz fornecerá um unificador.

No caso de pares flexível-flexível é fácil encontrar pelo menos um unificador, como será visto na subseção 4.4.

Todo o processo de procura de um unificador para o par $\langle F^1, F^2\rangle$ do exemplo acima gera a árvore de derivação da figura 4.5.

O unificador que foi sugerido intuitivamente no inicio do exemplo 4.3.1.1 pode ser obtido dando prosseguimento às unificações parciais usando-se os unificadores gerados por projeção.

Exemplo 4.3.1.4 Suponha que se deseja unificar

$$\mathrm{D} = \{\langle (f_1\ x), (g\ 1\ 1)\rangle\}$$

cujos tipos são os seguintes:

- tipo de g: $(int \rightarrow int \rightarrow int)$,
- tipo de f_1: $(int \rightarrow int)$ e
- tipo de x: int.

Aplicando MATCH a D tem-se por imitação, o seguinte unificador

$$f_1/\lambda w.g(h^1\ w)(h^2\ w)$$

porque g requer 2 argumentos. Como w é do tipo int e 1 é do tipo int, h^1 e h^2 serão do tipo $int \rightarrow int$. Por projeção será obtido

$$f_1/\lambda w.w$$

uma vez que α_1 é int e não funcional. Tem-se portanto duas substituições para iniciar a unificação de D.

4.3.2 Resolução ou P-derivação

Esta seção tem o objetivo de esclarecer e formalizar a ligação entre a unificação e o processo de se provar alguma consulta partindo de uma base de conhecimento. A consulta fornece o primeiro elemento do conjunto de discórdia inicial. Ele é uma das fórmulas da consulta, escolhida por algum critério heurístico. O segundo elemento do primeiro conjunto discórdia para a prova constitui-se do átomo A de alguma sentença da base de conhecimento. O processo de prova como um todo é chamado de P-derivação (porque a base de conhecimento é tambem chamada de programa P).

A noção formal da P-derivação é enunciada a seguir. Ela baseia-se exatamente no controle das variáveis livres, da fórmula objetivo e do conjunto de unificadores quando é executado algum dos "passos" para evoluir no processo: escolha e diminuição da fórmula objetivo, encadeamento e unificação.

Sejam os símbolos O, D, θ e V, com ou sem subscritos, as notações para conjuntos de fórmulas bem formadas do tipo t, conjunto de discórdia, substituições e conjuntos de variáveis, respectivamente. Define-se a seguinte relação de derivação, cujo nome é *P-Derivada de* entre tuplas da forma

$$\langle\ \mathrm{O},\ \mathrm{D},\ \theta,\ \mathrm{V}\ \rangle,$$

que é básica para a definição de uma P-derivação:

Seja P um conjunto de sentenças definidas. Diz-se que a tupla

$$\langle O_2, D_2, \theta_2, V_2 \rangle$$

é P-derivada de

$$\langle O_1, D_1, \theta_1, V_1 \rangle$$

se $D_1 \neq \mathbf{F}$ (de falso) e em adição, uma das seguintes situações se mantiver:

i) *(Passo de redução da fórmula objetivo)* $\theta_2 = \emptyset$ (vazio), $D_2 = D_1$ e existe uma fórmula objetivo $G \in O_1$, tal que tal que

   a) G é $\top$ e $O_2 = O_1 - G$ e $V_2 = V_1$ ou
   b) G é $G^1 \wedge G^2$ e $O_2 = (O_1 - G) \cup \{G^1, G^2\}$ e $V_2 = V_1$ ou
   c) G é $G^1 \vee G^2$ e para i = 1 ou i = 2, $O_2 = (O_1 - G) \cup \{G^i\}$ e $V_1 = V_2$ ou
   d) G é $\exists P$ e para alguma variável $y \notin V_1$ ocorre que $V_2 = V_1 \cup \{y\}$ e $O_2 = (O_1 - \{G\}) \cup \{\lambda norm(P\, y)\}$.

ii) *(Passo de Encadeamento)* Seja $G \in O_1$ um átomo positivo rígido e seja S $\in P$ tal que S $= \forall x^1. \cdots \forall x^n.G' \supset A$ para alguma sequência de variáveis $x^1, \cdots, x^n$ para as quais nenhum $x^i \in V_1$ (Neste caso S é uma sentença da base de conhecimento P). Então $\theta_2 = \emptyset, V_2 = V_1 \cup \{x^1, \cdots, x^n\}, O_2 = (O_1\{G\}) \cup \{G'\}$ e $D_2 = SIMPL(D_1 \cup \{\langle G, A\rangle\})$.

iii) *(Passo de unificação)* D_1 não é um conjunto resolvido e para algum par flexível-rígido $(F^1, F^2) \in D_1$ ou MATCH $(F^1, F^2, V_1) = \emptyset$ e $D_2 = \mathbf{F}$, ou existe um $\sigma \in$ MATCH (F^1, F^2, V_1) e neste caso $\theta_2 = \sigma, O_2 = \sigma(O_1), D_2 =$ SIMPL$(\sigma(D_1))$, e se $\sigma = \{\langle x, T\rangle\}$, $V_2 = V_1 \cup VL(T)$, onde VL(T) representa as variáveis livres de T.

No *passo de quebra ou redução da fórmula objetivo* o item (a) indica que $\top$ (um dos possíveis átomos da consulta) pode ser eliminado da fórmula . Nos itens (b) e (c) observa-se como fórmulas objetivo com os conectivos $\wedge$ e $\vee$ e podem ser simplificadas por meio do desmembramento de seus termos constituintes. No item (d) mostra-se a transformação de uma fórmula pertencente ao conjunto de fórmulas objetivo que contenha o quantificador existencial.

O *passo de encadeamento* consiste na escolha de uma regra ou fato (ou seja, de uma sentença) da base de conhecimento com possibilidade de unificação. À fórmula objetivo é incorporada a parte condicional da regra, se houver, e o conjunto de discórdia é acrescido do resultado da rotina SIMPL aplicada a parte esquerda da regra escolhida na base e a fórmula escolhida na consulta, com apropriado controle das variáveis. Neste passo, ainda não ocorre unificação. Só há a eleição de uma sentença da base, o acréscimo da parte condicional da sentença escolhida ao conjunto objetivo e o estabelecimento de um novo conjunto discórdia.

O *passo da unificação* consiste na aplicação do procedimento da unificação, já descrito, ao conjunto discórdia do passso anterior. No entanto observa-se a necessidade de, uma vez encontrada uma substituição resposta, aplicar a substituição ao conjunto restante de fórmulas objetivo e controlar as variáveis livres.

Seja O um conjunto de fórmulas objetivo. Diz-se que a sequência

$$\langle O_i, D_i, \theta_i, V_i\rangle, \{1 \leq i \leq n\}$$

é uma sequência *P-derivada para* O exatamente quando

- $O_1 = O$,
- $V_1 = VL(O_1)$, onde VL representa o conjunto de variáveis livres,
- $D_1 = \emptyset$, $\theta_1, = \emptyset$ e,
- $\langle O_{i+1}, D_{i+1}, \theta_{i+1}, V_{i+1}\rangle$ é P-derivada de $\langle O_i, D_i, \theta_i, V_i\rangle$, $1 \leq i \leq n$, por meio de um dos passos definidos acima.

Esta definição estabelece que a P-derivação se inicia tomando-se o conjunto objetivo como sendo O_1. Suas variáveis livres são colocadas em V_1 e os demais elementos da tupla inicial sendo vazios. A tupla assim formada fica sujeita aos passos acima especificados para que se possa progredir no processo.

Uma sequência de P-derivações

$$\langle O_i, D_i, \theta_i, V_i\rangle, \{1 \leq i \leq n\}$$

termina, isto é, não está contida numa sequência mais longa se,

- O_n é um conjunto objetivo vazio ou consiste apenas de átomos flexíveis e D_n é vazio ou contém apenas pares flexível-flexível, ou
- $D_n = \mathbf{F}$.

No primeiro caso diz-se que ela é uma sequência *terminada com sucesso*. A visão global do algoritmo de P-Derivação é fornecida na figura 4.5.

A escolha dos passos a executar, quando houver alternativas, prioriza a unificação.

Observa-se que pode ter restado no conjunto D algum par flexível-flexível e/ou o conjunto de fórmulas objetivo não ser vazio. Estes dois casos são abordados na seção seguinte, onde é visto como são trabalhados.

Quando $D_n = \mathbf{F}$, não houve sucesso na P-derivação.

Uma sequência de P-derivações $\langle O_i, D_i, \theta_i, V_i\rangle$, $\{\leq i \leq n\}$, para O que termina com sucesso é chamada de P-*derivação* de O e a composição de todos os unificadores obtidos em cada passo da unificação definido anteriormente gera a substituição resposta θ. Assim

$$\theta = \theta_n \circ \theta_{n-1} \circ \cdots \circ \theta_1$$

é chamada de substituição resposta. Nela o símbolo $\circ$ representa a composição de substituições.

Exemplo 4.3.2. Seja o seguinte conjunto de sentenças definidas uma base de conhecimento P.

$$\top \supset (\text{mapfun } f_1 \text{ nil nil})$$
$$(\text{mapfun } f_1 \ l_1 \ l_2\ ) \supset$$
$$(\text{mapfun } f_1 \ (\text{cons } x \ l_1) \ (\text{cons } (\text{f } x) \ l_2))$$

Seja O uma consulta e P uma base de conhecimento (programa) e D o conjunto discórdia, como definidos anteriormente

Enquanto não ocorrer um dos seguintes ítens

- O_n é um conjunto objetivo vazio ou consiste apenas de átomos flexíveis e D_n é vazio ou contém apenas pares flexível - flexível, ou
- $D_n = \mathbf{F}$.

escolha e execute uma das seguintes alternativas:

1. Passo de redução da fórmula objetivo
2. Passo de Encadeamento
3. Passo de unificação

Figura 4.6: Macro-definição do Algoritmo da P-derivação

Além disto sejam f_1 uma variável do tipo $int \rightarrow int$ e G a seguinte fórmula objetivo

(mapfun f_2 (cons 1 (cons 2 nil))
(cons (g 1 1) (cons (g 1 2) nil)))

em que a única variável livre é f_2. Então a tupla $\langle O_1, D_1, \theta_1, V_1 \rangle$ é P-derivada de $\langle \{G\}, \emptyset, \emptyset, \{f_2\} \rangle$ por um passo de encadeamento se

$$V_i = \{f_1, f_2, l_1, l_2, x\} \qquad \text{(variáveis livres)}$$
$$O_1 = \{(mapfun\ f_2\ l_1\ l_2)\}$$
$$D_1 = \{\langle f_1, f_2\rangle, \langle (f_1\ x), (g\ 1\ 1)\rangle, \langle l_1, (cons\ 2\ nil)\rangle, \langle l_2, (cons\ (g\ 1\ 2)\ nil)\rangle\}$$

em que f_2, l_1, l_2 e x são variáveis. De modo semelhante, se

$$V_2 = V_1 \cup \{h_1, h_2\}$$
$$O_2 = \{(mapfun\ f_2,\ l_1,\ l_2)\}$$
$$\theta_2 = \{\langle f_1, \lambda w.(g(h_1\ w)(h_2\ w))\rangle\}$$
$$D_2 = \{\langle l_1, (cons\ 2\ nil)\rangle, \langle l_2, (cons\ (g\ 1\ 2)\ nil\rangle \langle x, 1\rangle, \langle (h_1\ x), 1\rangle, \langle (h_2\ x), 1\rangle, \}$$

$$\langle f_2, \lambda w.(g(h_1\ w)(h_2\ w))\rangle\}$$

então a tupla $\langle O_2, D_2, \theta_2, V_2\rangle$ é P-derivada de $\langle O_1, D_1, \theta_1, V_1\rangle$ por um passo de unificação ao se tomar o par flexível -rígido $\langle (f_1, x), (g11)$ de D_1 e usar uma das substituições obtida por MATCH.

4.4 Pares Flexível-flexível e Fórmulas Obj. Flexíveis

Nota-se que que ao final de uma derivação terminada com sucesso o conjunto de fórmulas objetivo pode conter átomos flexíveis, assim como o conjunto discórdia que pode apresentar pares flexível-flexível.

A correção da P-derivação não é afetada pela escolha de certo tipo de susbstituição para estes casos ([Huet75],[Nada90]). No caso do átomo flexível, basta escolher uma função "constante" cuja cabeça é $\top$ e cujo binder seja adequado para o tipo da variável que está na cabeça do átomo em questão. Assim, por exemplo:

Átomo	Substituição
x	$\top$
(x a)	λy.$\top$

e de modo geral, a substituição é

$$\lambda x_{\beta_1} \cdots \lambda x_{\beta_k}.\top$$

No caso dos pares flexível-flexivel do conjunto de discórdia D as funções podem ser de um tipo que não seja proposiçional. Neste caso a substituição será

$$\lambda x_{\beta_1} \cdots \lambda x_{\beta_k}.y_\beta$$

onde y_β é uma variável qualquer do mesmo tipo da variável flexível e os β_i são os tipos de seus argumentos. Neste caso de pares flexíveis basta então escolher a mesma variável.

Exemplo 4.4.1 Suponha que a base de conhecimento contenha apenas a seguinte sentença:

$$\forall x(x \supset Pai\ Pedro)$$

e que se deseja responder à consulta:

$$\exists y(Pai\ y)$$

A sequência de derivações é a seguinte:

$$\langle\{\exists y(Pai\ y)\}, \emptyset, \emptyset, \emptyset\rangle$$

$$\langle\{(Pai\ y)\},\emptyset,\emptyset,\{y\}\rangle$$

$$\langle\{x\},\{\langle y, Pedro\rangle\},\emptyset,\{y,x\}\rangle$$

$$\langle\{x\},\emptyset,\{\langle y, Pedro\rangle\},\{y,x\}\rangle$$

e a substituição final de x por $\top$ atende ao questionamento feito pela consulta.

Exemplo 4.4.2 Suponha que se tenha o seguinte par de discórdia que se deseja unificar, e que f e g tem tipo $int \rightarrow int \rightarrow int$,

$$\langle(f\ 1\ x),(g\ 3\ 2)\rangle$$

em que se procura uma função f que aplicada a 1 e x resulte no mesmo valor que g aplicada a 3 e 2. O universo da solução, embora desconhecido, supõe-se numeroso ou infinito. No entanto ao assumir que tanto g quanto f são a função

$$\lambda z_1 z_2 . y$$

tem-se um unificador para o par. Deve-se notar que se ao invés de y fosse escolhida a constante 6 (de tipo apropriado para g e f) o unificador também estaria correto.

Esta opção de escolha do unificador sacrifica a completude da P-derivação. No entanto não introduz consequências danosas ao resultado obtido.

Quando uma fórmula flexível pertence ao conjunto objetivo, o sistema da P-derivação fica com o mecanismo de procura na base de conhecimento sem efeito, uma vez que ele é orientado pela busca de um átomo rígido que coincide com a cabeça da fórmula objetivo. Intuitivamente pretende-se que uma determinada variável predicativa tenha uma substituição que quando aplicada por determinados parâmetros resulte numa proposição verdadeira. Por exemplo se (f A x) for esta proposição, a P-derivação pretende uma substituição para f que a torne verdadeira. Se f for substituida por $\top$ ou uma proposição cuja cabeça é $\top$, como foi descrito acima, pode-se eliminar a fórmula do conjunto objetivo. Como consequência deve-se aplicar a mesma substituição ao restante do conjunto objetivo O e ao conjunto D para prosseguir na P-derivação.

O procedimento acima pode ser acrescentado formalmente como um dos passos que compõe a P-derivação, fornecidos na seção anterior, da seguinte maneira:

- *Eliminação de fórmulas flexiveis* : Se G pertence a O_1, com tipo $\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta$ para sua variavel da cabeça, tome $\theta_2 = \{(\lambda x_1 \cdots \lambda x_n . \top)\}$, $O_2 = \theta_2(O_1$ - G) e $D_2 =$ SIMPL$(\theta_2(D_1))$

Como esta pode não ser a substituição ideal, já que o algoritmo SIMPL pode emparelhá-las e unificá-las em outras condições e posteriormente instanciá-las, o interpretador deve protelar esta eliminação até que as demais fórmulas objetivo, não flexíveis tenham sido eliminadas do conjunto.

Capítulo 5

O Interpretador λ-Prolog

5.1 Introdução

Nos capítulos anteriores foi descrita uma linguagem que suporta a descrição do conhecimento, num paradigma que permite maior liberdade de expressão do que aquela da lógica de primeira ordem ao incorporar a lógica de ordem superior. A linguagem usa os princípios do λ-cálculo e da teoria dos tipos. O conhecimento assim representado fica organizado em uma hierarquia proveniente das diferentes relações de complexidade existentes entre as fórmulas que foram livremente formadas.

A linguagem, restrita ao subconjunto de fórmulas definidas de ordem superior, descritas no capítulo 4, apresenta uma aproximação com o modelo básico da linguagem da programação lógica de primeira ordem. Este, ao restringir suas sentenças às cláusulas de Horn, permite que uma base de conhecimento formada com estas cláusulas seja interpretada como um conjunto de procedimentos, o que favorece sua implementação.

Ao restringir a representação às fórmulas definidas de ordem superior manteve-se para a LOS algumas características daquele paradigma, principalmente a interpretação computacional, tornando-a uma generalização daquele modelo. Mantém-se a interpretação de uma consulta como sendo uma coleção de chamadas de procedimentos, cujas cabeças representam o nome e seus argumentos os parâmetros *atuais*. Cada sentença da base é um procedimento com nome igual ao da cabeça do predicado que aparece à esquerda. Os argumentos deste predicado representam os parâmetros *formais*. As fórmulas da direita são o corpo do procedimento e representam sub-chamadas ou subproblemas. Sob esta visão uma coleção de cláusulas é um programa.

A seguir, apresenta-se uma proposta para a implementação do interpretador λ-PROLOG que mecaniza a metodologia de provas para a linguagem da lógica de ordem superior.

O sistema como um todo divide-se em três módulos principais, distribuidos como na figura 5.1 :

- Inicialização

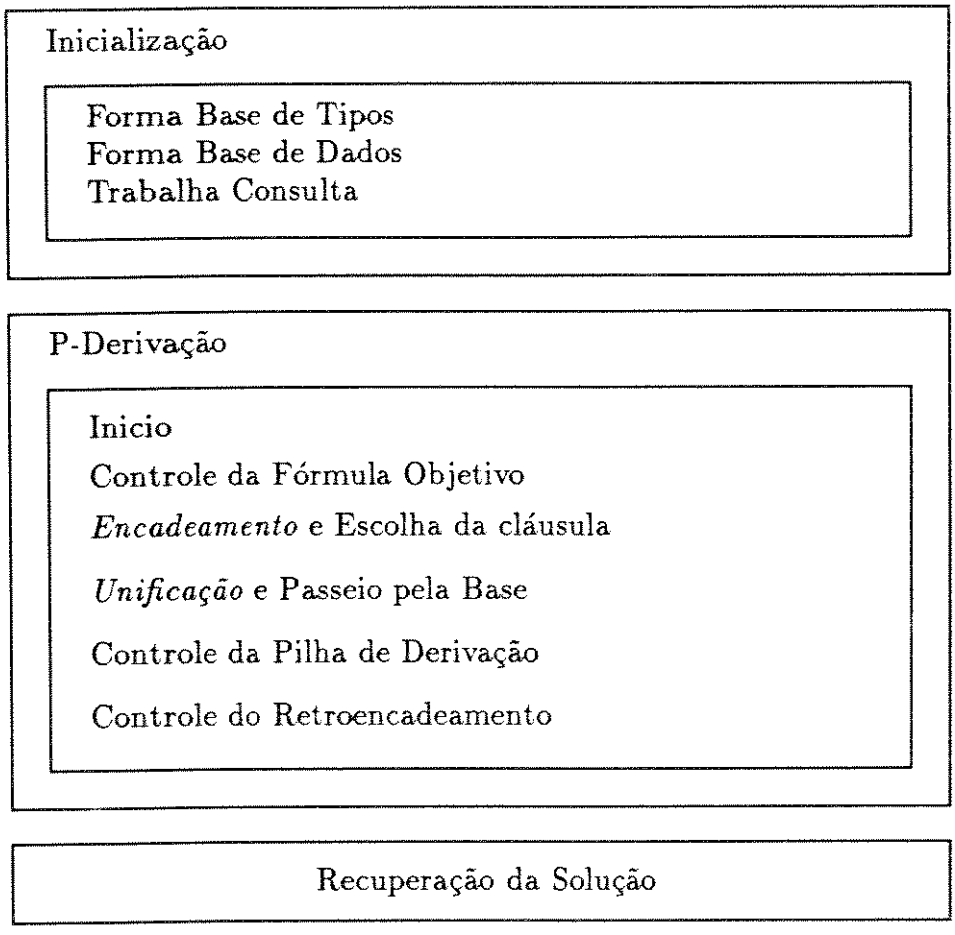


Figura 5.1: Módulos principais do sistema de provas

- Execução da P-Derivação
- Recuperação da resposta.

No paradigma da programação em lógica um programa é um conjunto de descrições de predicados. Não existe a diferenciação entre um conjunto de instruções a serem executadas e os dados sobre os quais as ações são executadas. A condição inicial para se desenvolver um mecanismo de prova é constituir o conjunto de descrições que também é chamado de base de conhecimento ou programa, na visão procedural. Na lógica de ordem superior esta tarefa requer ainda que sejam informados os tipos das variáveis e constantes que serão usadas para formar as sentenças que comporão a base.

O primeiro módulo do protótipo controla e gerencia as seguintes atividades ligada à formação da base de conhecimento:

- Declaração de tipos das variáveis e constantes utilizadas na base
- Correção gramatical, adequação de tipos e arquivamento das sentenças que farão parte da base ou programa
- Correção gramatical, adequação de tipos e arquivamento das fórmulas que constituem a consulta

Construida a base de conhecimento e de tipos, bem como as estruturas que permitem acessá-las, o sistema carece da programação do mecanismo de inferência que torne possível responder consultas feitas sobre a base. Formalmente este mecanismo é a P-derivação definida no capítulo 4 e mostrada como algoritmo na figura 4.6. O segundo módulo da figura 5.1 mostra sua implementação. As seguintes tarefas foram caracterizadas como sub-módulos importantes desta etapa:

- Início
- Controle da fórmula objetivo
- Encadeamento e escolha da cláusula
- Unificação
- Controle da Pilha de Derivação
- Controle do retroencadeamento

Como no paradigma de programação lógica não há atribuição de valores a variáveis e os valores são apenas ligados às variáveis. O mecanismo de prova necessita percorrer de volta todos os assinalamentos (ligações) feitos para as variáveis livres que fizeram parte da derivação afim de recuperar em que instâncias foi permitido concluir a consulta. A recuperação da resposta ou solução torna-se, porisso, um mecanismo não trivial do ponto de vista computacional e merecedor de um módulo especial, que é o terceiro.

Este capítulo descreve um protótipo da implementação do sistema acima descrito. Detalhamento parcial de cada módulo é feito de modo a capacitar o entendimento e possibilitar refinamentos posteriores.

Como não se encontram, na literatura, referências sobre a metodologia de implementação de sistemas baseados em LOS procurou-se orientar pelas implementações do PROLOG padrão que se assentam sobre a lógica de primeira ordem. Apenas mais recentemente [Nada89] contempla alguns aspectos da implementação de um interpretador λ-Prolog, buscando a otimização do sistema. No entanto a visão geral não é oferecida. A intenção, neste trabalho, foi implementar e oferecer o roteiro da implementação, possibilitando um estudo inicial de todos os aspectos que estão envolvidos nesta tarefa.

Seguindo uma orientação "top-down" cada item dos módulos citados na figura 5.1 é descrito e algumas propostas são lançadas para atender às situações que neles ocorrem.

Assim é que na seção 5.2 são descritos e analisados alguns aspectos ligados a inicialização como a formação das bases de conhecimento e de tipos e detalhes sobre a gramática a que estão sujeitas as cláusulas e que possibilitam a formatação correta da base de conhecimento. Nesta secção pode ser visto como a análise sintática oferece meios de conferir semanticamente os tipos das variáveis e das constantes das fórmulas.

Na secção 5.3 é mostrada a implementação da P-Derivação com ênfase especial para a pilha de derivação, que possibilita o encadeamento por meio de apontadores, dispensando concatenação de cadeias e elimina a necessidade de $\beta-$reduções em caso de retroencadeamento ou soluções múltiplas. Descreve-se a visão computacional de toda a P-Derivação e são destacados os mecanismos de controle de sua execução, do início até o retroencadeamento.

Numa secção à parte (5.4) é mostrada a implementação da unificação e seu pseudocódigo. A descrição é enriquecida com exemplos de sua execução.

Na seção 5.5 apresenta-se uma análise geral da implementação.

5.2 A Base de Conhecimento e de Tipos

Um interpretador λ-Prolog requer a existência da base de conhecimento que contem os fatos e regras sobre os quais se deseja trabalhar. Codificada segundo a linguagem da lógica de ordem superior, conforme uma sintaxe pré-estabelecida pelo implementador, ela é, em última instância, o programa lógico cuja execução se pretende com o interpretador.

A figura 5.2 contem um diagrama que descreve o fluxo das cláusulas e dos tipos quando é feita uma declaração que determinado tipo de variável ou constante, ou cláusula farão parte da base de conhecimento ou base de tipos. O usuário fornece uma das duas entradas ou, em particular uma consulta, que passam por um analisador sintático e após verificação da correção, são armazenados nas respectivas bases e apontadores são estabelecidos para facilitar posterior recuperação. Ao se tratar de uma consulta o procedimento se diferencia porque assinalamentos devem ser efetuados para que se possa disparar a derivação.

Para se construir a base de conhecimento, o sistema deve oferecer ao usuário uma gramática segundo a qual as sentenças possam ser descritas de modo único e sem ambiguidades. Ela estabelece a sintaxe da linguagem. O interpretador, antes de inserir qualquer cláusula na base executa um teste sintático (correção gramatical) e da semântica dos tipos.

Qualquer constante ou variável do conjunto elementar deve ser tipificada e pode ser usada livremente na formação das sentenças. Variáveis e constantes não pertencentes a este conjunto devem tambem ser tipificadas. No entanto seus tipos podem ser inferidos pelo sistema ou fornecidos pelo usuário.

A inferência de tipo pode ser melhor entendida com o seguinte exemplo: Seja a seguinte cláusula:

(raiz (f x) 4)

Se os tipos de f e *raiz* não foram explicitamente declarados e a fórmula estiver na forma

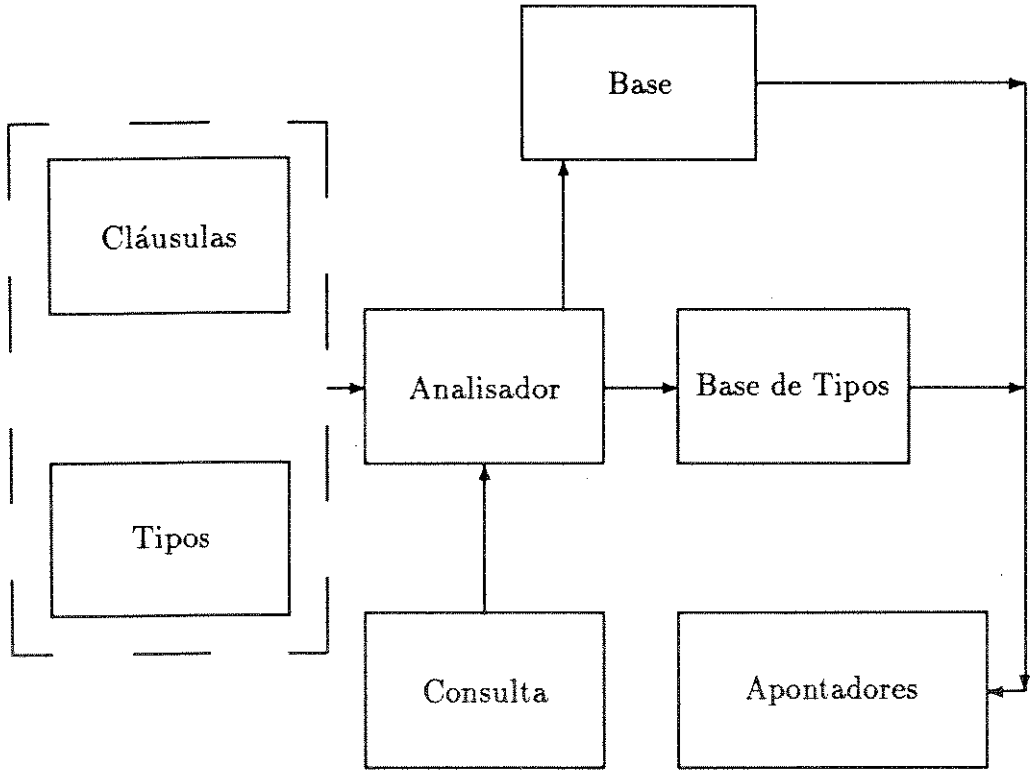


Figura 5.2: Estrutura básica da inicialização

normal de unificação, e sendo x uma variável de tipo elementar declarado, então o tipo de f está semi-definido e uma pequena consulta ao usuário poderá torná-lo conhecido. Assim, o tipo de f é

$$tipo_x \rightarrow \beta$$

em que o tipo β deve ser informado. No caso do tipo de *raiz*, β já é conhecido como sendo t (pois a sentença é uma proposição).

No processo de inferência, o sistema pode solicitar confirmação e auxílio. Nosso protótipo considera apenas a segunda opção de fornecimento dos tipos, desde que a inferência exige que as sentenças sejam oferecidas na forma normal de unificação e esta não é uma exigência em nosso sistema.

Há a necessidade, portanto, de uma base tipos onde cada constante ou variável usada na expressão do conhecimento tenha seu tipo explicitamente armazenado.

A inicialização do sistema de provas constitui-se de um módulo cujo núcleo contempla as três seguintes tarefas:

- Conferência gramatical
- Formação da base de conhecimento
- Formação da base de tipos

que são análisadas a seguir.

5.2.1 Gramática Para as Sentenças

A gramática da linguagem estabelece a sintaxe. A definição de alguns detalhes da sintaxe fica a cargo do implementador. A escolha dos simbolos delimitadores, a maneira de diferenciar constantes e variáveis, a escolha dos símbolos que representam os conectivos, por exemplo, são arbitradas pelo implementador. Em nosso protótipo as variáveis são sempre representadas por cadeias iniciadas por letra maiúscula, enquanto as constantes não lógicas são representadas por cadeias iniciadas por letras minúsculas. Para possibilitar uma melhor visualização, permitiu-se que os conectivos $\vee$ e $\wedge$ fossem usados infixadamente e que fossem substituidos respectivamente pelos caracteres , e ;. A constante $\exists$ foi substituida por Existe e o símbolo T foi reservado para representar a constante lógica $\top$. O simbolo λ foi substituido por \ (barra invertida). Os parênteses são usados para forçar a indicação de um termo e para indicar o escopo das abstrações.

A gramática proposta com as definições formais de termos e fórmulas bem formadas da lógica de ordem superior e que inclui os detalhes acima expostos encontra-se na figura 5.2.

Sua apresentação está ajustada à implementação na linguagem SNOBOL. Por se tratar de um protótipo, esta linguagem foi usada por sua grande potencialidade em comparar padrões. Esta particularidade facilitou, mais do que se conseguiria com outras linguagens de programação, em procedimentos que trabalham com manipulação sobre cadeias

```
LetraMaiusc     = 'A' | 'B' | ... ... | 'Z'
LetraMinus      = 'a' | 'b' | ... ... | 'z'
Digitos         = '0' | '1' | ... ... | '9'
LetraOuDigito   = LetraMinusc | Digitos
Cadeia          = LetraOuDigito
Var             = LetraMaiusc Cadeia
Conect          = ' , ' | ' ; '
Cons            = LetraMinus Cadeia
formula         = Cons | Var |
                  '\' Var '(' formula ')' |
                  formula ' ' formula |
                  '(' formula ')' Conect '(' formula ')' |
                  'Existe \' Var '(' formula ')' |
                  '(' formula ')'
AtomoRig        = Cons ' ' formula
Fato            = AtomoRig
Regra           = AtomoRig ' :- ' formula
Clausula        = Fato | Regra
```

Figura 5.3: Gramática para as cláusulas definidas

de caracteres, com comparações, substituições, contagem e outras funções típicas como balanceamento de parenteses, quebras, pesquisas de sub-cadeias, etc. Por conter grande quantidade de funções que são apropriadas para estas tarefas e por isso demandar menos tempo de programação, a linguagem SNOBOL, apresentou-se como candidata favorita no desenvolvimento do protótipo cujo desenvolvimento solicitou com frequência estas atividades.

A construção de um analisador para a gramática acima feita na linguagem SNOBOL é direta, isto é, as variáveis que precedem o sinal de "=" são definições de "patterns" e o algoritmo analisador é um comparador de padrões. A comparação do objeto com o padrão é feita forçando-se a tomada de todo o objeto em primeiro lugar. Desta forma o padrão *Regra* é testado para conter o sinal ":-" com um *AtomoRig* à esquerda e uma *formula* à direita.

Os oito primeiros padrões (de *LetraMaiusc* ··· *Cons*) são usados para satisfazer a definição de variavel (*Var*) de constante (*Cons*). Neles o símbolo | representa a conjunção "ou" e uma sequência de variáveis e constantes representa uma concatenação.

As definições dos demais padrões (*formula* ··· *Clausula*) seguem as definições para fórmulas, átomos rígidos, fatos, regras e cláusulas de ordem superior, encontradas no capítulo 3, e restritas ao subconjunto de cláusulas definidas positivas no capítulo 4.

O processamento do teste de sintaxe de uma cláusula, segundo esta gramática, gera uma sequência de derivação cuja árvore de derivação pode ser usada para se observar o entendimento que o interpretador teve da cláusula. A figura 5.3 mostra a árvore de derivação para a cláusula

rf \X(\Y(Existe \Z((S X Z) , (V Z Y)))) :- (rp S) , (rp V) .

A figura se presta, por exemplo, à observação de que *rf* foi aplicado a todo o ramo colocado à sua direita. A ramo à sua direita deve ser portanto de um tipo adequado para a aplicação de *rf*. Todas a regras usadas na formação de termos e fórmulas usadas naquela cláusula, cuja sintaxe está sendo testada, ficam descritas na árvore. A recuperação deste percurso gera uma estrutura que pode ser explorada para aferição e/ou tipificação das constantes, variáveis , termos e fórmulas da cláusula.

5.2.2 Base de Tipos

A base de tipos será consultada para verificar a correção da semântica de tipos de cada fórmula colocada na base de conhecimento e, para atender a vários requisitos da P-derivação durante o processo de unificação, quando novas variáveis podem ser geradas.

De modo semelhante, como os algoritmos da unificação trabalham com a fórmula na forma normal de unificação (FNU), e em muitos casos sua forma não é esta, há a necessidade de transformação. Este processo utiliza a η-expansão e é feito por meio da geração de novas variáveis cujos tipos deverão estar apropriados ao termo que é objeto da η-expansão. No caso de geração de novas variáveis é necessário registrar a criação e o tipo das mesmas, de modo que consultas possam ser feitas sobre elas.

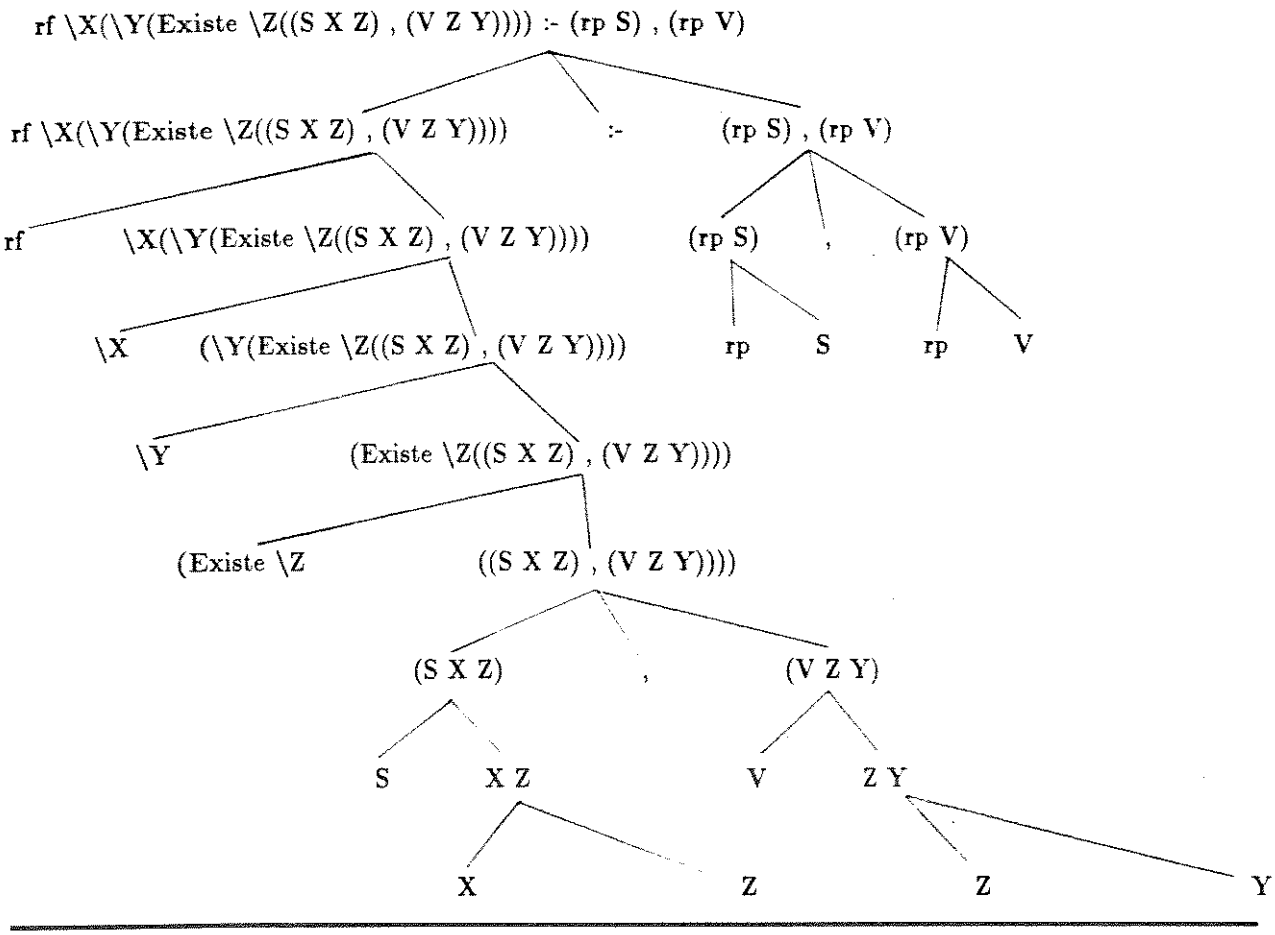


Figura 5.4: Árvore de derivação de uma cláusula

A base de tipos contem em cada registro de dois campos o nome da constante ou variável seguido de uma codificação para seu tipo. A codificação em nosso protótipo é uma concatenação de cadeias onde cada cadeia identifica um tipo elementar formador do tipo expresso. O aninhamento de tipos é feito por meio de parênteses e estes fornecem a ordem da fórmula que foi definida na seção 3.2. Uma sequência de exemplos é fornecida abaixo:

Variável ou Constante	Tipo	Registro
L1	(list int)	L1,li
X1	int	X1,i
F1	int → int	F1,ii
cons	int → (list int) → (list int)	cons,ilili
mae	e → e → t	mae,eet
mapfun	(int → int) → (list int) → (list int) →	mapfun,(ii)lilit

Uma fórmula ao ser fornecida pode estar sintaticamente correta. A verificação da semântica de tipos pode solicitar a tipificação de constantes e de variáveis ainda não tipificadas e ao mesmo tempo, conferir a adequação, isto é, observar se a aplicação ou a abstração estão obedecendo a tipificação ao serem executadas.

De posse de uma cláusula e com acesso à base de tipos, tem-se os elementos necessários para aferir a semântica de tipos. Ao conferir a sintaxe forma-se a árvore de derivação que permite formar uma estrutura que possibilita verificar a adequação dos tipos. Cabem as seguintes conferências:

- Adequação de tipos de todas as aplicações.
- Adequação de tipos para átomos rígidos e fórmulas objetivo.

O primeiro item diz respeito a que se f tem tipo $int \rightarrow int$ o argumento y na fórmula $f\ y$ deve ter tipo int. No segundo item, observa-se basicamente se os tipos caracterizam proposições.

Na figura 5.3 observa-se que as variáveis e constantes que precisam ser tipificadas estão nas folhas da árvore cuja fronteira forma a fórmula em análise. O percurso "botton-up" mostra, inicialmente, os pares X,Z e Z,Y que devem ser ou estar tipificados. Como são argumentos, pois não são precedidos do símbolo "(" nada se pode inferir sobre seus tipos.

No nível inferior da árvore, dados X, Y e Z tipificados, pode-se inferir sobre V e S. Além da premissa de se conhecer os tipos de X, Y e Z sabe-se também que S e V são proposições pois estão unidos por "," (e) e porisso seu tipo termina com $\cdots \rightarrow t$. Este percurso permite pois aproveitar a estrutura da árvore de derivação para análise da semântica de tipos. Esgotar as possibilidades não foi nosso objetivo neste trabalho. A implementação desta sistemática foi feita apenas parcialmente.

5.2.3 Base de Conhecimento

Ao final dos procedimentos de conferência sintática e de tipos, a cláusula é colocada na base de conhecimento. Quando a fórmula objetivo G da cláusula A :- G é T, tem-se um *fato*. Nos outros casos, diz-se que ela é uma *regra*. A consulta é um caso especial de cláusula definida em que o átomo A é T.

Os procedimentos que realizam esta tarefa podem ser vistos como elementos constituintes da inicialização do processo de P-derivação. Para facilitar o trabalho posterior de consulta à base, alguns procedimentos executados durante a inicialização geram tabelas de apontadores que contêm as posições dentro do arquivo de cláusulas onde se encontra cada cláusula (procedimento), identificada pela cabeça do átomo rígido. Esta indexaçào evita que toda a base de conhecimento seja pesquisada a cada passo de unificação.

O formato de uma base de dados usada para testes pode ser visto apêndice A.

A consulta é uma cláusula especial da forma T :- G que requer tambem a verificação da sintaxe e da tipificação de seus termos. Ela dispara o processo da P-derivação e é colocada como última cláusula da base de conhecimento.

5.3 A P-Derivação

A derivação ou P-derivação em lógica de ordem superior constitui-se da execução de tarefas que possibilitem obter a sequência de tuplas P-derivadas por meio de um dos passos definidos na secção 4.3 e expostos em forma algoritmica na figura 4.6. Em relação à figura 5.1, ela representa o segundo módulo do sistema de provas.

O módulo constitui-se dos sub-módulos listados na figura 5.1 que são:

- Inicio
- Controle da Fórmula Objetivo
- Encadeamento e Escolha da cláusula
- Unificação e Passeio pela Base
- Controle da Pilha de Derivação
- Controle do Retroencadeamento

Para a execução desta tarefa, pressupõe-se a existência de um conjunto de fórmulas objetivo que constituem a consulta e de uma base de conhecimento e de tipos construidas conforme foi exposto nas seções iniciais deste capítulo.

O módulo da P-Derivação atende às três seguintes tarefas principais: unificação, encadeamento e controle da fórmula objetivo. Duas tarefas que fazem parte do controle geral

da P-Derivação completam a descrição do módulo: o controle da pilha de derivação e o controle do retroencadeamento.

A redução da fórmula objetivo é um conjunto de procedimentos feitos para controlar as diversas formas que o sistema pode atuar sobre o conjunto de fórmulas objetivo, transformando-a no caso dos conectivos $\vee$ e $\wedge$ e do quantificador $\exists$, elegendo a prioridade das *fórmulas que serão atendidas*.

O passo da unificação é a aplicação do algoritmo explicado na seção 4.2 com as devidas atualizações das variáveis livres e verificação do possível insucesso do processo. Sua implementação é descrita na secção 5.4.

Já o encadeamento é o passo que merece a maior atenção no algoritmo, pois é nele que se processa a escolha das sentenças da base a serem unificadas, a atualização do conjunto objetivo e das variáveis livres, além do controle do retroencadeamento. Este passo, conforme foi descrito na seção 4.3, sugere que a fórmula objetivo ou conjunto objetivo seja acrescida, a cada passo, de novas fórmulas. Na verdade, esta é apenas uma imagem da operação de encadeamento de novas regras ao conjunto objetivo. No processamento, o encadeamento destas novas regras é feito por meio de apontadores que formam uma lista encadeada como será visto na seção 5.3.3.

Todo os procedimentos deste módulo foram projetados de forma a permitir o rastreio de toda a prova ao final da P-derivação. Este rastreio é a tarefa do terceiro módulo do sistema que objetiva a recuperação das ligações ocorridas durante a derivação da prova.

5.3.1 O Controle da Fórmula Objetivo

Como pode ser visto pela gramática fornecida na seção anterior, uma fórmula objetivo pode conter disjunções, conjunções, conectivos existenciais ou ser uma simples proposição.

A consulta é uma fórmula objetivo. Há a necessidade de se gerenciar a escolha das fórmulas que a compõe para que estas sejam confrontadas com a base de conhecimento. Esta escolha ou controle é função, basicamente, dos dois conectivos $\vee$ e $\wedge$ e do simbolo existencial $\exists$, que possam existir na consulta, ou que nelas possam ser inseridos. Seja por exemplo a consulta:

Avo X , MaisDeSessenta X.

A consulta é uma conjunção. Supondo que exista na base a regra

Avo Y :- Pai Y , Casado Y

e que a P-derivação a escolha para unificar, a consulta tornar-se-á

Pai Y , Casado Y, MaisDeSessenta Y.

Houve uma expansão da consulta ou fórmula objetivo. Poderia haver contração, caso em que a unificação ocorresse com um *fato* da base. Observa-se ainda que a inserção poderia trazer consigo conectivos como $\vee$ ou $\wedge$ e o existencial $\exists$, já que provêm da parte direita de uma fórmula objetivo. O modo de encaminhar o processamento de uma consulta depende

portanto de sua forma. Para prosseguir a P-derivação, deve-se optar por uma das fórmulas da consulta para se proceder a unificação. Esta opção obedece a critérios estabelecidos pelo implementador.

Para o de seleção propõe-se aquele adotado pelo PROLOG padrao, cujo interpretador contem os seguintes passos que resumem seu ciclo de resolução.

- Inicio : Forma a lista de cláusulas objetivo
- Regra de seleção da fórmula objetivo : Escolha a primeira fórmula do conjunto para a unificação
- Regra de seleção da sentença dentro da base:
  1. Selecione a próxima cláusula dentro do Programa P. Tente a unificação desta cláusula com a fórmula objetivo escolhida. Seja θ o unificador.
  2. Formação da nova fórmula objetivo: Aplique a substituição θ ao corpo da sentença que unificou, se esta não for um fato e o insira no inicio da consulta, em substituição à formula escolhida. Aplique θ ao conjunto obtido.
  3. Se o resultado for uma cláusula vazia, retorne sucesso. Senão volte para o item de seleção da fórmula objetivo.
- Retroencadeamento : Se houver fracasso para alguma sentença da base, tome a sentença seguinte com a mesma cabeça, até que não mais exista tal sentença ou tenha havido um sucesso.

Esta sistemática de escolha de fórmulas e sentenças, desde que conhecidas, pode ser usada para interferir no mecanismo de provas pois a ordem com que as fórmulas são colocadas na consulta e as sentenças na base indicam quais ramos serão primeiro testados na árvore da resolução. No entanto, o PROLOG padrão contem a maneira mais simples de implementação. Uma *função heurística de avaliação* pode ser acrescida ao mecanismo padrão para otimizar o número de passos necessários para uma prova. [Chan73] cita uma série características da base e da consulta que podem ser usadas para criar tal função. Dentre elas destacam-se: o número de fórmulas no corpo da sentença a escolher, a profundidade da cláusula, o número de constantes da fórmula, etc.

Nosso protótipo adota o método de opção do PROLOG padrão, e deixa para etapas posteriores a preocupação com a otimização neste nível, por acreditar que para a lógica supeiror existem pontos de otimização mais promissores como a unificação e que representa um ponto que demanda mais investigaçoes.

A técnica do PROLOG padrão para processar proposições simples e conjunções consiste em se obter uma P-derivação para a primeira fórmula objetivo da conjunção e, em seguida, P-derivar θG_2, onde G_2 é a segunda fórmula objetivo da conjunção, e θ é a substituição resposta para a P-derivação de G_1. Se a conjunção contiver mais fórmulas objetivo, segue-se adiante aplicando as substituições resposta e tentando suas P-derivações. Este método

requer o controle de todo o mecanismo e é descrito com maiores detalhes adiante. A conjunção é considerada como um conjunto de fórmulas objetivo. Se há uma simples proposição tem-se um conjunto unitário.

No caso de existirem disjunções o controle transforma a fórmula objetivo em dois conjuntos de fórmulas objetivo. Assim se o conjunto objetivo tem a forma $O = \{G_1 \vee G_2\} \wedge G$ obter-se-á $O_1 = G_1 \vee G$ e $O_2 = G_2 \vee G$. O sistema procurará P-derivar a primeira opção e retornará ao segundo conjunto O_2 como se fora o processamento de uma nova consulta.

O tratamento do quantificador existencial é feito pela substituição da variável existenciada por uma nova váriavel livre y, ainda não pertencente ao conjunto de variáveis livres. Esta operação equivale à obtençao da forma β-normal da aplicação da fórmula existenciada a y.

Este controle é permanente. Ao suceder uma unificação, novos conectivos podem ser introduzidos e o controle é executado a cada entrada de novas fórmulas no conjunto.

5.3.2 Encadeamento: Escolha da Regra ou Fato

Para entender melhor o procedimento da P-derivação e as informações que devem ser guardadas, para posterior recuperação, seja a seguinte base de conhecimento:

$$
\begin{array}{l}
A_{p1} : - G^1_{p1}, G^2_{p1}, \cdots, G^n_{p1} \\
A_{p2} : - G^1_{p2}, G^2_{p2}, \cdots, G^n_{p2} \\
\vdots \\
A_{pi} : - G^1_{pi}, G^2_{pi}, \cdots, G^n_{pi} \\
A_{pj} : - G^1_{pj}, G^2_{pj}, \cdots, G^n_{pj} \\
\vdots \\
A_{pm} : - G^1_{pm}, G^2_{pm}, \cdots, G^n_{pm}
\end{array}
$$

e a seguite consulta:

$$G^1, G^2, \cdots, G^n.$$

Cada A_{pi} representa um novo procedimento (na visão procedural) e cada G_{pi} uma fórmula objetivo dentro da cláusula.

Com a fórmula objetivo inicial controlada conforme descrito em 5.3.1, dispara-se o processo da P-derivação. Um G^i da consulta é escolhido para ser P-derivado.

A escolha pode ser aleatória, seguir alguma função heurística, ou mesmo seguir um ordenamento sequencial, da primeira à última, como é o caso do protótipo em discussão.

A P-derivação inicia-se com a busca, na base de conhecimento, de uma cláusula cuja cabeça do atomo rígido A_{pi} se iguale àquela da primeira fórmula objetivo da consulta, e que potencialmente seria unificável. Em seguida é tentada uma unificação deste G^i com o possível candidato A_{pi}.

Todas as fórmulas candidatas devem ser testadas. Elas podem não unificar e o procedimento seleciona outra candidata, caso haja. Quando houver sucesso na unificação com, por exemplo A_{pj}, ocorre o passo do encadeamento, o que equivale a acrescentar θG^i_{pj}, $0 \leq i \leq n$ à consulta ou conjunto objetivo, retirando-se dela a fórmula G^i escolhida para unificar. Aqui θ é o unificador ou um dos unificadores entre A_{pj} e G_i. O processo de inserção de novas fórmulas objetivo na consulta, bem como a eliminação das que já tiverem sido resolvidas foi feito por meio de apontadores que simularam uma lista encadeada de fórmulas.

A figura 5.5 indica as posições sobre o conjunto objetivo e sobre a base de conhecimento que devem ser anotadas para dar prosseguimento à derivação. Nela usou-se a consulta *Avo X , MaisDeSessenta X* sobre uma base de dados que continha, dentre outras, as seguintes sentenças:

Avo Y :- Pai Y , Casado Y
Avo Y :- Pai Y , Filho X Y, Pai X

Os símbolos ↑ apontam para posições dentro da cadeia a partir das quais o processo deve ser retomado. O símbolo → identifica sentenças que estão sendo trabalhadas enquanto o símbolo ⇒ indica, quando existir, a próxima sentença a ser investigada. Este sistema de apontadores tem por consequência não ser necessário movimentar cadeias.

Para isto foram usadas algumas variáveis gerais de controle que descrevem o estado da P-derivação e uma *pilha de derivação*. Na pilha são colocadas as referências necessárias à continuação do processo. Cada linha da pilha registra as informações referentes à cláusula cujo átomo A foi unificado. Se a escolha daquela cláusula levar a um insucesso, as linhas da pilha posteriores à sua linha serão descartadas.

Tanto a pilha de derivação como a descrição de ambiente que será nela inserida, fazem parte do método denominado WAM, [Ait-90], e em alguns casos a denominação é a mesma. Com algumas alterações, a implementação utilizou as técnicas nela contidas.

Informações que devem ser mantidas na pilha, a cada entrada são:[1]

a) Ponto de Retorno : (PontoDeRetorno) Sempre indica para a próxima fórmula objetivo a ser P-derivada.

b) Linha da pilha pai (LinhaPai) para a linha atual. Nela se encontram informações sobre a cláusula pai da cláusula atual.

c) Próxima cláusula candidata (ProximaClauCandidata) à uma possível unificação com a fórmula objetivo considerada no momento.

d) Ponto de retroencadeamento anterior (PtoRetroAnterior) : indica a linha da pilha onde se encontra o mais recente ponto de retorno.

Fora da pilha, as seguintes variáveis são importantes:

a) Linha do mais recente ponto de retrocesso (LiMaisRecenteRetro). Atualizará o PtoRetroAnterior sempre que nova linha for criada na pilha.

b) Linha da última cláusula chamadora (LinhaUltClauChamadora): aponta sempre para a mais recente linha criada cujo passo deixou ainda uma cláusula candidata sem testar.

[1] Os nomes entre parenteses indicam os nomes que as variáveis assumiram no interpretador

Situação inicial :

Consulta :

Avo Y , MaisDeSessenta Y.
↑

Base de Dados:

⋮
Avo Y :- Pai Y , Casado Y
Avo Y :- Pai Y , Filho X Y, Pai X
⋮

Situação após a primeira seleção:

Consulta:

Pai Y , MaisDeSessenta Y.
↑

Base de Dados

⋮
→ Avo Y :- Pai Y , Casado Y
↑
⇒ Avo Y :- Pai Y , Filho X Y, Pai X
⋮

Figura 5.5: Apontadores necessários para encadear a consulta

c) Linha atual da pilha (LinhaAtual).

d) Cláusula para a linha atual (CláusulaAtual)

A escolha da cláusula, as anotações sobre as próximas cláusulas, o controle da cláusula que está sendo trabalhada e alguns outros detalhes podem ser vistos na rotina abaixo:

Antes de iniciar a P-Derivação estas variáveis são inicializadas. LinhaAtual recebe 1, CláusulaAtual aponta para a Consulta, PontoDeRetorno recebe 'saida' e as demais variáveis são anuladas.

```
Prcedimento Escolha da Cláusula

    Se ClausulaAtual é um fato
                        { Um fato é um procedimento sem
                        chamadas. Neste caso deve-se procurar anotar
                        como proxima chamada algum ponto de retorno
                        anterior que ainda não tenha sido explorado}
    então
       inicio
          FormObjChamadora := PontoDeRetorno[LinhaAtual]
          enquanto FormObjChamadora = 'saida' e LinhaUltClauChamadora ≠ 1
             inicio
                FormObjChamadora := PontoDeRetorno[LinhaUltClauChamadora]
                LinhaUltClauChamadora := LinhaPai[LinhaUltClauChamadora]
             fim
          Se FormObjChamadora = 'saida' então
             inicio
                Saia do sistema com a solução /* a consulta está vazia */
                Vá para o procedimento de retroencadeamento
             fim
       fim
    senão
       inicio
          FormObjChamadora := Primeira Chamada dentro da ClausulaAtual
          LinhaUltClauChamadora := LinhaAtual
          Se não houver cláusula candidata para FormObjChamadora
             então vá para a rotina de retroencadeamento.
             senão ClausulaCandAtual := Primeira cláusula candidata na
                                                    sequência
                   Continua no Procedimento de inserção na Pilha
       fim
```

Eleita a fórmula objetivo chamadora e a cláusula candidata atual, o sistema continuará procurando e testando cláusulas candidatas até que ocorra uma unificação. Pode ser que nenhuma candidata encontrada unifique. Neste caso ClausulaAtual será nulo. Se em algum instante ClausulaAtual for nulo, deve-se ir para o procedimento de retroencadeamento. Há ainda a possibilidade de haver uma ou mais cláusulas candidatas. O processo se desenvolve testando todas as candidatas. Quando uma delas unificar pode restar candidatas a testar. Neste caso, há um ponto de retroencadeamento que deve ser anotado. Observa-se que o procedimento de escolha da cláusula apenas faz as anotações necessárias para disparar o processo.

A rotina de unificação devolve um sinal de insucesso ou um conjunto de unificadores,

onde cada elemento do conjunto é um conjunto composto de substituições X^i/Y^i com $0 \leq i \leq n$ onde n é o número de variáveis livres do par da fórmula objetivo unificada.

5.3.3 O Mecanismo de Controle da P-derivação

O conjunto de regras e fatos constitui um programa lógico P. Uma consulta G pode ser respondida gerando-se uma árvore de pesquisa e identificado-se os ramos que conduziram a um sucesso. O objeto sintático obtido pela identificação destes ramos chama-se prova de G em P.

A escolha da cláusula da base candidata à unificação e a escolha da fórmula da consulta podem obedecer a critérios heurísticos. Quando se tem vários unificadores a escolha de um deles pode sofrer influência de condicionantes visando encontrar mais rapidamente uma solução. Considerando a implementação, oportunidades de melhorar a velocidade de execução residem no mecanismo da unificação, no compartilhamento de estruturas de dados, na compactação da pilha de controle, dentre outros.

Como já foi feito para a linguagem da lógica de primeira ordem, e continua sendo objeto de investigações, funções heuristicas podem ser utilizadas para acelerar e/ou melhorar este processo de busca.

Em nossa proposta, a P-derivação conta inicialmente com um vetor, obtido na inicialização, que indexa cada regra ou fato da base de conhecimento, pela sua posição. O apêndice A contem um exemplo deste vetor. A partir desta situação, dada uma fórmula objetivo G, opta-se por um tratamento adequado para a sua redução conforme descrito na subsecção 5.3.1. Tal redução elegerá uma das fórmulas constituintes da fórmula objetivo G, como candidata ao passo de encadeamento.

Os passos da pesquisa geram uma árvore cuja raiz é rotulada com a consulta. Cada arco é rotulado com a fórmula escolhida na base para unificar com uma das fórmulas da consulta expandida ou reduzida, na dependência do fato ou regra utilizados.

Para controle de toda a execução e para atender a todo o processamento de busca de novas soluções, bem como do retrocesso na árvore, é necessário que a cada fato ou regra (cláusula) testados uma série de apontamentos sejam feitos.

A técnica consagrada para isso é o uso de pilha ou pilhas, basicamente pela possibilidade de descartar segmentos inteiros, no caso de insucesso.

A *pilha de derivação* (figura 5.6) proposta conterá todas as informações necessárias para a retomada da P-derivação a partir de qualquer nó da árvore.

As informações são anotadas na pilha cada vez que uma regra ou asserção da base consegue ser unificada. Se há um insucesso na unificação alguns apontadores são alterados para a continuação do processo. No entanto a pilha não é acionada. Isto significa que a pilha manterá um histórico de todo o ramo da árvore que levou ao sucesso, ignorando as opções que levaram ao insucesso, e contendo ainda anotações sobre as cláusulas que ainda não foram testadas e que são uma opção para sucesso.

Além das informações listadas na subseccão anterior, a pilha armazena ainda:

a) Número de soluções (de unificação) não usadas.

b) Soluções ainda não usadas.
c) Número de variáveis livres.
d) Nome de cada variável livre
e) Posição no vetor TT, a partir da qual são armazenadas informações para recuperação da resposta para a prova. O vetor TT é uma pilha que armazena todas as ligações de variáveis a termos ou constantes feitas ao se passar de uma linha da pilha de derivação para outra, quando há um encadeamento.
f) Valores diversos que complementam a descrição do ambiente da P-derivação do momento e que permitem recuperá-lo posteriormente. Quando, em determinado ponto da derivação, o sistema ou usuário optou por um dos unificadores possíveis, a pilha de derivação registra toda a progressão da derivação até aquele momento. Outros conjuntos de unificadores podem ficar a espera para um retroencadeamento por unificação, isto é, para aproveitar todos os unificadores encontrados. As informações que são necessárias são dependentes da implementação. Em nosso protótipo, algumas informações são tidas como globais para todo o processo de derivação, e não estão armazenadas na pilha. Em especial, são estas informações que são anotadas para a recuperação do ambiente. Para esta eventualidade, as seguintes anotações foram feitas:

1) O mais recente ponto de retroencadeamento.
2) A linha na pilha que está sendo tratada.
3) Um apontador para a linha que originou a linha pai da atual.
4) Cláusula da base que está sendo trabalhada.
5) Cabeça da cláusula que está sendo trabalhada.
6) Posição dentro da cláusula.
7) Próxima cláusula candidata.
8) As soluções da unificação bem como as soluções da linha pai.

g) Para cada variável e linha da pilha é registrado um vetor de soluções, com os unificadores.

O seguinte segmento de algoritmo descreve a criação de uma nova linha na Pilha de derivação, após ocorrência de uma unificação.

```
Procedimento de criação de linha da Pilha
    LinhaAtual := LinhaAtual + 1
    LinhaPai[LinhaAtual] := LinhaUltClauChamadora
    Se FormObjChamadora for a ultima da Cláusula
            então PontoDeRetorno := 'saida'
            senão PontoDeRetorno := Formula Objetivo seguinte à atual
                                    dentro do Cláusula
    Se ClausulaCandAtual ≠ ∅ então
        inicio
            ProximaClauCand[LinhaAtual] := ClauCandAtual
            PtoRetroAnterior[LinhaAtual] := LiMaisRecenteRetro
            LiMaisRecenteRetro := LinhaAtual
        fim
    Vá para o procedimento de escolha de cláusula.
```

A cada unificação bem sucedida toma-se nova LinhaAtual que é atualizada com as informações necessárias.

Em continuidade novo G_{pi} será escolhido para prosseguir na unificação; nova pesquisa é feita dentre os procedimentos candidatos e todo o processo se repete. Novo conjunto de informações é armazenado ao suceder uma unificação; procedimentos candidatos podem ficar sem teste e por isso tal informação deve ser anotada. Após várias unificações e escolha de procedimentos a P-derivação pode terminar, conforme foi descrito na secção 4.3. Um exemplo de uso da pilha de derivação é fornecido no Apêndice A.

Neste momento, todo o histórico da P-derivação estará registrado na *Pilha de Derivação*. Pode-se então recuperá-lo para se obter uma explicação do caminho que conduziu ao sucesso ou simplesmente voltar a um ponto que permita continuar o processo da P-derivação por meio de outros ramos da árvore. No apêndice A pode ser visto o estado da pilha de derivação após a P-derivação da consulta.

5.3.4 A Operação de Retroencadeamento

A operação de retroencadeamento ocorre quando uma das chamadas torna impossível o prosseguimento da pesquisa. Pode ocorrer porque não há possibilidade de unificação e não há mais cláusulas candidatas e quando se chegou a um resultado e há ainda casos a explorar. Este último caso pode se dever a soluções de unificação que não foram testadas ou a cláusulas da base que tambem poderiam conduzir a um resultado mas que ainda não foram pesquisadas.

A proposta contida no protótipo mantém sempre um apontador para a posição da pilha de P-derivação que contem a descrição do ambiente da última possibilidade de um retroencadeamento. Seja este valor atribuido a variável *LiMaisRecenteRetro*. Este retroencadeamento atende a casos de sentenças da base ainda não trabalhadas como ocorre no PROLOG padrão. De modo parecido, existe a possibilidade de retorno para testar outros unificadores apresentados em casos onde a unificação devolveu vários conjuntos deles e ainda restam alguns que não foram utilizados. Esta linha é identificada como *LinhaX* no algoritmo.

Qualquer destas variáveis permite o reinício da P-derivação aproveitando-se a pilha até aquele ponto. A situação é similar no caso de uma P-derivação que termina com sucesso. Neste caso, deve haver a recuperação do resultado antes do retroencadeamento.

O controle deste mecanismo pode ser visto no algoritmo Procedimento para Retroencadeamento fornecido adiante.

O fato de não ser encontrado um unificador *"mgu"* na lógica de ordem superior acrescentou ao processo mais um ponto de retroencadeamento. Antes de voltar para tentar uma nova cláusula visando a atender determinada chamada, o interpretador deve verificar se todas as soluções de unificação foram testadas, quando houve a última unificação.

Quando se unificam duas fórmulas de ordem superior, e se escolhe um dos unificadores possíveis, pode-se estar interferindo no processo da P-derivação inserindo nele um não-

Pilha de Derivação							
LinhaPai	PontoDe Retorno	PróximaClau Candidata	PtoRetro Anterior	$N°$ Desta Linha	ClausulaAtual Anterior	Posição de P_j na Tabela	$N°$ de Var Livres
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Pilha de Derivação (cont.)									
$N°$ de Soluções Não Usadas	Variaveis Livres							Unificadores Disponiveis	Indice no Vetor TT
	$N°$ 1	$N°$ 2	$N°$ 3	$N°$ 4	$N°$ 5	.	.		
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

Pilha de Derivação (cont.) Espaço para Salvar Ambiente							
Mais R.Pt de Retro	Linha na Pilha	Linha Pai da Pai	Clausula da Base	Cabeça da Cláusula	Posição na Cláus.	Prox.Claus Candidat.	Soluções de Unif.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Figura 5.6: Pilha de derivação para controle da P-derivação

determinismo. Há portanto a necessidade de se testar todos os unificadores encontrados antes de se passar a testar novos procedimentos na base.

```
Procedimento para Retroencadeamento
    Se há unificadores não usados
    então identifique a linha como LinhaX
        Se LinhaX ≤ LiMaisRecenteRetro
            então Restabelece ambiente de LinhaX
                Vá para o procedimento de escolha da cláusula
    senão
    Se LiMaisRecenteRetro = ∅
        então termina execução
        senão
          inicio
            ClausulaCandAtual := ProximaClauCandidata[LiMaisRecenteRetro]
            FormObjetivoChamadora := Clausula que precede o
                        PontoDeRetorno[LiMaisRecenteRetro]
            LinhaUltClauChamada := LinhaPai[LiMaisRecenteRetro]
            LinhaAtual := LiMaisRecenteRetro - 1
            LiMaisRecenteRetro = PtoRetroAnterior[LiMaisRecenteRetro]
            Vá para o procedimento de escolha de cláusula
          fim
```

Na prática, o interpretador testa se o valor da linha de retrocesso para LiMaisRecenteRetro é maior do que aquele valor de LinhaX que representa um caso de soluções não utilizadas. Se o último ponto de volta for posterior a um ambiente onde não foram testadas todas as soluções, então o retrocesso se dará para aquele ponto. Se o último ponto de retrocesso que permite escolher outras sentenças for anterior a um ambiente no qual existem unificadores que ainda não foram utilizados, então novo unificador para aquele par de discórdia deve ser usado. Nesta situação, o ambiente é restabelecido naquele ponto, um novo unificador é escolhido dentre os disponiveis e o processo continua.

5.4 A Unificação

A unificação no λ-PROLOG, dentro do procedimento geral da P-derivação, tem a finalidade de tornar dois termos iguais, por meio de substituições adequadas. Conforme foi descrito no capítulo 4, é uma tarefa decidível no universo das cláusulas definidas de ordem superior.

Ao tentar unificar duas fórmulas pode-se obter um *insucesso* ou um *sucesso* com um ou mais unificadores. A possibilidade de haver mais do que um unificador introduz uma diferença acentuada no mecanismo de controle da derivação quando comparado com o mesmo controle na resolução de primeira ordem. Uma vez determinado o par de discórdia, a rotina de unificação é chamada, devolvendo o sinal de insucesso, ou os unificadores encontrados.

O acionamento da unificação corresponde portanto ao envio de duas fórmulas f^1 e f^2 para a tentativa de casamento ("matching"). Encontrado o conjunto de soluções, cabe estabelecer um critério para a escolha de um dos conjuntos unificadores para que se possa prosseguir na P-Derivação. Há portanto a necessidade de se criar mecanismos para aproveitar as soluções não usadas no momento.

O algoritmo da unificação tem sua estrutura geral apresentada na figura 5.7 e é descrito pelo pseudo-código colocado na figura 5.9. Ambas as figuras representam uma junção dos algoritmos colocados nas figuras 4.2 e 4.3, SIMPL e MATCH respectivamente, que foram propostos por [Huet75] e com pequenas alterações por [Nada87].

A estrutura apresenta a disposição dos módulos no procedimento. As duas partes mais significativas, SPL e MATCH, representam a simplificação e a unificação propriamente dita. Uma rotina auxiliar chamada SIMPLO é usada na simplificação. Já no módulo MATCH, há uma rotina de preparação (PRE-MATCH) e as duas rotinas IMIT e PROJECT representam os dois mecanismos básicos da unificação que possuem o mesmo nome. Observa-se ainda que o procedimento é recursivo. Ele só termina depois que os unificadores foram encontrados para o par de discórdia ou caso tenha ocorrido um insucesso. A chamada da unificação pode ocorrer diversas vezes na unificação de um par de discórdia já que o par pode ser quebrado em diversos pares de argumentos.

O procedimento geral pode ser melhor analizado na figura 5.8. O laço de repetição das quatro últimas linhas do procedimento realiza a recursão referida acima. Os pares $< f_1^i, f_2^i >$ citados no corpo do procedimento representam os pares de argumentos emparelhados. Os pares de letras *ff, fr, rr e rf* foram usados para simbolizar pares de fórmulas flexível-flexível, flexível-rígido, rígido-rígido e rígido-flexível, respectivamente.

Para implementação deste algoritmo foram desenvolvidas algumas rotinas específicas usadas no tratamento de expressões de ordem-superior, em especial:

1) Uma rotina para converter uma fórmula por meio da β- redução (B-RED). Como foi visto na seção 4.2 a aplicação de uma substituição a uma fórmula resulta numa fórmula λ-normal. Para a obtenção da formula λ-normal é necessária a

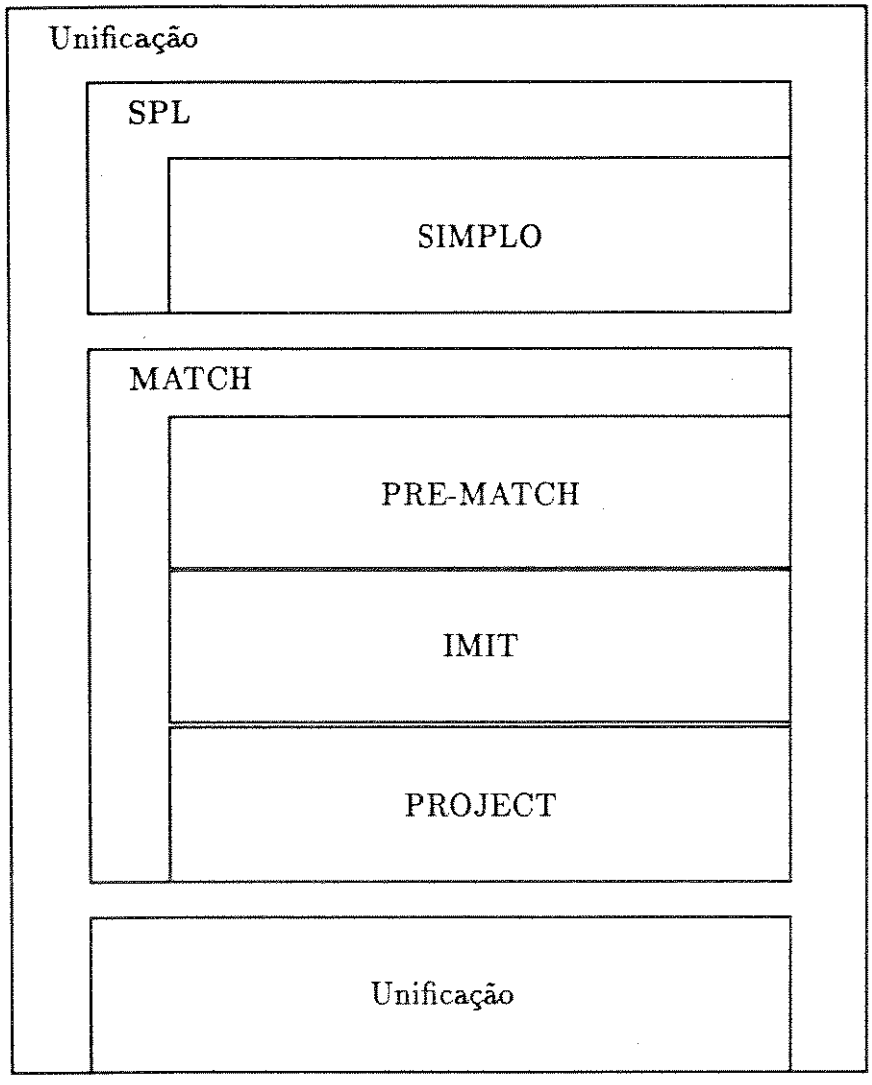


Figura 5.7: Estrutura da unificação

```
Unificação { recebe um conjunto discórdia }
Inicio
      Simpl := ∅
      Repita
      Para cada par de formulas < f1, f2 > do conjunto
        Inicio
          {Quebra o par em seus pares de argumentos}
          Para cada par < f1^i, f2^i > encontrado
          Classifique-o em ff, fr, rf, rr duplas flexível x rígido
          caso o par seja
             fr : continua como está
             ff : continua como está
             rf : transforma o par em < f2^i, f1^i >
             rr : Se as cabeças forem diferentes retorna 'Falha'
                  Senão Se tipo for fundamental elimina par do conjunto
                        Senão Se não estiver na FNU, coloque na FNU
                              Expande o par com os pares de argumentos
          Agrega resultado em simpl
        Fim
      Até que náo tenha havido expansão
      {Neste ponto só existem em simpl pares ff e fr ou Falha
      O conjunto recebido está simplificado}
      caso simpl seja
        ∅ : houve sucesso na unificação. Pare e retorne unificadores
        Falha : não houve sucesso na unificação. Pare e retorne.
        Outro : continue pois há pares ff e fr em simpl.
      Para cada par < f1^i, f2^i > ∈ simpl
        caso par seja
          ff : unificador := < f1^i / f2^i >
          fr : Coloca f1^i e f2^i na FNU
               Prepara f1^i e f2^i para MATCH
               Processa Imitação de < f1^i, f2^i >
               Processa Projeção de < f1^i, f2^i >
               {Retorna um conjunto de unificadores chamado c.match}
      Para cada unificador do c.match
        Aplique o unificador em simpl {Usa a β-redução}
        O conjunto simpl não estará mais simplificado
        Chame recursivamente a unificação para simpl
Fim
```

Figura 5.8: Pseudocódigo para o algoritmo da unificação

aplicação da β-redução. Esta transformação equivale a encontrar, no cálculo, a representação correta para a aplicação de f(x) a 3, por exemplo. Encontrado um unificador para uma variável, sua substituição é aplicada ao termo, para que o conjunto unificador total seja encontrado. Nesta aplicação a rotina B-RED é usada.

2) Uma rotina para classificar um par de termos em uma das possíveis combinações de flexível-rígido (FLEX-RIGID). Pares de termos flexível-rígido são unificáveis. Pares de termos rígido-rígido são unificáveis. Pares de termos rígido-rígido devem ter suas cabeças iguais para que a rotina de unificação prossiga. Os demais casos também devem ser tratados.

3) Uma rotina para transformar uma fórmula na sua forma normal de unificação (FNU). A forma normal de unificação foi definida na seção 4.2. Toda fórmula para ser unificada deve estar nesta forma para se adaptar ao algorítimo MATCH visto na figura 4.3. Esta rotina requer um conhecimento do tipo dos parâmetros e variáveis presentes no termo a ser normalizado.

4) Uma rotina para expandir um par de fórmulas rígidas (EXPANSÃO). Esta faz o emparelhamento dos argumentos de duas fórmulas rígidas do mesmo tipo e que estão em suas formas normais de unificação.

5) Uma rotina que (SPL) que controla os pares de fórmulas a serem passados a outra rotina (SIMPLO) qu fará o emparelhamento de seus argumentos.

6) Uma rotina (PRE*MATCH) que prepara as fórmulas flexível-rígida para se aplicar as rotinas IMIT e PROJECT. As duas últimas serão discutidas mais detalhadamente na seção seguinte e são baseadas no algoritmo da figura 4.3.

O pseudo-código colocado na figura 5.8 torna-se então em mais alto nível o algoritmo colocado na figura 5.9.

Atribuimos o nome de Match-Tree à rotina que constroi toda a árvore de unificação. O primeiro passo dentro desta rotina será aplicar a rotina SIMPL, e seu resultado contem o conjunto discórdia simplificado, com pares de discórdia flexível-flexível ou flexível-rígido ou uma indicação de que a unificação não foi possível.

Neste ponto ocorre uma opção de escolha do par de discórdia dentro do conjunto simplificado, ao qual se aplicará a rotina MATCH. A escolha deste par flexível-rígido deverá seguir algum critério pré-estabelecido. Nosso protótipo segue o PROLOG padrão ao deixar que o sistema opte pelo primeiro par da esquerda dentro do conjunto.

Escolhido o par que será fornecido a MATCH, a rotina devolverá os unificadores obtidos por meio da imitação e projeção.

Ao se optar por um dos unificadores do par e aplicá-lo ao restante do conjunto de discórdia estabelece-se um caminhamento por um ramo na árvore de unificação.

A aplicação do unificador ao conjunto elimina o par escolhido, ou reduz as duas fórmulas do par eliminando-se suas cabeças. No segundo caso, a unificação é aplicada em seguida aos pares de seus argumentos. Exige para isto, em geral, o uso da β-redução.

Unificação {Match*tree}
SPL {Toma todos os pares e em cada um executa SIMPLO}
SIMPLO {Quebra cada par em seus pares de argumentos}
FLEX*RIGID
FNU {Coloca na forma normal de unificação}
EXPANSÃO {Se houver expansão expande e retorna em SPL }
{No retorno só existem pares *ff*, *fr* ou **falha** }
{Toma o primeiro par }
FLEX*RIGID
MATCH
PRE*MATCH
IMIT
PROJECT
B*RED {β-redução do par}
Unificação {Para pares restantes}
{O Retorno tem vários conjuntos unificadores }

Figura 5.9: Estrutura da unificação. (A indentação identifica quais rotinas são chamadas e por quem).

Exemplo 5.4.1 Seja D o conjunto de discórdia

$\{\langle$(mapfun F_1 nil nil),
(mapfun F_2 (cons 1 (cons 2 nil))
(cons (g 1 1) (cons (g 1 2) nil))))$\}$,

onde F_1 e F_2 são variáveis do tipo *int* → *int* , nil é do tipo *(list int)*, cons é do tipo *int* → *(list int)* → *(list int)*, 1 e 2 são do tipo *int* e g é uma constante do tipo *int* → *int* → *int* . Neste caso SIMPL(G) = **F**, pois na expansão serão formados alguns pares rígido x rígido (rr) com cabeças diferentes, por exemplo ⟨nil, (cons 1 (cons 2 nil))⟩.

Por outro lado se D for o conjunto

$\{\langle$((mapfun F_1 (cons X L_1) (cons (F_1 X) L_2)),
(mapfun F_2 (cons 1 (cons 2 nil))
(cons (g 1 1) (cons (g 1 2) nil))))$\}$

onde X, L_1 e L_2 são variáveis, então SIMPL(D) será o conjunto de discórdia

$\{\langle F_1, F_2\rangle$, ⟨X, 1⟩ , ⟨(F_1 X),(g 1 1)⟩,
⟨L_1, (cons 2 nil)⟩, ⟨L_2, (cons (g 1 2) nil)⟩$\}$

Exemplo 5.4.2 Este exemplo contem os passos intermediários da unificação de duas fórmulas cujo resultado final foi fornecido em [Nada87], sem detalhamento. Oferece assim uma visão mais global do processo de pesquisa de um unificador e da árvore de unificação. Seja D o conjunto de discórdia

$\{\langle$(mapfun F_1 (cons X L_1) (cons (F_1 X) L_2)),
(mapfun F_2 (cons 1 (cons 2 nil))
(cons (g 1 1) (cons (g 1 2) nil))) ⟩$\}$

onde os tipos das constantes e das variáveis são os mesmos do exemplo anterior ou, se não declarados, são apropriados para a expressão.

A árvore construída na busca de unificadores para o par é mostrada abaixo. Em detalhe é mostrado um ramo da árvore formada na pesquisa. Este ramo foi obtido da àrvore de unificação fornecendo-se à MATCH sempre o primeiro par flexível-rígido à esquerda dentro do conjunto discórdia e seguindo pelo ramo rotulado pelo primeiro unificador (à esquerda) fornecido para o par escolhido. É um caminhamento descendente à esquerda.

A raiz é rotulada com D e nela é aplicado o procedimento SIMPL. As variáveis que aparecem na pesquisa são resultado da expansão de fórmulas para que o par, ou um dos termos do par, se torne um termo na forma normal de unificação.

A cada aplicação de uma substituição pode ocorrer a necessidade de uma β-redução, no caso em que o termo que substitui contenha uma abstração. O símbolo C foi usado para representar (cons (g 1 2) nil) para diminuir espaço.

$\{\ \langle (mapfun\ F_1\ (cons\ X\ L_1)(cons\ (F_1\ X)\ L_2)), (mapfun\ F_2$
$(cons\ 1\ (cons\ 2\ nil))(cons\ (g\ 1\ 1)(cons\ (g\ 1\ 2)\ nil)))\rangle\ \}$

⋮

rotina SIMPL ⋮

⋮

$\{\langle F_1, F_2\rangle, \langle X, 1\rangle, \langle L_1, (cons\ 2\ nil)\rangle, \langle (F_1\ X), (g\ 1\ 1)\rangle,$
$\langle L_2, (cons\ (g\ 1\ 1)(cons\ (g\ 1\ 2)\ nil))\rangle\ \}$

⋮

⋮

$X/1$ ⋮

⋮

$\{\langle F_1, F_2\rangle, \langle L_1, (cons\ 2\ nil)\rangle, \langle (F_1\ 1), (g\ 1\ 1)\rangle, \langle L_2, (cons\ (g\ 1\ 2)\ nil)\rangle\}$

⋮

⋮

$L_1/(cons\ H_1\ H_2)$ ⋮

⋮

⋮

$\{\langle F_1, F_2\rangle, \langle H_1, 2\rangle, \langle H_2, nil\rangle, \langle (F_1\ 1), (g\ 1\ 1)\rangle, \langle L_2, (cons\ (g\ 1\ 2)\ nil)\rangle\}$

⋮

⋮

$H_1/2$ ⋮

⋮

⋮

$\{\langle F_1, F_2\rangle, \langle H_2, nil\rangle, \langle (F_1\ 1), (g\ 1\ 1)\rangle, \langle L_2, (cons(g\ 1\ 2)nil)\rangle\}$

⋮

⋮

H_2/nil ⋮

⋮

⋮

$\{\langle F_1, F_2\rangle, \langle (F_1\ 1), (g\ 1\ 1)\rangle, \langle L_2, (cons(g\ 1\ 2)nil)\rangle\} \otimes 1$

$F_1/\lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))$ $\qquad$ $F_1/\lambda W_1(W_1).$

$\{\langle F_2, \lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))\rangle, \langle (H_3\ 1), 1\rangle, \langle (H_4\ 1), 1\rangle, \langle L_2, C\rangle\} \otimes 2$

$F_2/\lambda W_2(g\ (H_5\ W_2)\ (H_6\ W_2))$ $\qquad$ $F_2/\lambda W_2(W_2)$

$\{\langle \lambda W_3(H_5\ W_3), \lambda W_3(H_3\ W_3)\rangle, \langle \lambda W_3(H_6\ W_3), \lambda W_3(H_4\ W_3)\rangle, \langle (H_3\ 1), 1\rangle, \langle (H_4\ 1), 1\rangle, \langle L_2, C\rangle\} \otimes 3$

$H_3/\lambda W_3(1)$ $\qquad$ $H_3/\lambda W_3(W_3)$

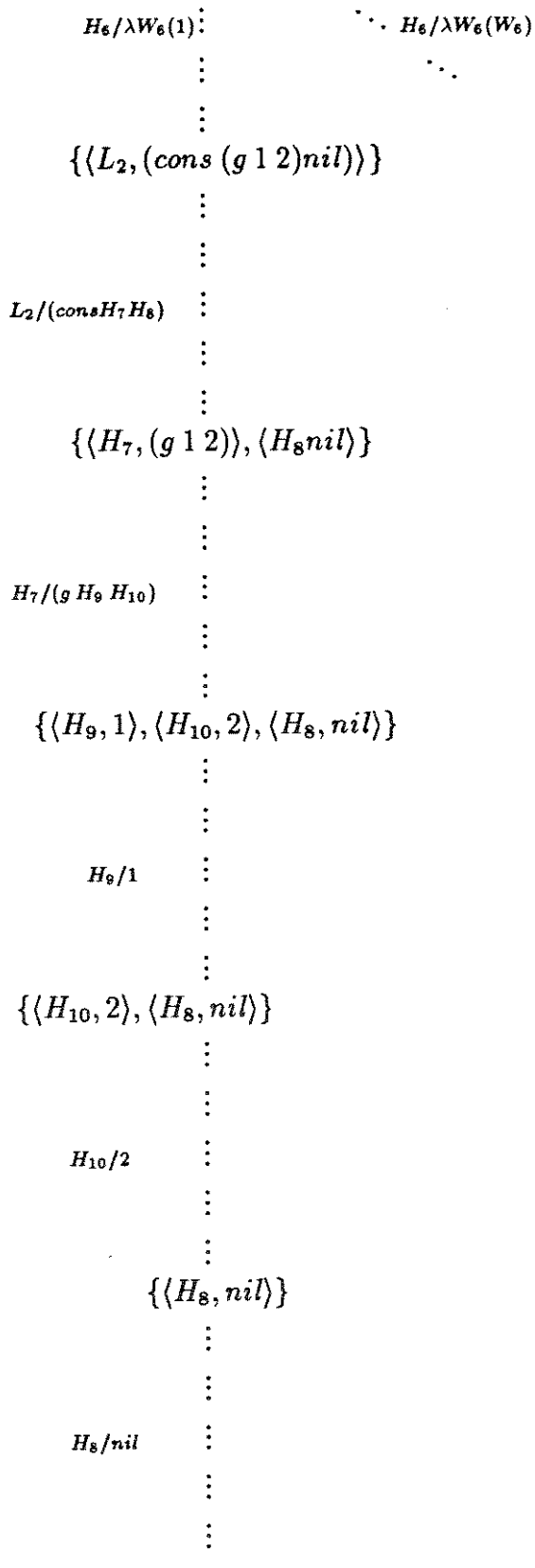
$\{\langle \lambda W_3(H_5\ W_3), \lambda W_3(1), \rangle, \langle \lambda W_3(H_6\ W_3), \lambda W_3(H_4\ W_3)\rangle, \langle (H_4\ 1), 1\rangle, \langle L_2, C\rangle\} \otimes 4$

$H_5/\lambda W_4(1)$ $\qquad$ $H_5/\lambda W_4(W_4)$

$\{\langle \lambda W_3(H_6\ W_3), \lambda W_3(H_4\ W_3)\rangle, \langle (H_4 1), 1\rangle, \langle L_2, C\rangle\} \otimes 5$

$H_4/\lambda W_5(1)$ $\qquad$ $H_4/\lambda W_6(W_6)$

$\{\langle \lambda W_3(H_6 W_3), \lambda W_3(1)\rangle, \langle L_2, C\rangle\} \otimes 6$

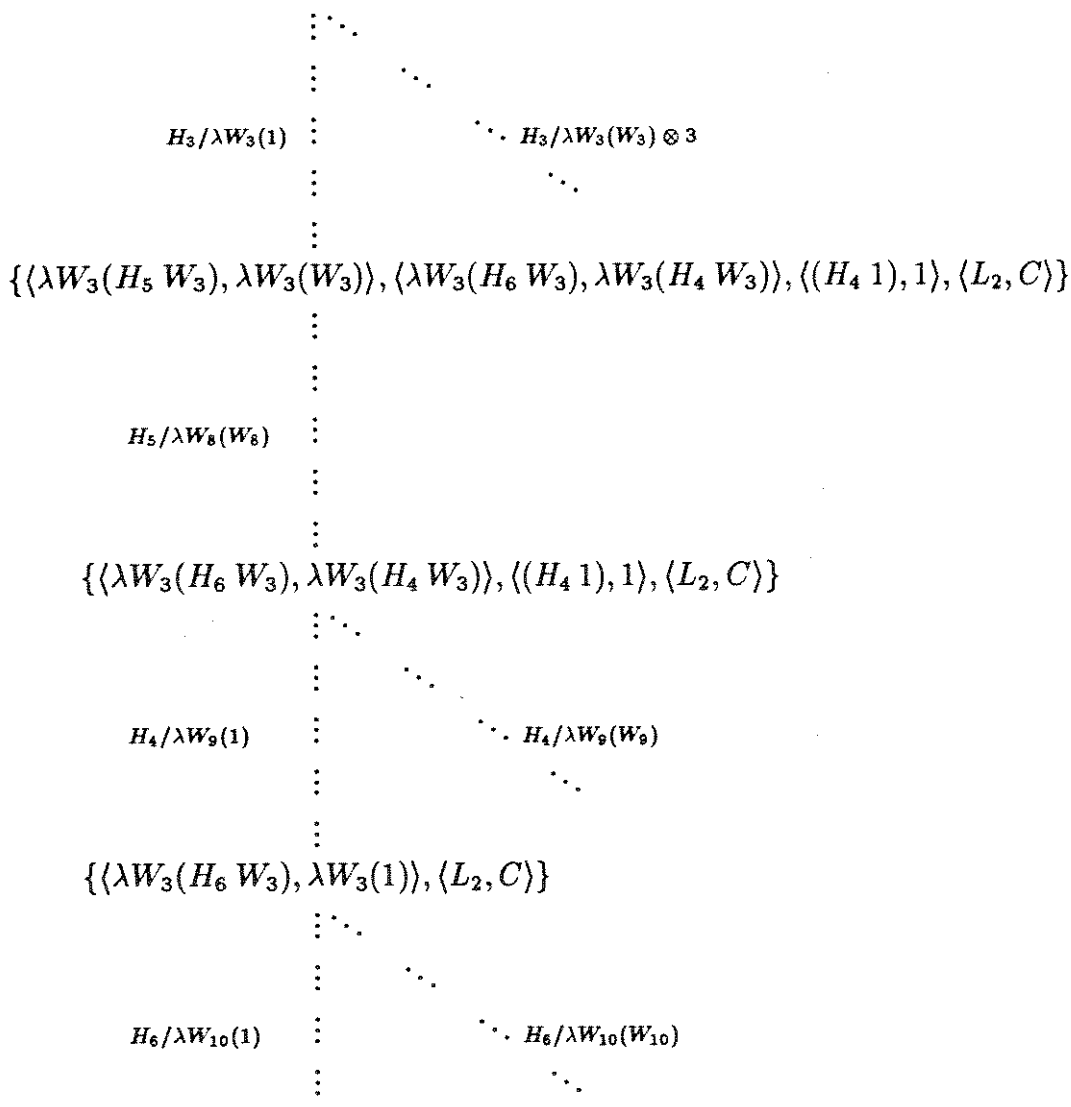

$H_6/\lambda W_6(1)$
$H_6/\lambda W_6(W_6)$
$\{\langle L_2, (cons\ (g\ 1\ 2) nil)\rangle\}$
$L_2/(cons H_7 H_8)$
$\{\langle H_7, (g\ 1\ 2)\rangle, \langle H_8 nil\rangle\}$
$H_7/(g\ H_9\ H_{10})$
$\{\langle H_9, 1\rangle, \langle H_{10}, 2\rangle, \langle H_8, nil\rangle\}$
$H_9/1$
$\{\langle H_{10}, 2\rangle, \langle H_8, nil\rangle\}$
$H_{10}/2$
$\{\langle H_8, nil\rangle\}$
H_8/nil

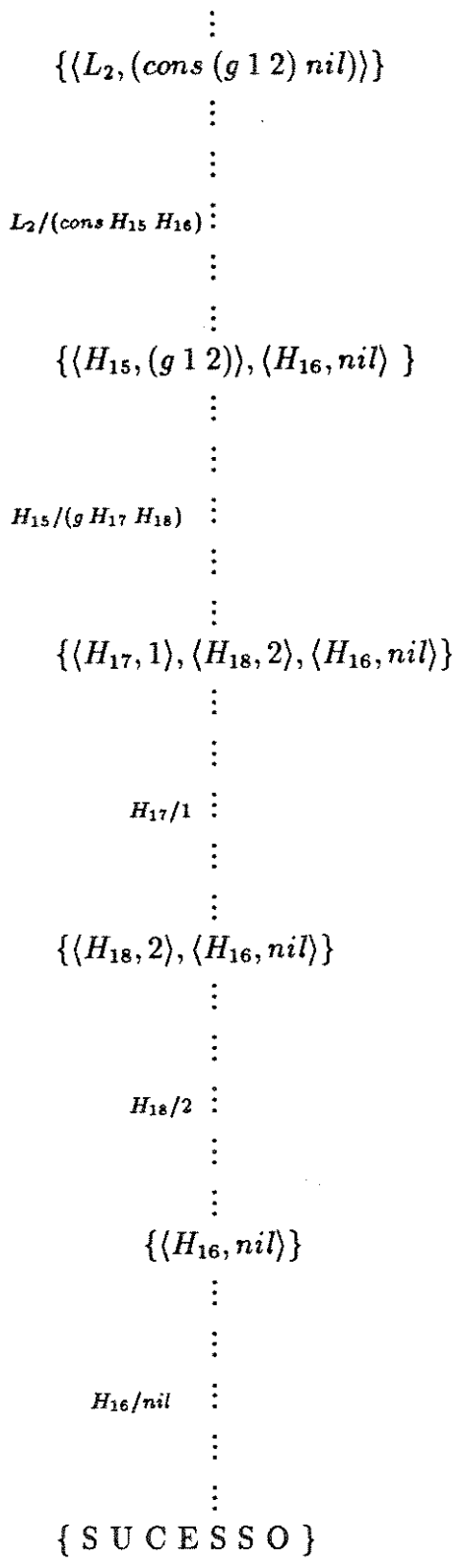
{ S U C E S S O }

Os símbolos ⊗i foram usados para identificar alguns pontos de ramificação da árvore de unificação que serão referenciados na sequência desta apresentação.

O número de ramos nestes pontos é determinado pela rotina MATCH. Os ramos ⊗1 e ⊗2 levam ao insucesso da unificação no primeiro nó a ser estabelecido. Isto é, a opção pelo segundo unificador, mais especificamente aquele gerado por projeção, leva a um insucesso da unificação. O ramo ⊗3 é o único a formar uma sub-árvore com bifurcações e terminam com duas folhas de sucesso e uma de falha.

Os ramos ⊗4, ⊗5 e ⊗6 terminam em folhas de falha, sucesso e falha, respectivamente. A explosão do ramo ⊗3 é feita abaixo, escolhendo-se novamente o unificador da esquerda quando houver mais de um para o par escolhido.





Cada caminho sobre a árvore de unificação que termina numa folha com sucesso gera

uma sequência de unificadores σ_i , que são unificadores parciais para cada par dentro do conjunto dicórdia. Neste exemplo, quatro caminhos terminam com sucesso e as sequências de unificadores σ_i são mostradas abaixo:

Sequência 1 (Esta sequência está no primeiro ramo do exemplo):

$\vdots$
$X/1$
$L_1/(cons\ H_1\ H_2)$
$H_1/2$
H_2/nil
$F_1/\lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))$
$F_2/\lambda W_2(g\ (H_5\ W_2)(H_6\ W_2))$
$H_3/\lambda W_3(1)$
$H_5/\lambda W_4(1)$
$H_4/\lambda W_5(1)$
$H_6/\lambda W_6(1)$
$L_2/(cons\ H_7\ H_8)$
$H_7/(g\ H_9\ H_{10})$
$H_9/1$
$H_{10}/2$
H_8/nil

Sequência 2

$X/1$
$L_1/(cons\ H_1\ H_2)$
$H_1/2$
H_2/nil
$F_1/\lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))$
$F_2/\lambda W_2(g\ (H_5\ W_2)(H_6\ W_2))$
$H_3/\lambda W_3(1)$
$H_5/\lambda W_4(1)$
$H_4/\lambda W_5(W_5)$
$L_2/(cons\ H_{11}\ H_{12})$
$H_{11}/(g\ H_{13}\ H_{14})$
$H_{13}/1$
$H_{14}/2$
H_{12}/nil

Sequência 3
(Refere-se ao ramo ⊗3):

$X/1$
$L_1/(cons\ H_1\ H_2)$
$H_1/2$
H_2/nil

$F_1/\lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))$
$F_2/\lambda W_2(g\ (H_5\ W_2)(H_6\ W_2))$
$H_3/\lambda W_3(W_3)$
$H_5/\lambda W_8(W_8)$
$H_4/\lambda W_9(1)$
$H_6/\lambda W_{10}(1)$
$L_2/(cons\ H_{15}\ H_{16})$
$H_{15}/(g\ H_{17}\ H_{18})$
$H_{17}/1$
$H_{18}/2$
H_{16}/nil

Sequência 4:

$X/1$
$L_1/(cons\ H_1\ H_2)$
$H_1/2$
H_2/nil
$F_1/\lambda W_1(g\ (H_3\ W_1)(H_4\ W_1))$
$F_2/\lambda W_2(g\ (H_5\ W_2)(H_6\ W_2))$
$H_3/\lambda W_3(W_3)$
$H_5/\lambda W_8(W_8)$
$H_4/\lambda W_9(W_9)$
$H_6/\lambda W_{11}(W_{11})$
$L_2/(cons\ H_{19}\ H_{20})$
$H_{19}/(g\ H_{21}\ H_{22})$
$H_{21}/1$
$H_{22}/2$
H_{20}/nil

As variáveis livres em D cujas substituições se espera conseguir com a unificação são: X,F_1, L_1 e L_2. Os quatro caminhos acima, onde cada rótulo de ramo indica um unificador σ_i, formam cada um uma sequência $\sigma_1 \cdots \sigma_n$, a partir da raiz. A composição $\sigma_n \circ \cdots \circ \sigma_2 \circ \sigma_1$ fornecerá a substituição resposta para cada sequência:

Substituição resposta 1:

$\{\ F_1/\lambda W_1(g\ 1\ 1)\ ,$
$X/1,$
$L_1/(cons\ 2\ nil),$
$L_2/(cons\ (g\ 1\ 2)\ nil)\ \}$

Substituição resposta 2:

$\{\ F_1/\lambda W_1(g\ 1\ W_1),$
$X/1,$

$L_1/(cons\ 2\ nil)$,
$L_2/(cons\ (g\ 1\ 2)\ nil)$ }

Substituição resposta 3:

{ $F_1/\lambda W_1(g\ W_1\ 1)$,
$X/1$,
$L_1/(cons\ 2\ nil)$,
$L_2/(cons\ (g\ 1\ 2)\ nil)$ }

Substituição resposta 4:

{ $F_1/\lambda W_1(g\ W_1\ W_1)$,
$X/1$,
$L_1/(cons\ 2\ nil)$,
$L_2/(cons\ (g\ 1\ 2)\ nil)$ }

Um procedimento que busca uma P-derivação pode então ser visto como um conjunto de ações que ora escolhem as fórmulas a serem unificadas, ora unifica fórmulas. A escolha das fórmulas, como na resolução em lógica de primeira ordem, considera a base de conhecimento e a fórmula objetivo (consulta).

O nome *P*-derivação está associado a um Programa que será a base de conhecimento. Assim, no exemplo anterior, a seguinte situação seria a geradora daquela unificação, por exemplo:

Consulta:
(mapfun F_2 (cons nil)) (cons (g 1 1) (cons (g 1 2) nil)))
Programa :
(mapfun F_1 nil nil)
(mapfun F_1 (cons X L_1) (cons (F_1 X) L_2)) :- (mapfun F_1 L_1 L_2)

onde a *P*-derivação deve considerar as duas regras do programa e a estrutura da consulta.

O controle do fluxo da derivação fica mais complexo a medida que a consulta não seja tão simples e um conjunto maior de regras e fatos constituirem o programa. O apêndice A mostra com maior detalhe a execução da pesquisa acima.

5.5 Considerações Gerais

Este capítulo descreveu a experiência da implementação de um protótipo de interpretador λ-Prolog. À medida que alguma etapa apresentou características diferentes daquelas do PROLOG padrão, tomado como referencial, principalmente no que se refere ao mecanismo de derivação, alguns métodos de implementação foram propostos. O protótipo foi construido utilizando-se os algoritmos MATCH e SIMPL expostos em [Nada87].

5.5.1 Implementação

As características específicas das lógicas que modelam cada interpretador acarretaram adaptações de alguns procedimentos como aqueles da unificação e da formação da base de dados. Acréscimos mais acentuados no λ-PROLOG foram feitos para explorar as novas possibilidades de retrocesso criadas pela obtenção de mais do que um unificador, cujo aproveitamento exigiu procedimentos especiais.

Para o controle ampliou-se o conceito consagrado nos interpretadores PROLOG que usa pilhas de apontadores. A parte acrescida em torno dos múltiplos unificadores fez com que a proposta existente neste trabalho incorporasse o primeiro e, sob este aspecto, se tornasse uma generalização para a ordem superior.

Os desafios das implementações na primeira ordem sempre foram aqueles relativos à eficiência. Ao reconhecidamente maior tempo de processamento dos programas lógicos, pode-se atribuir duas causas: o não determinismo e a inexistência de técnicas eficientes de manuseio de dados. Admitindo-se que o mecanismo de busca da prova usando a base de conhecimento é uma pesquisa em grafo, cujas soluções admite-se bem conhecidas, o mecanismo de controle já proposto para tais tarefas foi aceito como resolvido. Tornaram-se então tópicos de preocupações: a representação das atribuições e estruturas dos dados, a demanda por memória e o tempo de processamento.

As preocupações com a atribuição de valores dos dados às variáveis se liga principalmente a necessidade de construir, descartar e alterar dados. Estas ações são executadas um número muito grande de vezes e qualquer mudança na forma de executá-las pode provocar alterações significativas no desempenho dos algoritmos. Considerando benefícios que se pode ter (1) com a escolha de uma boa estrutura para armazenar os diferentes valores que uma variável pode assumir e (2) com uma boa representação para o dado, várias propostas foram feitas visando a vantagens neste sentido.

A maior parte dos esforços, em primeira ordem, foi alocada na melhoria do processamento de termos com representação estruturada. Uma destas técnicas propõe o compartilhamento de uma estrutura com diversos valores, por meio de vários assinalamentos sobre a mesma estrutura, dispensando múltiplas cópias do mesmo dado, o que provoca grande compactação no armazenamento. O assunto continua sendo objeto de discussão e a escolha entre técnicas de compartilhamento ou não compartilhamento das estruturas depende do tipo de aplicação pretendida. A maioria da implementações usam compartilhadamente as estruturas.

Dados que utilizam espaço maior, e estruturas irregulares ainda não foram devidamente aquinhoados com pesquisas para dar-lhes tratamento que proporcione melhor tecnologia para a implementação.

O estado da arte descrito acima pertence à lógica de primeira ordem, à construção do interpretador PROLOG. Assumindo que o λ-PROLOG incorporaria o PROLOG, várias características de sua implementação são dele importadas. No entanto outras características devem ser mais bem analisadas ou submeter-se a pesquisas por que sua proposta se assenta basicamente sobre a estrutura dos λ-termos, diferentes dos termos da lógica de primeira

ordem.

As implementações existentes do λ-PROLOG são ainda grosseiras para permitir verificar se se pode assumir que um interpretador tenha incorporado outro. Segundo [Nada88], muito trabalho deve ser feito na direção de implementações sérias do λ-PROLOG. Este trabalho apresenta uma série de pontos que confirmam esta assertiva, acentuadamente em detalhes de estrutura de dados e interface com o usuário. No entanto, o maior impecilho para o aproveitamento da implementação pode estar no preparo do entendimento da linguagem pelo usuário, que demandaria grande esforço e elevada formação acadêmica. A despeito deste fato, resta a possibilidade de aproveitá-la em situações que demandariam pouca ou nenhuma interferência humana externa.

A demanda por memória tem seu principal alvo no tamanho da pilha que deve ser mantida para o controle do processo de derivação. O protótipo desenvolvido utiliza uma pilha que é implementada na forma de matriz para manutenção das informações. O aumento do número de pilhas usadas para este fim representa um refinamento do processo de controle que pode ser aproveitado, como na primeira ordem. Grande número de informações deve ser mantido durante o processamento de uma consulta, e a técnica desenvolvida por David Waren [Ait-90] possibilita eliminar uma série delas quando ocorre uma unificação. Outros melhoramentos levam em conta o estado da pilha de controle, considerando os apontadores gerados quando houve a última chamada dentro de uma cláusula e representam pequenos detalhes de implementação que devem também ser considerados. Otimizam o uso de memória e conseguem ao mesmo tempo uma melhora no tempo de processamento.

No PROLOG, algumas oportunidades de melhorar o tempo de processamento exploraram o algoritmo da unificação. Vários testes dentro do algoritmo são feitos e estes devem se acomodar às estruturas propostas para representar e armazenar os termos. O "teste de ocorrência" e o operador "cut", usado como forma de eliminar tentativas indesejadas de unificação, fazem parte destas opções. A implementação do operador "cut" é uma possibilidade aberta em nosso protótipo.

A árvore de unificação formada na busca dos unificadores para dois termos, é um dos pontos do interpretador λ-PROLOG em que se pode prever melhorias na busca por eficiência. Há segmentos que se repetem com muita frequência, e heurísticas podem ser incluídas para acelerá-lo.

Grande parte da proposta de implementação fornecida com este protótipo pode sofrer reformulações em detalhes. No terceiro passo da árvore de unificação do exemplo 5.4.2, por exemplo, o par fr, $\langle L_1$, (cons 2 nil)$\rangle$ gasta 4 passos para ser completamente unificado e eliminado do conjunto discórdia. Os unificadores parciais são :

$L_1/cons\ H_1\ H_2$
$H_1/2$
H_2/nil

A composição dos três unificadores resulta em L_1 / cons 2 nil. Lógico, como [Huet75] demonstra, que nesta ocasiões basta formar o unificador L_1 / cons 2 nil e muito tempo seria ganho. Esta heurística de verificar os pares ⟨ X,F ⟩ onde X é uma variável e F um termo, fornece de modo direto a substituição {(X,F)}, desde que X não ocorra livre em F.

Além disso, certos ramos da árvore de unificação são repetições de outros ramos já verificados e o aproveitamento do trabalho já executado poderia ser feito. Veja como alguns segmentos do ramo 3 do exemplo 5.4.2 são repetições de outras unificações ocorridas em ramos já estabelecidos.

Se as ramificações resultassem em termos α-conversíveis, cujas unificações parciais devessem ser refeitas, algum mecanismo de representação destes termos, como aquela de Bruijn(1972) , poderia dispensar a repetição do processo. Nesta forma de representação os termos são expressos pelas aplicações e abstrações, dispensando o nome da variáveis.

Aperfeiçoamentos na implementação deverão ser os passos seguintes na tentativa de se ter um bom interpretador λ-PROLOG. Nosso protótipo foi implementado na linguagem SNOBOL, por facilidades de trabalho com cadeias. Naturalmente não é a linguagem indicada para um interpretador que pretende ser eficiente. No entanto, ela é específica e compacta e permitiu, com um relativamente pequeno número de linhas de código, trabalhar o sistema.

Ao se propor estas novas melhorias na implementação do sistema, direcionou-se a uma aproximação de um modelo chamado máquina abstrata de Warren (WAM) [Nada89], visto no capítulo 2. Para atingir este paradigma deve-se considerar ainda do uso de código compilado como elemento de otimização, o que não feito nesta fase do nosso trabalho.

5.5.2 Base de Dados e Tipos

O interpretador construido pressupõe a existência da base de conhecimentos e de tipos codificados de acordo com a sintaxe da gramática estabelecida. Sintaticamente, a construção das cláusulas não apresenta grandes dificuldades. No entanto a semântica da tipificação nem sempre é de fácil compreensão para o usuário. São tarefas do interpretador, por exemplo , aferir que os dois lados do sinal :- na cláusula sejam proposições. Isto é, sejam do tipo t. Em especial que o lado esquerdo seja um átomo e que o direito seja uma fórmula objetivo. Outras exigências de tipo são, por exemplo, que a fórmula no escopo de um símbolo Existe seja do tipo $\alpha \rightarrow t$. Há enfim, um conjunto de regras vinculadas à adequação dos tipos para a formação correta da cláusula que devem ser entendidas e obedecidas para que a base seja construida corretamente.

Este entendimento é pesado para o usuário. Para atenuar tal trabalho sugeriu-se aproveitar a estrutura da árvore de derivação da fórmula, produzida no rastreio da gramática e tentar obter, através dela, o tipo de todas as variáveis não fundamentais existentes. Assumindo que sintaticamente a cláusula esteja correta o interpretador é capaz de gerar o tipo de grande parte das variáveis de ordem superior a um. Neste ponto o sistema poderá arguir se este era mesmo o tipo pretendido, interagindo com o usuário, e conferir qual a situação desta constante ou variável no cadastro de tipos, ou tomar outra atitude de acordo com a situação. O uso da forma de representação de Bruijn(1972), da árvore de derivação vista na seção 5.2 e da árvore de derivação de tipos de [Nune89] auxiliam a determinação dos tipos não fundamentais de uma cláusula sintaticamente bem formada.

5.5.3 Usabilidade

A obtenção do ambiente que possibilitou um mecanismo formal de tratar as noções de ordem superior dentro da programação lógica gerou pesquisas diversas, algumas com a implementação de interpretadores. Uma vez estabelecido o alfabeto, fórmulas bem formadas e conversões, a base lógica foi definida e propriedades foram provadas. A derivação de uma fórmula objetivo foi definida. Provou-se a consistência e completude da P-derivação com base em uma base de conhecimento ou programa P. Com mecanismos corretos , embora que com implementações ainda experimentais, o aproveitamento das potencialidades do formalismo começou a ser pesquisado.

A incorporação das noções de função de ordem superior e dos λ-termos, abriu perpectivas de resolver problemas até então resolvidos de forma menos elegante ou não resolvidos. Como consequência de seu uso ressalta-se de imediato e de forma ilustrativa a definição de uma função similar à função maplist do LISP.

Outra potencialidade foi obtida especialmente pelo fato de que a principal estrutura tratada na linguagem são os λ-termos. Estes são especialmente convenientes para tratar tarefas que envolvem manipulação de objetos contendo variáveis ligadas. A diferença deste tratamento pela lógica de ordem superior aumenta à medida que for usada uma estrutura livre de nomes para representar os λ-termos.

Espera-se também que esta estrutura de representação mais explícita (do que aquela da primeira ordem) possa estimular e simplificar o processo de provas de teoremas de ordem superior. Alguns usos desta lógica foram publicados demonstrando aplicação [Mill85] da linguagem. Ainda [Mill85] mostrou uma metodologia computacional lógica para semântica e sintaxe que permite manipular expressões de primeira ordem e em [Mill86] Miller propôs uma teoria de módulos para programação lógica alterando, basicamente, a cláusula definida e permitindo consultas com a sintaxe:

$$G := T \mid A \mid G_1 \vee G_2 \mid G_1 \wedge G_2 \mid \exists x G \mid D \supset G$$

Esta forma de consulta permite visualizar uma derivação de módulos sendo que o símbolo $\supset$ fornece uma semântica desejável no mecanismo de derivação.

Capítulo 6

Algoritmo UNIF para a Unificação

6.1 Introdução

Excluindo o procedimento de inicialização, que contempla a conferência gramatical das sentenças e a tipificação, a implementação do sistema para se obter uma prova sobre um programa P na lógica de ordem superior contém procedimentos parecidos com aqueles da resolução que é o mecanismo de inferência usado na primeira ordem. A forma de controle é muito parecida, os métodos de encadeamento e de retroencadeamento são semelhantes e até mesmo as técnicas de armazenamento de informações em pilhas podem ser aproveitados de um sistema para outro.

Pesquisas foram feitas com o objetivo de otimizar os processos existentes no sistema de resolução de primeira ordem. Algumas delas se aplicam também ao domínio da lógica de ordem superior, com pequenas adaptações.

As lacunas ou deficiências de nosso protótipo residem nas mesmas propostas de pesquisas levantadas para a primeira ordem e já citadas no capítulo anterior, ou seja: representação para os λ-termos, formas de controle para ocupar menos memória, e compartilhamento de memória na representação dos termos.

Com relação à representação dos termos [Nada89] apresentou uma técnica que permite examinar sua estrutura interna ao mesmo tempo que permite facilitar a β-redução. As pesquisas neste sentido estão apenas começando e vários tópicos se apresentam como temas de investigação. Inclue-se neste conjunto a possibilidade de se colocar alguns módulos compilados dentro do sistema.

No entanto algumas partes do sistema são específicas para a nova linguagem. Como tal, evidenciam-se dois aspectos:

- A unificação que na LOS pode apresentar mais de um unificador, ao contrário da lógica de primeira ordem onde existe um único unificador mais geral.

- O mecanismo de retroencadeamento mais complexo da LOS que conta com mais uma razão para acontecer, em virtude da multiplicidade de unificadores.

A unificação é um procedimento muito mais complexo do que aquele da primeira ordem. Até o momento, este fato representa um desestímulo para que haja avanços no uso da lógica de ordem superior, quer por seu entendimento quer pelas dificuldades computacionais advindas de sua complexidade. O algoritmo MATCH é pesado computacionalmente e deve-se considerar seriamente qualquer chance de se atenuar sua carga sobre o tempo de processamento. Para uma mensuração do trabalho computacional envolvido pode-se observar a árvore de unificação apresentada no Apêndice B.

O mecanismo de retroencademento para a LOS, em termos de implementação, consiste, grosseiramente falando, de um laço cujo contador é o número de unificadores encontrados, dentro do qual o prosseguimento da derivação é feito. Como um laço, sua otimização tem como parâmetro principal o contador. Desta forma, qualquer ganho em termos de diminuição do números de unificadores poderia representar considerável economia de esforço computacional. No entanto a redução do número de unificadores seria apenas correta se algum mecanismo previsse que a escolha de determinados unificadores levaria a derivação ao insucesso. Esta linha de pesquisa não foi investigada neste trabalho.

Com o propósito de diminuir o trabalho computacional da unificação aborda-se mais detalhadamente, neste capítulo, o algoritmo da figura 4.3 buscando especificar a *imitação* e a *projeção* a ponto de sugerir algoritmos mais eficientes na busca dos unificadores de um par de discórdia D. Ao mesmo tempo, a análise facilita a visualização do processo e dos conjuntos unificadores obtidos, permitindo fornecer subsídios para se escolher unificadores com chance de sucesso na derivação.

A análise dos métodos tem como base os algoritmos de simplificação (SIMPL) e unificação (MATCH) expostos nas figuras 4.2 e 4.3, respectivamente.

Parte resultante desta abordagem foi exposta em [Luda92] e [Ludb92] onde é proposto o algoritmo UNIF como alternativa a MATCH.

Além do exemplo 5.4.2 que ilustra o uso dos dois algoritmos, e de outros contidos neste trabalho, um exemplo é fornecido no Apêndice B. Além de fornecer uma visão de toda a árvore de unificação, ele contém maior número de detalhes que favorecem seu acompanhamento e entendimento.

Para o estudo de MATCH e a proposição de novos algoritmos considera-se que o primeiro par da esquerda do conjunto discórdia sempre é tomado para dar sequência ao processo de unificação. Isto é, não há função de seleção para pares dentro do conjunto.

Adota-se a caracterização de "unificador parcial" para a *composição* dos unificadores obtidos até determinado nó da árvore de unificação, que pode ser ainda chamado de unificador para o ramo. Além disto, o unificador que permite a passagem de um nó para outro é chamado de unificador do arco.

6.2 Análise da Projeção e da Imitação

Considera-se que $\langle F^1, F^2\rangle$ é um par de discórdia com duas fórmulas de mesmo tipo, uma flexível e outra rígida assim descrito:

$$\langle F^1, F^2\rangle = \langle \text{ f } a_1^1 \cdots a_r^1, C\ a_1^2 \cdots a_s^2\rangle$$

Lema 6.1 *Uma sequência de imitações aplicadas a qualquer par de discórdia*

$$\langle f a_1^1 \cdots a_r^1, C\ a_1^2 \cdots a_s^2\rangle$$

produz um ramo de sucesso cujo unificador parcial é da forma

$$\theta_n = f/w_1 \cdots w_r\ C\ a_1^2 \cdots a_n^2\ (h_{n+1} w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)$$

e o número total de arcos np(f) para atingir este nó da árvore de unificação é definido recursivamente por:

- *np(f) = 1 se r=0 e*
- *np(f) = 1 + $\sum_{i=1}^{r} np(a_i^1)$ se r > 0*

Demonstração: Por indução na altura da árvore de derivação. No primeiro arco da imitação tem-se o unificador $f/C(h_1 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)$, pela definição de imitação para $r \geq 1$, confirmando hipótese com um passo. Este unificador produz um conjunto discórdia com r pares da forma $\langle (h_i a_1^1 \cdots a_r^1), a_i^2\rangle$, $1 \leq i \leq r$. Se r = 0, a substituição é f/F^2 e o conjunto discórdia resultante é vazio. Neste caso a unificação é uma unificação de primeira ordem sobre átomos, especificamente de uma variável com um termo. Se a cabeça de F^2 for uma variável ligada, então F^1 e F^2 não teriam o mesmo tipo e a pressuposição inicial de igualdade de tipos não estaria satisfeita.

Para a segunda etapa da unificação, tomando-se o primeiro par à esquerda do conjunto discórdia gerado para unificar, tem-se a imitação aplicada ao par $\langle (h_1 a_1^1 \cdots a_r^1), a_1^2\rangle$ que gera após np(h_1) passos, pela hipótese de indução, o unificador $h_1/\lambda w_1 \cdots w_r.a_1^2$. A composição deste unificador com o primeiro, resulta na substituição

$$\text{f}/\lambda w_1 \cdots w_r C\ a_1^2\ (h_2 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r),$$

o que está de acordo com o lema. De posse deste unificador parcial que atualiza o conjunto discórdia, deve-se unificar o conjunto discórdia restante. Supondo ter havido imitações até o n-ésimo argumento de F^1, a situação no nó é a seguinte:

$$\text{D} = \{\langle (h_{n+1} a_1^1 \cdots a_r^1), a_{n+1}^2\rangle, \cdots \langle (h_s a_1^1 \cdots a_r^1), a_s^2\rangle\}$$

$$\theta_n = \{f/\lambda w_1 \cdots w_r\ C\ a_1^2 \cdots a_n^2\ (h_{n+1} w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)\}$$

em que θ_n é o unificador parcial obtido até o momento e D é o conjunto discórdia resultante. O próximo passo de unificação por imitação, será feito para o par $(\langle h_{n+1} a_1^1 \cdots a_r^1), a_{n+1}^2\rangle$. Pelo próprio lema o unificador desta etapa é $h_{n+1}/\lambda w_1 \cdots w_r . a_{n+1}^2$. Para que isto ocorra utilizando-se a hipótese de indução espera-se que o par acima, ou seja,

$$\langle (h_{n+1} w_1 \cdots w_r), a_{n+1}^2\rangle. \tag{6.1}$$

obedeça as pressuposições de tipo e de par flexível-rígido. A imposição de tipos iguais é satisfeita quando da geração de h_{n+1}. Já a segunda nem sempre é satisfeita pois pode existir a_i^2 flexível, tornando o primeiro par de D flexível-flexível. Este caso foi tratado na seção 4.4 onde se propôs uma substituição do tipo $\lambda w_1 \cdots w_r . y$ para as duas variáveis flexíveis, com y do tipo adequado. Propõe-se aqui que o procedimento continue com uma simplificação das duas cabeças que se tornaram iguais, e emparelhe os pares de seus argumentos, como a rotina SIMPL trata os pares rígido-rígido. O efeito desta simplificação e aquele da substituição proposta acima, é o mesmo. Logo não se leva em consideração o aparecimento de pares flexível-flexível admitindo-se que se eles aparecerem nas imitações sucesssivas serão do mesmo tipo e podem ser simplificados como pares rígido-rígido de cabeças iguais.

Esta etapa consumiu $1 + \sum_{i=1}^{n} \mathrm{np}(a_i \text{ de } a_{n+1}^1)$, onde u é o número de argumentos de a_{n+1}^1, admitindo-se novamente a hipótese de indução. A composição deste unificador obtido com θ_n gera

$$\theta_{n+1} = f/\lambda w_1 \cdots w_r\ C\ a_1^2 \cdots a_{n+1}^2 (h_{n+2} w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r).$$

O número de passos anterior era $np_n(f) = 1 + \sum_{1=1}^{n} np(a_i^1)$ que somados a $1 + \sum_{i=1}^{n} np(a_i^1 \text{ de } a_{n+1}^1)$, resultam em

$$np_{n+1} = 1 + \sum_{i=1}^{n+1} np(a_i^1),$$

verificando-se o lema.

A proposta de se tratar os pares flexível-flexível da maneira explicada acima é inovadora. O algoritmo MATCH está proposto apenas para pares flexível-rígido e seu resultado consiste de um unificador onde o conjunto D pode ser vazio, conter o indicador **F** (de falso, caso em que não houve unificação) ou contém apenas pares flexível- flexível. Neste ponto, como na parte que se segue, a abordagem de nosso método obterá sempre como resultado um unificador com $D = \emptyset$ ou $D = \mathbf{F}$.

Na projeção, segundo o algoritmo MATCH, quando um dos argumentos de F^1 é funcional, o unificador não gera apenas $\lambda w_1 \cdots w_r.w_i$ como unificador do arco mas

$$\lambda w_1 \cdots w_r.w_i(h_1^i w_1 \cdots w_r) \cdots (h_t^i w_1 \cdots w_r)$$

onde t é o número de variáveis ligadas por λ no argumento a_i^1 projetado. Na FNU todas as variáveis abstraidas aparecem no conjunto de variáveis de ligação.

Este unificador existirá se houver unificação entre a_i^1 aplicado a

$$\xi = ((f/h_1^k)F^1, (f/h_2^k)F^1, \cdots, (f/h_t^k)F^1)$$

e F^2, isto é, a_i^1 aplicado a uma série de t argumentos $(f/h_i^k)F^1$, tantos quantas forem as variáveis de ligação[1] de F_1.

Doravante representa-se esta aplicação por $\xi(a_i^1)$ com o objetivo de facilitar a leitura. Para argumentos a_i^1 não funcionais, não haverá variáveis de ligação e

$$\xi(a_i^1) = a_i^1.$$

Sujeito pois a condição de que exista um ou mais unificadores para $\langle \xi(a_i^1), F^2 \rangle$ o unificador parcial inicial obtido por projeção será então

$$\{\ f/\lambda w_1 \cdots w_r\ C[\ w_i(h_1^i w_1 \cdots w_r) \cdots (h_t^1 w_1 \cdots w_r)\ ] \cdots \}.$$

representando a projeção do i-ésimo argumento de f.

Lema 6.2 *Uma projeção de a_i^1 de F^1 sobre F^2, após imitações sucessivas sobre os k-1 a_i^1 anteriores, $(k \geq 2)$, gera unificadores parciais que têm a seguinte forma:*

$$\{\ \phi(f/w_1 \cdots w_r C\ a_1^2 \cdots a_{k-1}^2\ (w_i\ (h_1^k w_1 \cdots w_r) \cdots (h_t^k w_1 \cdots w_r)) \cdots (h_s w_1 \cdots w_r) \cup \phi\ \}$$

em que $\phi = unif(\langle \xi(a_i^1), a_k^2 \rangle)$, unif representa um dos conjuntos de unificadores para o par, k a posição que $(w_i(h_1^k w_1 \cdots w_r) \cdots (h_t^k w_1 \cdots w_r))$ ocupou como argumento na substituição para f e t, o número de variáveis ligadas de a_k^2, não iguais à sua cabeça.

A segunda parte do conjunto pode ser vazia ou ser um conjunto de substituições, indicando se houve sucesso na projeção.

Demonstração: Após a primeira imitação, as projeções resultam nos seguintes unificadores para os arcos correspondentes a cada i, $(1 \leq i \leq r)$, com a aplicação da regra de projeção admitindo-se a igualdade de tipos e o tipo de w_i como funcional com t argumentos:

$$h_1/\lambda w_1 \cdots w_r.w_i(h_1^1 w_1 \cdots w_r) \cdots (h_t^1 w_1 \cdots w_r)$$

pois o primeiro par do conjunto de discórdia após a primeira imitação é $\langle (h_1 a_1^1 \cdots a_r^1), a_i^2 \rangle$.

[1] $(f/h_i^k)F^1$ representa F^1 com f substituido por h_i^k

a) Supondo t = 0, o unificador parcial do ramo obtido pela combinação de cada unificador com o unificador inicial

$f/\lambda w_1 \cdots w_n C\ (h_1 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)$,

será

$$f/\lambda w_1 \cdots w_n C\ w_i\ \ (h_2 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r) \qquad (6.2)$$

e a aplicação do unificador parcial assim obtido ao conjunto discórdia inicial, seguido de β-redução gera o seguinte conjunto discórdia

$$\{\ \langle a_i^1, a_1^2\rangle\ \langle (h_2 a_1^1 \cdots a_r^1), a_2^2\rangle, \cdots (h_s a_1^1 \cdots a_r^1), a_s^2\rangle\ \}$$

cujo primeiro par deve ser unificado para que (6.2) seja um unificador parcial com sucesso. No entanto esta unificação gera uma nova árvore que terá ou não seus unificadores. Em caso de sucesso na unificação, seus unificadores serão compostos com o unificador parcial anterior. A composição não altera (6.2) pois na unificação do par $\langle a_i^1, a_1^2\rangle$ não existirão substituições para as variáveis w_i e tampouco para h_i.

Assim sendo o lema fica comprovado deste que no passo 2 não existe a sequência $a_i^1 \cdots a_i^{k-1}$ precedendo w_i como argumentos de C, tampouco argumentos h_1^k para w_i.

A composição dos unificadores parciais é portanto o conjunto formado por (6.2) unido aos unificadores de $\langle a_i^1, a_1^2\rangle$.

Após estas duas etapas de unificação o conjunto discórdia será da forma

$$\langle (h_2\ \phi(a_1^1 \cdots a_r^1)), a_2^2\rangle \cdots \langle (h_s\ \phi(a_1^1 \cdots a_r^1)), a_s^2\rangle$$

em que ϕ representa um dos unificadores para o par de discórdia $\langle a_i^1, a_1^2\rangle$ e $\phi(a_1^1 \cdots a_r^1)$ representa sua aplicação à sequência de argumentos.

b) Se $t \neq 0$, o unificador em (6.2), é, no passo 2,

$$f/\lambda w_1 \cdots w_n C\ (w_i[(h_1^1 w_1 \cdots w_r) \cdots (h_t^1 w_1 \cdots w_r))]$$

$$(h_2 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r). \qquad (6.3)$$

Este unificador é obtido pela própria definição de projeção aplicada a qualquer dos argumentos a_i^1. A aplicação do unificador assim obtido ao conjunto

discórdia anterior da árvore, seguido de β-redução gera o seguinte conjunto discórdia

$$\langle a_i^1(h_1^1 a_1^1 \cdots a_r^1) \cdots (h_t^1 a_1^1 \cdots a_r^1), a_1^2 \rangle \langle (h_2 a_1^1 \cdots a_r^1), a_2^2 \rangle, \cdots \langle (h_s a_1^1 \cdots a_r^1), a_s^2 \rangle.$$

O lado esquerdo do primeiro par é $\xi(a_i^1)$, pois admite-se que o argumento é funcional. Deste modo o conjunto dicórdia é

$$\langle \xi(a_i^1), a_1^2 \rangle \langle (h_2 a_1^1 \cdots a_r^1), a_2^2 \rangle, \cdots \langle (h_s a_1^1 \cdots a_r^1), a_s^2 \rangle.$$

A unificação do primeiro destes pares de discórdia gera uma nova árvore de unificação que pode ou não atingir o sucesso. Seja ϕ qualquer destes unificadores. Existindo pelo menos um deles suas substituições comporão com o unificador parcial anterior (6.3). Esta composição altera apenas variáveis livres que ainda não estão à esquerda de substituições em (6.3), pois estas são as mesmas de F^1 e F^2, exceto f, e que ainda não têm substituções em (6.3) e as novas variáveis h_t^1. A composição agrega a (6.3) os novos unificadores obtidos com a unificação deste primeiro par, obtendo-se então

$$\{f/\lambda w_1 \cdots w_n C\ (w_i\ \phi([(h_1^1 w_1 \cdots w_r) \cdots (h_t^1 w_1 \cdots w_r))] \\ (h_2 w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)) \cup \phi\} \quad (6.4)$$

Observa-se ainda que neste caso o argumento i de F^1, objeto da projeção, não terá mais variáveis ligadas, no conjunto discórdia. As variáveis ligadas que apareciam no corpo do argumento foram substituidas por α-conversões de F^1. Ou seja, o primeiro termo do primeiro par de discórdia constituiu-se de t aplicações de F^1 α-convertido àquele argumento.

Com a unificação do primeiro par, para esta nova etapa, o novo conjunto de discórdia não o conterá mais. Porém os pares restantes sofrerão a aplicação dos unificadores ϕ obtidos para o (primeiro) par considerado, tornando-se

$$\langle (h_2\ \phi(a_1^1 \cdots a_r^1)), \phi(a_2^2) \rangle, \cdots, \langle (h_s\ \phi(a_1^1 \cdots a_r^1)), \phi(a_s^2) \rangle.$$

que pode melhor ser expresso por

$$\phi(\{\langle (h_2\ a_1^1 \cdots a_r^1), \phi(a_2^2) \rangle, \cdots, \langle (h_s\ a_1^1 \cdots a_r^1), a_s^2 \rangle\}).$$

A composição de ϕ com (6.3) neste caso provocará alterações nas variáveis h_i^1 e acrescentará, como visto, ao unificador parcial (6.3) as novas substituições obtidas. Observa-se então a correção do lema para a primeira projeção, mesmo com argumentos funcionais.

Na etapa correspondente à projeção do (n+1)-argumento de F^1 após n imitações sobre argumentos de F^1 o unificador parcial (pelo lema 1) bem como o conjunto de discórdia são os seguintes:

$$\theta_n = \{f/\lambda w_1 \cdots w_r\ C\ a_1^2 \cdots a_n^2\ (h_{n+1} w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)$$

$$\mathrm{D} = \{\langle (h_{n+1} a_1^1 \cdots a_r^1), a_{n+1}^2 \rangle, \cdots \langle (h_s a_1^1 \cdots a_r^1), a_s^2 \rangle\}$$

O próximo par a ser unificado é $\langle (h_{n+1} a_1^1 \cdots a_r^1), a_{n+1}^2 \rangle$ por meio da projeção dos a_i^1 sobre a_i^2. O unificador para este arco i tem a seguinte forma:

$$h_{n+1}/\lambda w_1 \cdots w_n (w_i (h_1^{n+1} w_1 \cdots w_r) \cdots (h_t^{n+1} w_1 \cdots w_r))$$

que combinado com o anterior gera

$$f/\lambda w_1 \cdots w_n C\ a_i^2 \cdots a_n^2\ (w_i (h_1^{n+1} w_1 \cdots w_r) \cdots (h_t^{n+1} w_1 \cdots w_r))$$
$$(h_s w_1 \cdots w_r). \quad (6.5)$$

cujo sucesso parcial dependerá da unificação do par $\langle \xi(a_i^1), a_i^2) \rangle$.

A diferença para os casos analisados em a) e b) é que parte-se de um unificador parcial que já contém n argumentos a_i^1 para a substituição de f.

Tudo funciona como nos casos a) e b) anteriores, com exceção dos n argumentos já colocados na substituição para f. Estes, quando da composição dos unificadores, sofrerão aplicação dos unificadores obtidos para o par de discórdia que resultou após a projeção.

Para se eliminar de D o primeiro par de discórdia é necessário que ele seja unificável. Seja ϕ algum unificador para o par. Acontecendo isto, os unificadores obtidos devem ser aplicados a D novamente, e compostos com o unificador anterior, chegando-se a novos unificadores parciais. Após esta etapa de unificação o unificador parcial será uma aplicação de ϕ à equação (6.5), acrescido do próprio ϕ, ou seja

$$\{\phi(f/\lambda w_1 \cdots w_n C\ a_i^1 \cdots a_n^1\ (w_i (h_1^{n+1} w_1 \cdots w_r) \cdots (h_t^{n+1} w_1 \cdots w_r))$$
$$(h_s w_1 \cdots w_r)) \cup \phi\}$$

que é a previsão do lema.

Lema 6.3 *Qualquer unificador parcial θ_n de f, obtido por meio do uso dos lemas 1 e 2, depois de obtidos os unificadores ϕ, pode ser transformado em um unificador de $\langle F^1, F^2 \rangle$ pela substituição de cada termo cuja cabeça é h_i, $(1 \leq i \leq r)$ existente no unificador parcial por seu correspondente $\theta_n(a_i^2)$. O termo $\theta_n(a_i^2)$ representa a aplicação do unificador parcial a a_i^2*

Demonstração: Depois da aplicação do lema 1, o unificador parcial e o conjunto discórdia restante são, respectivamente

$$\theta_n = f/w_1 \cdots w_r C a_1^2 \cdots a_n^2\ (h_{n+1} w_1 \cdots w_r) \cdots (h_s w_1 \cdots w_r)$$

$$D = \{ \langle (h_{n+1} a_1^1 \cdots a_r^1), a_{n+1}^2 \rangle, \langle (h_s a_1^1 \cdots a_r^1), a_s^2 \rangle, \}$$

Neste caso $\phi = \emptyset$. Para o caso do lema 2, θ_n e D são:

$$\theta_n = \{\phi(f/\lambda w_1 \cdots w_r C\ a_i^2 \cdots a_{n-1}^2\ (w_i(h_1^{n+1} w_1 \cdots w_r) \cdots (h_i^n w_1 \cdots w_r)) \cdots (h_s w_1 \cdots w_r)), \phi\}$$

$$D = \{\langle (h_{n+1}\ \phi(a_1^1 \cdots a_r^1)), \phi a_2^2 \rangle \cdots \langle (h_s\ \phi(a_1^1 \cdots a_r^1)), \phi a_s^2 \rangle\ \}$$

Em qualquer das duas situações, o conjunto discórdia restante é constituido por sequências $\langle (h_{n+1}\ \phi(a_1^1 \cdots a_r^1)), a_2^2 \rangle \cdots \langle (h_s\ \phi(a_1^1 \cdots a_r^1)), a_s^2 \rangle$ sendo $\phi = \emptyset$ no caso do lema 1.

Pelo lema 1, qualquer destes h_i tem como unificador o termo

$$\{\phi(h_i/\lambda w_1 \cdots w_r a_i^2)\}.$$

Este unificador combinado com o anterior reduz o conjunto de discórdia eliminando o termo de cabeça h_i de D e transforma o unificador parcial em

$$\{\phi(f/\lambda w_1 \cdots w_r C\ a_i^2 \cdots a_n^2\ (w_i\ (h_1^n w_1 \cdots w_r) \cdots (h_i^n w_1 \cdots w_r))\ a_{n+1}^2 \cdots (h_s w_1 \cdots w_r)) \cup \phi\}$$

Iterativamente chega-se ao proposto pelo lema.

6.3 Obtenção dos Unificadores "Principais"

Como consequência dos lemas anteriores pode-se ter a partir da primeira imitação, unificadores que são F^2 abstraida de $\lambda w_1 \cdots w_n$ com qualquer w_i... substituindo um dos argumentos de F^2.

Método similar ao usado nos lemas 1, 2 e 3 permite mostrar que os termos w_i podem substituir qualquer argumento i de $\lambda w_i \cdots w_r.F^2$ tornando-se um unificador para o par, independente da profundidade do argumento dentro do unificador parcial. Naturalmente a profundidade máxima da fórmula é dada pela ordem do tipo de f.

Os três lemas anteriores e suas extensões levam à obtenção dos seguintes unificadores para f,

1. $\lambda w_i \cdots w_r.F^2$ no qual não há variáveis w_i como argumentos
2. $\lambda w_i \cdots w_r.C\ \phi(a_i^2 \cdots w_i[\cdots h_j^k \cdots] \cdots a_s^2)$ com apenas um $w_i[\cdots h_j^k \cdots]$ substituindo um dos argumentos de F^2 , $(1 \leq j \leq t)$ e ϕ = outras substituições obtidas para as demais variáveis livres.
3. $\lambda w_i \cdots w_r.C\ \phi(a_i^2 \cdots\cdots a_s^2)$ com apenas um w_i substituindo recursivamente um dos argumentos de a_i^2 e ϕ = outras substituições obtidas para as demais variáveis livres.

O universo dos unificadores de $\langle F^1, F^2 \rangle$ pode conter ainda qualquer combinação dos unificadores acima listados.

Para melhor entendimento considere-se ϕ como condicionador para a existência de um dos unificadores listados em 2 e 3 acima. Ou seja ϕ estipula uma lista de substituições necessárias para que exista, por exemplo, o seguinte unificador:

$$f/\phi(\lambda w_i \cdots w_r.C\ \phi(a_i^2 \cdots w_i[\cdots h_j^k \cdots] \cdots a_s^2))$$

Como pode haver w_{i_s} em qualquer posição, a árvore de unificação permite observar que se existirem dois unificadores com w_i em posições diferentes então pode-se obter um unificador que contenha os dois w_i em suas posições correspondentes desde que seja possível uma combinação entre os unificadores ϕ correspondentes à unificação dos pares $\langle a_i^1, a_k^2 \rangle$. Em outras palavras, se os condicionadores puderem funcionar simultaneamente. O conceito de "unificador mais geral" está sendo usado nesta "acomodação". Dois unificadores podem ser combinados se houver um unificador mais geral que os absorva.

Esta observação conduz à seguinte proposição:

Proposição 6.1 *Sejam* $F^1 = f\ a_1^1 \cdots a_r^1$ *e* $F^2 = C\ a_1^2 \cdots a_s^2$. *Então tem-se*

- $f/\lambda w_1 \cdots w_r C\ a_1^2 \cdots a_s^2$ *é um unificador para* f.
- $\{\ f/\phi(\lambda w_1 \cdots w_r C\ a_1^2 \cdots w_i[\cdots h_i^k \cdots] \cdots a_s^2 \cup \phi\ \}$ *é um unificador para* f *se* $\phi = unif(\langle a_i^1, a_k^2 \rangle)$.

A prova vem das observações anteriores sobre os lemas 1, 2 e 3.

A proposição a seguir indica como combinar unificadores.

Proposição 6.2 *Sejam os unificadores entre* $\langle a_i^1, a_k^2 \rangle$ *e* $\langle a_j^1, a_l^2 \rangle$, $(l \neq k)$ *representados por*

$$\phi_1 = \{v_1/tv_1, v_2/tv_2 \cdots v_m/tv_m\ \}\ e$$
$$\phi_2 = \{u_1/tu_1, u_2/tu_2 \cdots u_l/tu_l\}.$$

Seja ainda definida a pseudo-composição dos dois unificadores como $\phi_1 \circ \phi_2$, *que é a composição sujeita a não eliminação das substituições* u_i/tu_i *quando* $u_i = v_j$ *para qualquer* i *e* j. *Estes pares de substituição devem ser substituidos por uma substituição mais geral formada a partir das duas. Se isto for possível então tem-se o unificador*

$$\{\phi_2(\phi_1(f/\lambda w_1 \cdots w_r C\ a_i^2 \cdots w_i[\cdots h_i^k \cdots] \cdots w_j[\cdots h_i^l \cdots] \cdots a_s^2)) \cup \phi_1 \circ \phi_2\ \}.$$

Esta proposição está baseada na conceituação de que ϕ_1 e ϕ_2 são condições para que existam os unificadores $w_i[\cdots h_i \cdots]$ nas posições l e k respectivamente. Subentende-se também que a pseudo-composição é comutativa pois representa apenas a união dos dois unificadores.

Demonstração: Já que $l \neq k$, isto é tem-se projeções sobre argumentos diferentes de a_i^2, pode-se supor $l > k$. Na árvore de unificação este fato equivale a dizer que foi gerado o unificador

$$\{(\phi_1(f/\lambda w_1 \cdots w_r C\ a_i^2 \cdots w_i[\cdots h_i^k \cdots] \cdots a_s^2)) \cup \phi_1\ \}$$

em primeiro lugar, após uma série de imitações de a_i^1, com a condição de que $unif(\langle a_i^1, a_k^2 \rangle) = \phi_1 \neq \mathbf{F}$. Este unificador tem como v_i apenas variáveis livres do par a ser unificado, ou seja $\langle a_i^1, a_k^2 \rangle$, nas suas substituições, além dos h_i^k específicos. Se houve unificação, estes unificadores serão aplicados a todo o conjunto de discórdia restante e estas variáveis livres não mais aparecerão nas unificações posteriores da árvore.

De forma semelhante uma projeção sobre a posição l, precedida apenas de imitações, gera o unificador

$$\{\phi_2(f/\lambda w_1 \cdots w_r C\ a_i^2 \cdots w_j[\cdots h_i^l \cdots] \cdots a_s^2)) \cup \phi_2\ \}$$

sujeito a existência de $\phi_2 = unif(\langle a_j^1, a_l^2 \rangle)$.

Em qualquer dos nós da árvore de unificação em que tenha havido a projeção k, seu unificador é aplicado aos termos a_i do par $\langle F^1, F^2 \rangle$ restantes no conjunto discórdia. Na pesquisa para se unificar este conjunto restante, nova projeção pode ser tentada, por exemplo, na posição l. Esta projeção só terá sucesso se sua condicionante (ϕ_2) não se conflitar com a condicionante anterior (ϕ_1). Isto ocorre apenas se as suas substituições satisfizerem às condições:

- $u_i \neq v_i, \forall i$
- Se $u_i = v_i$ então $tu_i = tv_i$ ou
- Se $u_i = v_i$ então existe substituição mais geral para tu_i e tv_i.

A partir do nó em que a projeção k tenha sido gerada, o conjunto discórdia restante não conterá mais as variáveis livres que formam as substituições de ϕ_1. Logo, em qualquer nó posterior em que for feita a projeção l, as novas variáveis v_i de ϕ_2 serão diferentes daquelas de ϕ_1. As variáveis comuns aos dois (condicionantes) são aquelas que formam o ramo comum até à projeção k. Pode-se observar que a única variável livre com substituição neste ramo é f. Destes dois fatos conclui-se que a variável comum aos dois unificadores é f. A argumentação é válida também para $k > l$. Na sequência do ramo em que foi feita a projeção k seguida de n-imitações ($0 \leq n \leq r$) a substituição u_i/tu_i correspondente a f, será sujeita a aplicação do unificador ϕ_2 que a torna um unificador mais geral, com um argumento $w_i[\cdots h_i^k \cdots]$ também na posição k.

Pode-se observar na árvore de unificação que todos os unificadores resultantes de duas projeções são obtidos nesta sequência de etapas.

A conclusão deste argumento é que não é necessário testar a árvore de unificação completa. Basta obter os unificadores parciais ϕ_1 e ϕ_2 e testar se os mesmos se "combinam".

Todo o desenvolvimento visto acima pode ser estendido para as r projeções que podem ser feitas sobre F^1.

No caso de p projeções ($1 \leq p \leq r$) o unificador será obtido pela "combinação" de p projeções únicas. Neste caso as p posições devem ser diferentes.

Considere-se agora os ramos da árvore de unificação cujo primeiro arco é uma projeção. A intenção com estas projeções iniciais é verificar a possibilidade de se unificar pares de discórdia da forma:

$$\langle a_i^1, F^2 \rangle.$$

Obedecidas as condições de tipo, os primeiros unificadores parciais serão:

$$f/\lambda w_1 \cdots w_r.w_i(h_1 w_1 \cdots w_r) \cdots (h_t w_1 \cdots w_r)$$

onde t tem o significado a ele atribuído anteriormente.

O primeiro conjunto discórdia resultante destas projeções é (já descrita no lema 3) constituída de t pares de discórdia da seguinte forma

$$\langle (h_i^k a_1^1 \cdots a_r^1), a_i^2 \rangle$$

O procedimento de obtenção dos unificadores para estas projeções segue recursivamente os conceitos estabelecidos nos lemas anteriores. No entanto, nestes casos não pode haver combinação pois a substituição para f tem uma unica possibilidade para $w_i[\cdots h_i^k \cdots]$

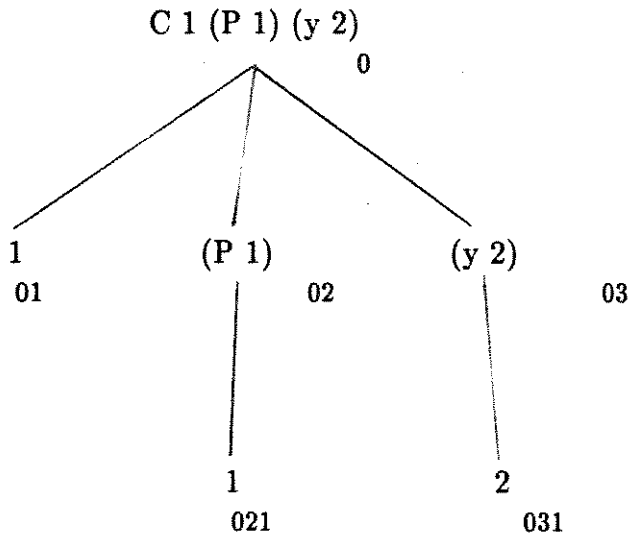
Tem-se agora em condições de visualisar um algoritmo geral para a unificação do par

$$\langle F^1, F^2 \rangle = \langle f\ a_1^1 \cdots a_r^1, C\ a_1^2 \cdots a_s^2 \rangle$$

de mesmo tipo e expressas na FNU.

Para facilitar o trabalho define-se a seguinte estrutura de árvore para a fórmula F^2: (1) O nó raiz é rotulado por F^2. (2) Recursivamente os nós filhos são os argumentos do nó pai.

Assim F^2 = C 1 (P 1) (y 2) é representada pela seguinte estrutura:



Os nós são enumerados da seguinte forma. A raiz (F^2) tem número 0 e é sub-entendida como argumento 0 (zero). Os nós subsequentes herdam a numeração do nó pai seguida do número sequencial do argumento que eles representam. Desta forma o número 031 para o nó representa que ele é o primeiro argumento do terceiro argumento da raiz. Em cada nó, o número que o segue indica o número de abstrações daquele termo.

Para cada arco, a existência de um rótulo i indica que seu nó está ligado à i-ésima abstração, sequencialmente até aquele arco da árvore.

A figura 6.1 contém o algoritmo que obtém os unificadores cujos ramos atingiram o sucesso e nos quais apenas um arco foi obtido por meio de projeção na árvore de unificação. Ele utiliza a estrutura de árvore acima proposta e os obtém por meio da unificação dos argumentos de F^1 com os nodos da árvore.

Note que nas duas condições do algoritmo da figura 6.1, a união com $unif$ resulta numa nova aplicação do algoritmo, como será visto adiante. Este conjunto que é uma nova árvore de unificação pode ter n unificadores, unificar sem substituição ou não unificar. Não existirá o unificador ij se não houver unificação do par sujeito a $unif$.

Observe que este algoritmo é obtido pela aplicação direta do lema 3. Segue um exemplo de aplicação do algoritmo ADF. Em primeiro lugar é colocada a substituição para f obedecendo os critérios do algoritmo. Seguindo o símbolo $\longrightarrow$ é colocado o par cuja unificação condiciona a existência daquela substituição para f. Se houver novos unificadores, (isto é ϕ), o procedimento continua a ser aplicado ao par condicionante como para ij = 1 03. As palavras Sucesso e Falha acusam o resultado da existência de ϕ.

Algoritmo ADF para obtenção dos unificadores "principais" para um par de discórdia $\langle F^1, F^2 \rangle$:
Considere dois valores i e j, onde i representa o índice de um argumento a_i^1 e j o número de um nó qualquer da estrutura de F^2. São unificadores para o par $\langle F^1, F^2 \rangle$, gerados por imitação ou projeção, para $(m \geq 0)$

- Para $1 \leq i \leq r$ e $\forall j$ ($j \in$ {nós da árvore F^2 }) o unificador ij é
  $\{\phi(f/\lambda w_1 \cdots w_r C\ a_1^2 \cdots w_i[\cdots h_i^j \cdots] \cdots a_s^2) \bigcup unif_m(\langle a_i^1,$ rótulo do nó j $\rangle)\ \}$
  O índice j representa qual termo da fórmula F^2 deve ser substituido por $w_i[\cdots h_i^j \cdots]$ e ϕ representa $unif_m$. Se i = 0, então $f/\lambda w_1 \cdots w_n C\ a_1^2 \cdots a_s^2$.
- Para $1 \leq i \leq r$ e j=0, o unificador ij é
  $\{\phi(f/\lambda w_1 \cdots w_r.w_i(h_1^k a_1^1 \cdots a_r^1) \cdots (h_t^k a_1^1 \cdots a_r^1)) \bigcup unif_m(\langle a_i^1, F^2 \rangle\ \}$
  em que t é o número de variáveis ligadas em a_i^1 e ϕ representa $unif_m(\langle a_i^1, F^2 \rangle)$

Figura 6.1: ADF - Algoritmo para obtenção dos unificadores "principais", que são o resultado de imitações e de n $(0 \leq n \leq 1)$ projeções.

Exemplo 6.3.1 Seja $\langle F^1, F^2 \rangle = \langle f\ 1\ (T\ 2)\ (y\ 2), C\ 1\ (P\ 1)\ (y\ 2) \rangle$. A aplicação do algoritmo ADF ao conjunto discórdia gera os seguintes unificadores principais:

i	j	subs. para f $\longrightarrow \phi$	Sucesso/Falha
0	0	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ 2)$	Sucesso
1	0	$f/\lambda w_1 \cdots w_3.w_1 \longrightarrow \langle 1, C\ 1\ (P\ 1)\ (y\ 2) \rangle$	Falha
1	01	$f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ (y\ 2) \longrightarrow \langle 1, 1 \rangle$	Sucesso
1	02	$f/\lambda w_1 \cdots w_3 C\ 1\ w_1\ (y\ 2) \longrightarrow \langle 1, (P\ 1) \rangle$	Falha
1	03	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_1 \longrightarrow \langle 1, (y\ 2) \rangle$	
		0 0 $y/\lambda w_1.1$	Sucesso
		1 0 $y/\lambda w_1.w_1\langle 2, 1 \rangle$	Falha
1	021	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ (y\ 2) \longrightarrow \langle 1, 1 \rangle$	Sucesso
1	031	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ w_1) \longrightarrow \langle 1, 2 \rangle$	Falha
2	0	$f/\lambda w_1 \cdots w_3.w_2 \longrightarrow \langle (T\ 2), C\ 1\ (P\ 1)\ (y\ 2) \rangle$	Falha

2 01 $f/\lambda w_1 \cdots w_3 C\ w_2\ (P\ 1)\ (y\ 2) \longrightarrow \langle (T\ 2), 1\rangle$ Falha

2 02 $f/\lambda w_1 \cdots w_3 C\ 1\ w_2\ (y\ 2) \longrightarrow \langle (T\ 2), (P\ 1)\rangle$ Falha

2 03 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_2 \longrightarrow \langle (T\ 2), (y\ 2)\rangle$

0 0 $y/\lambda w_1.1$ Sucesso

1 0 $y/\lambda w_1.w_1 \longrightarrow \langle 2, (T\ 2)\rangle$ Falha

1 11 $y/\lambda w_1.T\ w_1 \longrightarrow \langle 2, 1\rangle$ Sucesso

2 021 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_2)\ (y\ 2) \longrightarrow \langle (T\ 2), 1\rangle$ Falha

2 031 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ w_2) \longrightarrow \langle (T\ 2), 2\rangle$ Falha

3 0 $f/\lambda w_1 \cdots w_3.w_3 \longrightarrow \langle (y\ 2), C\ 1\ (P\ 1)\ (y\ 2)\rangle$ Falha

3 01 $f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ 1)\ (y\ 2) \longrightarrow \langle (y\ 2), 1\rangle$

0 0 $y/\lambda w_1.1$ Sucesso

1 0 $y/\lambda w_1.w_1$ Falha

3 02 $f/\lambda w_1 \cdots w_3 C\ 1\ w_3\ (y\ 2) \longrightarrow \langle (y\ 2), (P\ 1)\rangle$

0 0 $y/\lambda w_1.P\ 1$ Sucesso

1 0 $y/\lambda w_1.w_1 \longrightarrow \langle 2, (P\ 1)\rangle$ Falha

1 11 $y/\lambda w_1.(P\ w1) \longrightarrow \langle 2, 1\rangle$ Falha

3 03 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_3 \longrightarrow \langle (y\ 2), (y\ 2)\rangle$ Sucesso

3 021 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_3)\ (y\ 2) \longrightarrow \langle (y\ 2), 1\rangle$

0 0 $y/\lambda w_1.1$ Sucesso

1 0 $y/\lambda w_1.w_1\langle 2, 1\rangle$ Falha

3 031 $f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ w_3) \longrightarrow \langle (y\ 2), 2\rangle$

0 0 $y/\lambda w_1.2$ Sucesso

1 0 $y/\lambda w_1.w_1$ Sucesso

6.4 Combinação de Unificadores:

A idéia sobre a qual se elabora o conceito de combinação de unificadores parte da seguinte observação que já foi exposta na propriedade 2. Sejam dois unificadores

$\phi_1 = g/\lambda w_1 w_2 C\ w_1\ 2$ e

$\phi_2 = g/\lambda w_1 w_2 C\ 3\ w_2$

para um par de discórdia. Neste caso $g/\lambda w_1 w_2 C w_1 w_2$ também é um unificador para o par. Intuitivamente percebe-se que o primeiro argumento de C é 3 e o segundo é 2.

Exemplo 6.4.1 A aplicação do algoritmo ADF no exemplo 6.3.1 gerou os seguintes ramos com Sucesso, obtendo-se os seguintes unificadores "principais".

i	j	substituição para f	ϕ
01	0	0	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ 2)$,	$\phi = \{\}$
02	1	01	$f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ (y\ 2)$,	$\phi = \{\}$
03	1	03	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_1$,	$\phi = \{y/\lambda w_1.1\ \}$
04	1	021	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ (y\ 2)$,	$\phi = \{\}$
05	2	03	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_2$,	$\phi = \{y/\lambda w_1.1\ \}$
06	2	03	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_2$,	$\phi = \{y/\lambda w_1.T\ w_1\ \}$
07	3	01	$f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ 1)\ (y\ 2)$,	$\phi = \{y/\lambda w_1.1\ \}$
08	3	02	$f/\lambda w_1 \cdots w_3 C\ 1\ w_3\ (y\ 2)$,	$\phi = \{y/\lambda w_1.1\ \}$
09	3	03	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_3$,	$\phi = \{\ \}$
10	3	021	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_3)\ (y\ 2)$,	$\phi = \{y/\lambda w_1.1\ \}$
11	3	031	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ w_3)$,	$\phi = \{y/\lambda w_1.2\ \}$
12	3	031	$f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ (y\ w_3)$,	$\phi = \{y/\lambda w_1.w_1\ \}$

Em um grande número de casos o unificador é um conjunto de substituições com as variáveis livres do par de discórdia à esquerda da substituição. Dois unificadores, para se combinarem precisam ter os unificadores de suas variáveis livres combinados.

Pode-se testar se duas substituições se combinam gerando uma terceira que é obtida com a aplicação do algoritmo COMB da figura 6.2.

Nota-se que se em ξ_1 não houver $w_i[\cdots h_i^k \cdots]$ não há combinação.

Para se combinar dois unificadores considerando-os como conjuntos de substituições para as variáveis livres de F^1 e F^2 observa-se se as substituições, com a mesma variável livre à esquerda, se combinam. O algoritmo COMBT da figura 6.3 foi proposto com esta finalidade.

Nestes unificadores ainda não foi aplicado o unificador ϕ sobre a substituição para f. Com esta aplicação o terceiro argumento do 10^0 unificador torna-se 1 e o do 11^0 torna-se 2, por exemplo.

Obtidos os unificadores principais, os demais unificadores do par de discórdia são todas as combinações entre os unificadores parciais. Define-se na figura 6.4 o algoritmo UNIF($\langle F^1, F^2 \rangle$) que obtém todos os unificadores para o par

---

Algoritmo COMB : Combinação de duas substituições para uma mesma variável.

Sejam ξ_1 e ξ_2 as duas substituições par uma variável livre g.
Em ξ_1 localize as posições onde há $w_i[\cdots hi^k \cdots]$ na substituição para g (exceto no conjunto de variáveis de ligação). Use para isso a estrutura de árvore, por exemplo.

1. COMB := $\emptyset$.
2. Copie ξ_2 para ξ_3.
3. Para cada $w_i[\cdots h_i^k \cdots]$ existente no nó ijkl... da árvore da substituição para g de ξ_1.

   inicio Verifique se os nós anteriores do ramo (formado pela sequência de nós i, ij, ijk, ...($i \geq 0$) de ξ_3) não estão ocupados por algum $w_i[\cdots h_i^k \cdots]$, ($1 \leq i \leq r$).

   Se estiver, a combinação fracassa. COMB := $\emptyset$. Vá para 4.

   Caso contrário acrescente $w_i[\cdots h_i^k \cdots]$ na posição ijkl... da correspondente substituição de ξ_3. COMB := ξ_3.

   fim
4. FIM {COMB é a combinação entre ξ_1 e ξ_2.}

Figura 6.2: Algoritmo para combinação de duas substituições para a mesma variável livre gerando um unificador mais geral

Algoritmo COMBT: combina dois unificadores para um par de discórdia $\langle F^1, F^2 \rangle$

Sejam ϕ_1 e ϕ_2 estes unificadores ou conjuntos de substituições.

- COMBT := $\emptyset$.
- $\phi_3 := \emptyset$.
- Para cada substituição $\xi = y/t_{\phi_i}$, y pertencente às variáveis livres, $\xi \in \phi_i$, $(1 \leq i \leq 2)$
  1. inicio Se não existir uma substituição y/t_{ϕ_j} em $\phi_j, j \neq i$, $\phi_3 := \phi_3 \cup \{y/t_{\phi_i}\}$
  2. Se $t_{\phi_1} = t_{\phi_2}$ então $\phi_3 := \phi_3 \cup \{y/t_{\phi_1}\}$.
  3. Senão se $t_{\phi_1} \neq t_{\phi_2}$ e $COMB(\xi_{\phi_1}, \xi_{\phi_2}) = \emptyset$ então $COMBT(\phi_1, \phi_2) := \emptyset$. Va para FIM.
     Caso contrário $\phi_3 := \phi_3 \cup \{COMB(\xi_{\phi_1}, \xi_{\phi_2})\}$.
  4. fim
- FIM

Figura 6.3: Algoritmo COMBT para combinação de dois unificadores para um par de discórdia

Algoritmo UNIF($\langle F^1, F^2 \rangle$): Obtém todos os unificadores para o par .

- Seja P(ADF) o conjunto potência de ADF($\langle F^1, F^2 \rangle$). Para cada elemento de P(ADF), combine seus elementos da seguinte maneira. Seja um elemento γ de P(ADF) = $\{\theta_1 \cdots \theta_n\}$.
  1. Se n = 0 então UNIF($\langle F^1, F^2 \rangle$:= UNIF($\langle F^1, F^2 \rangle$) $\cup$ γ.
  2. Se n = 1 então UNIF($\langle F^1, F^2 \rangle$) := UNIF($\langle F^1, F^2 \rangle$) $\cup$ θ_1
  3. Se n = 2 então UNIF($\langle F^1, F^2 \rangle$) := UNIF($\langle F^1, F^2 \rangle$) $\cup$ $COMBT(\theta_1, \theta_2)$
  4. Se n$\geq$ 3 então UNIF($\langle F^1, F^2 \rangle$) := UNIF($\langle F^1, F^2 \rangle$) $\cup$ CC onde CC é obtido por
     CC := $\emptyset$
     Para i = 1 até n CC := $COMBT(CC, \theta_i)$

Figura 6.4: Algoritmo UNIF de Unificação

Exemplo 6.4.2 Unificadores para $\langle F^1, F^2 \rangle$ obtidos por combinação utilizando o algoritmo UNIF sobre os unificadores principais do exemplo 6.3.1 colocados no exemplo 6.4.1.

---

13 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_1,\ y/\lambda w_1.1\}$

14 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ (y\ 2)\}$

15 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_2,\ y/\lambda w_1.T\ 2\}$

16 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_3\}$

17 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_3)\ 1,\ y/\lambda w_1.2\}$

18 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ w_3\ (P\ 1),\ y/\lambda w_1.P\ 1\}$

19 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ w_3)\ w_3,\ y/\lambda w_1.P\ 1\}$

20 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_1,\ y/\lambda w_1.1\}$

21 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_3\}$

22 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_3)\ w_3,\ y/\lambda w_1.1\}$

23 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_1)\ w_1,\ y/\lambda w_1.1\}$

24 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_3)\ w_1,\ y/\lambda w_1.1\}$

25 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_1)\ w_3,\ y/\lambda w_1.1\}$

26 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_3)\ w_3,\ y/\lambda w_1.1\}$

27 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_3,\ y/\lambda w_1.w_1\}$

28 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ 2,\ y/\lambda w_1.2\}$

29 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_3,\ y/\lambda w_1.w_1\}$

30 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_3,\ y/\lambda w_1.2\}$

31 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_3,\ y/\lambda w_1.2\}$

32 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ 1)\ w_3,\ y/\lambda w_1.1\}$

33 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_3)\ 1),\ y/\lambda w_1.1\}$

34 $\{f/\lambda w_1 \cdots w_3 C\ 1\ w_3\ w_3,\ y/\lambda w_1.P\ 1\}$

35 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_3)\ w_3,\ y/\lambda w_1.1\}$

36 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_2,\ y/\lambda w_1.T\ w_1\}$

37 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ w_3,\ y/\lambda w_1.w_1\}$

38 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ 1)\ 2,\ y/\lambda w_1.2\}$

39 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_3,\ y/\lambda w_1.w_1\}$

40 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ 1)\ w_3,\ y/\lambda w_1.2\}$

41 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_2,\ y/\lambda w_1.T\ 2\}$
42 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_1)\ w_2,\ y/\lambda w_1.T\ w_1\}$
43 $\{f/\lambda w_1 \cdots w_3 C\ w_1\ (P\ w_3)\ w_1,\ y/\lambda w_1.1\}$
44 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_1,\ y/\lambda w_1.1\}$
45 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ 1)\ w_1,\ y/\lambda w_1.1\}$
46 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_3)\ w_1,\ y/\lambda w_1.1\}$
47 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_2,\ y/\lambda w_1.(T\ 2)\}$
48 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_2,\ y/\lambda w_1.(T\ w_1)\}$
49 $\{f/\lambda w_1 \cdots w_3 C\ w_3\ (P\ w_1)\ 1,\ y/\lambda w_1.1\}$
50 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_3\}$
51 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ w_3,\ y/\lambda w_1.w_1\}$
52 $\{f/\lambda w_1 \cdots w_3 C\ 1\ (P\ w_1)\ 2,\ y/\lambda w_1.2\}$

Deseja-se mostrar que

1. Todos os unificadores finais θ da árvore de unificação que contem apenas um argumento da forma $w_i[\cdots hi^k \cdots]$ são gerados pelo algoritmo ADF, e são os unificadores chamados de "principais".

2. Que os demais são combinações daqueles obtidos em 1, sendo a combinação definida pelo algoritmo COMBT como na discussão anterior.

Primeira parte: Observe-se que pelo lema 2 todos os argumentos de F^2 podem ser testados para uma projeção, gerando um unificador que é uma cópia de F^2 abstraida pelas variáveis $w_i \cdots w_r$ com o argumento cuja projeção resultou em Sucesso sendo substituido por $w_i[\cdots hi^k \cdots]$. A este unificador é aplicado o unificador ϕ.

Este comportamento do algoritmo MATCH ao gerar tais unificadores na árvore de unificação é simulado pelo algoritmo ADF uma vez que a árvore que representa F^2 consiste de uma estrutura que indica exatamente este argumento. Desta forma, todos os unificadores contidos na árvore, com a forma indicada em 1, são também gerados por ADF. Como todos os unificadores de ADF estão também em UNIF (Passo 2 do algoritmo), UNIF também os gera.

Encontram-se na árvore de unificação também os unificadores cuja substituição para f tem como cabeça uma das variáveis w_i. Estes estão colocados como item 2 do algoritmo ADF, e seguem também para UNIF.

Segunda parte: Os demais unificadores que são encontrados pela árvore de unificação de MATCH são aqueles compostos de mais do que um argumento $w_i[\cdots hi^k \cdots]$ na substituição para f. Considera-se aqui apenas os argumentos deste tipo para a substituição de f embora

eles possam ocorrer recursivamente e em outras substituições. Além disso o processo pode se repetir na geração de ϕ.

Quando ocorreu projeção nas posições k, l, ... m, com imitações sobre as posições restantes, o processo se deu na seguinte sequência:

- Houve uma primeira projeção que gerou um unificador
  $\{f/\lambda w_1 \cdots w_r C \cdots \phi_1(a_i^2) \cdots \phi_1(w_i[\cdots hi^k \cdots] \cdots) \cup \phi\}$
- Numa segunda projeção, antecipada ou não de imitações gerou-se um unificador da forma
  $\{f/\lambda w_1 \cdots w_r C \cdots \phi_2(\phi_1(a_i^2)) \cdots \phi_2(\phi_1(w_i[\cdots hi^k \cdots]))$
  $\cdots \phi_2(\phi_1(w_i[\cdots hi^l \cdots])) \cdots\} \cup \phi_2 \circ \phi_1\}$
  Nesta sequência, ϕ representa a composição conforme definição já estabelecida na lógica de primeira ordem.
- $\vdots$
- Na n-ésima projeção o unificador terá a seguinte forma
  $f/\lambda w_1 \cdots w_r C \cdots \phi_n(\cdots(\phi_2(\phi_1(a_i^2)))) \cdots \phi_n(\cdots(\phi_2(\phi_1(w_i[\cdots hi^k \cdots]))))$
  $\cdots \phi_n(\cdots(\phi_2(\phi_1(w_i[\cdots hi^l \cdots])))) \cdots \phi_n(\cdots(\phi_2(\phi_1(w_i[\cdots hi^m \cdots]))))$
  $\cup\ \phi_n \circ \cdots \circ \phi 2 \circ \phi_1\}$

A aplicação do unificador ϕ ao conjunto discórdia restante, após um ramo de sucesso, elimina as variáveis livres do mesmo. Portanto, não pode haver substituição y/t_1 em ϕ_1 e esta mesma variável y aparecer como substituição em qualquer $\phi_i, (i > 1)$. Assim o unificador obtido na árvore após n projeções, obedeceu à sequência de 1 a 4 acima e todos os argumentos da forma $w_i[\cdots hi^k \cdots]$ ficaram em posições diferentes, por força da técnica de construção da árvore de derivação. Pela mesma razão, não substituiu argumento de um argumento já substituido desta forma.

Nossa tese é de que este unificador é o mesmo que aquele conseguido por meio do algoritmo COMBT sobre os n unificadores principais correspondentes, que são os seguintes:

1. $f/\lambda w_1 \cdots w_r C \cdots \phi_1(a_i^2) \cdots \phi_1(w_i[\cdots hi^k \cdots] \cdots)$
2. $f/\lambda w_1 \cdots w_r C \cdots \phi_2(a_i^2) \cdots \phi_2(w_i[\cdots hi^l \cdots] \cdots)$
3. $\vdots$
4. $f/\lambda w_1 \cdots w_r C \cdots \phi_n(a_i^2) \cdots \phi_n(w_i[\cdots hi^m \cdots] \cdots)$
   em que os ϕ_i são diferentes daqueles acima, para $i > 1$.

É preciso mostrar que se $\langle a_i^1, a_k^2 \rangle$ se unificam com as substituições de ϕ_1 e se $\phi_1(\langle a_i^1, a_l^2 \rangle)$ se unificam com as substituições de ϕ_2 então a composição $\phi_2 \circ \phi_1$ é a combinação (COMBT) entre os unificadores de $\langle a_i^1, a_k^2 \rangle$ e de $\langle a_i^1, a_l^2 \rangle$.

Se a primeira condição ocorreu, isto é, houve a composição, tem-se que as substituições de ϕ_1 satisfizeram também à unificação do segundo par. e que as substituições em ϕ_2 apenas acrescentam as demais substituições necessárias. Este fato indica que as substituições de ϕ_1 também são substituições para o segundo par.

O algoritmo COMBT executa a combinação nas seguintes condições: (a) que se existir uma substituição do primeiro unificador para uma variável livre y, esta deve ser igual àquela do segundo ou não existir nele, (b) que se y não se repetir nos dois unificadores sua substituição é acrescida ao unificador e (c) se os termos da substiuição para y forem diferentes, ou há Falha ou se combinam as duas substituições.

A condição a) busca atender ao fato de que na hipótese de ter havido a composição ou a variável livre y não aparece no segundo par, ou sua substituição permite a sua unificação. A condição b) estabelece que variáveis livres diferentes tem suas substituições independentes e por último c) mostra como se pode combinar duas substituições diferentes.

Iterativamente, a análise pode ser feita até ϕ_n.

As condições de ocorrência da unificação sequencial ϕ_1 e ϕ_2 implicam no atendimento das condições de COMBT. Desta forma a existência de um unificador deste tipo na árvore de unificação implica na existência do mesmo unificador obtido por meio de COMBT.

6.5 Vantagens

Utiliza-se a seguir o exemplo 6.3.1 para estabelecer, ainda que apenas sobre alguns aspectos, um contraste entre a utilização da combinação dos algoritmos MATCH e SIMPL e o algoritmo UNIF, aqui proposto. Naquele exemplo tinha-se o seguinte par $\langle F^1, F^2 \rangle$, rígido-rigido

$$\{\langle (\text{mapfun } F_1 \text{ (cons X } L_1) \text{ (cons } (F_1 \text{ X}) \ L_2)),$$
$$(\text{mapfun } F_2 \text{ (cons 1 (cons 2 nil))}$$
$$(\text{ cons (g 1 1) (cons (g 1 2) nil }))) \rangle\}$$

que por meio da rotina SIMPL gerou o seguinte conjunto discórdia:

$$\{\langle F_1, F_2 \rangle, \langle X, 1 \rangle, \langle L_1, (cons\ 2\ nil) \rangle, \langle (F_1\ X), (g\ 1\ 1) \rangle,$$
$$\langle L_2, (cons\ (g\ 1\ 2)\ nil) \rangle \}$$

Os pares flexível-rígido do conjunto são

1. $\langle X, 1 \rangle$
2. $\langle L_1, (cons\ 2\ nil) \rangle$
3. $\langle (F_1\ X), (g\ 1\ 1) \rangle$

4. $\langle L_2, (cons\ (g\ 1\ 1)(cons\ (g\ 1\ 2)\ nil))\rangle$

Os pares 1, 2 e 4 terão ADF unitário, por isso não têm combinações e UNIF = ADF com os unificadores

$X/1$, $L_1/(cons\ 2\ nil)$ e $L_2/(cons\ (g\ 1\ 2)\ nil))$

respectivamente. Para se encontrar estes unificadores não houve qualquer movimento de termos. Apenas verificou-se a adequação de tipo. Geração de variáveis h_i não foi necessária, tampouco β-reduções.

O terceiro par foi alterado para $\langle (F_1\ 1), (g\ 1\ 1)\rangle$. Nele é gritante a vantagem de UNIF pois bastou a comparação de $\langle 1, 1\rangle$ e posteriormente $\langle 1, 1\rangle$ para se obter os três unificadores principais que são

$$F_1/\lambda w_1(g\ 1\ 1)$$
$$F_1/\lambda w_1(g\ 1\ w_1)$$
$$F_1/\lambda w_1(g\ w_1\ 1)$$

e a combinação gerou o quarto unificador que é

$$F_1/\lambda w_1(g\ w_1\ w_1)$$

A proposta alternativa do algoritmo UNIF para a unificação de um par de discórdia aponta para um ganho significativo de eficiência. A despeito de não terem sido executados testes exaustivos para comprovação de eficiência e funcionalidade, observações preliminares como os exemplos acima, estimulam este comportamento otimista.

Ressalta-se que seu desenvolvimento originou-se de uma compreensão da lógica que se escondia por trás dos mecanismos de imitação e projeção, e de uma percepção semântica do algoritmo MATCH que foi além das regras de geração de variáveis livres e de substituições parciais.

Por isso, e somente a partir deste conhecimento, foi possível perceber rapidamente quais os unificadores para um par como $\langle (F1), (g11)\rangle$ sem necessitar rastrear os algoritmos SIMPL e MATCH, com sua grande carga de detalhes.

Neste algoritmo a unificação foi feita apenas para os unificadores chamados de "principais". Os demais são conseguidos por meio de mecanismos de combinação e dispensam toda a mecânica de pesquisa incluida em MATCH. O número máximo de argumentos principais é função do número de argumentos de F^2, em qualquer profundidade, e pode ser melhor visto pela estrutura de árvore proposta na seção 6.3 para o termo rígido. Em muitos casos este número não atinge a metade do número total de unificadores.

Além da redução representada pela diminuição do número de unificadores pesquisados, o próprio tabalho de busca destes unificadores é consideravelmente reduzido no algoritmo UNIF. Este fato pode ser observado porque:

- β-reduções são quase desnecessárias.
- Geração de novas variáveis livres só ocorrem quando há abstração de algum argumento de F^1.

- Não existe repetição de segmentos como foi visto na seção anterior.
- A rotina SIMPL é simplificada

Restritas a observações empíricas preliminares, estas vantagens conferem ao algoritmo características promissoras.

Capítulo 7

Conclusões e Perspectivas

Iniciou-se este trabalho com a percepção que o λ- cálculo tipado permitia incorporar características antes não incluídas nas linguagens lógicas. Via-se mais: que este formalismo de representação não se distanciava daquele da primeira ordem a ponto comprometer as condições necessárias para a mecanização da linguagem formada, com vistas à prova de teoremas.

Percebia-se que a implementação de um interpretador para tal linguagem teria fundamental importância no amadurecimento de conceitos que poderiam esclarecer aspectos práticos envolvidos na sua implementação, enquanto permitiria analisar com mais clareza as opções de implementação e contrastar as tomadas de decisão do processo que às vezes é não determinístico.

De fato, apenas depois do protótipo ter sido executado com sucesso, sobre alguns exemplos, alguns da literatura, foi possível amadurecer sobre o sistema como um todo e perceber os amplos aspectos acima citados, merecedores de atenção e carentes de pesquisa. No entanto, esta faceta do protótipo, parece não destoar dos demais existentes, que aparentemente bem mais elaborados, sofrem reclamações da necessidade de melhorar a implementação por parte dos pesquisadores envolvidos e que vem há muito tempo trabalhando no assunto. Alegam que seus interpretadores têm falhas em termos de desempenho, estão em fase de testes, com problemas na linguagem de implementação.

O trabalho pretendeu, inicialmente, avançar em termos de eficiência do interpretador, por meio de uma implementação que melhorasse deficiências de desempenho dentre outras. A realidade da implementação do interpretador mostrou que antes de se atender a este desafio, duas facetas deveriam ser bem entendidas: a P-Derivação com seus não determinismos e o grande "mistério" que circunda a unificação de ordem superior. Enquanto estes dois mecanismos não se tornarem rotineiros, talvez seja prematuro pensar que a eficiência do interpretador como um todo seja prioritária. Usuários já acostumados com a linguagem afirmam ser inicialmente difícil escrever programas que façam uso significante do potencial do λ-PROLOG. Com o tempo, talvez se possa atingir um entendimento maior de todo o processo que justifique ênfase acentuada de todo o processo de derivação.

Assentado na abrangência do assunto em questão, o trabalho contemplou e pode ser

segmentado em três partes:

1. Formalização, exemplificação e ilustração da teoria que o fundamenta.
2. Descrição da prototipação do interpretador, modelada a partir do PROLOG padrão, com propostas de implementação em procedimentos onde tal modelo não apresentou adequação.
3. Análise detalhada do algoritmo da unificação já conhecido, com proposta alternativa para o mesmo.

No desenvolvimento do protótipo, além do fornecimento dos principais algoritmos para as diversas fases da P-derivação e de seus esquemas modulares, foram propostas algumas formas de tratamento para problemas típicos da implementação como:

1. A formulação de uma gramática para a formação da base de conhecimento que impede a inclusão na base de sentenças e consultas mal formadas.
2. Uma forma de representação dos termos que auxilia o usuário na fornecimento dos tipos não fundamentais de uma cláusula sintaticamente bem formada.
3. A pilha de controle que agrega todas as informações da pesquisa e controle contidas nos interpretadores PROLOG às informações complementares devidas à dupla potencialidade de retroencadeamento.

Destaca-se como contribuição mais significativa deste trabalho, que é resultado da análise e da implementação acima, a proposição de uma forma diferente de abordar a unificação e que é sintetizada no novo algoritmo chamado de UNIF. Tem-se a expectativa de que ele possibilite avanços expressivos em termos de desempenho de interpretadores que venham a ser desenvolvidos.

Neste algoritmo a unificação é feita apenas para os unificadores chamados de "principais". Os demais são conseguidos por meio de mecanismos de combinação e dispensam toda a mecânica de pesquisa incluída em MATCH. O número máximo de argumentos principais é função do número de argumentos da fórmula rígida do par de discórdia e, em qualquer profundidade. Em muitos casos este número não atinge a metade do número total de unificadores.

Além da redução representada pela diminuição do número de unificadores pesquisados, o trabalho de busca destes unificadores é consideravelmente reduzido, pois

- β-reduções são quase desnecessárias.
- Geração de novas variáveis livres só ocorrem quando há abstração de algum argumento da fórmula flexível.
- Não existe repetição de unificação de pares segmentos, ou de segmentos da árvore de unificação.

- A rotina SIMPL é menos usada.

Restritas a observações empíricas preliminares, estas são vantagens muito promissoras.
O presente trabalho fornece suporte para indicar algumas conjecturas e para apontar um direcionamento para futuras pesquisas como

- Experiências no uso do interpretador que ainda são poucas e é se encontrar aplicações que aproveitem este novo formalismo, mesmo que restrito, por exemplo à segunda ordem, mas que venham despertar e efetivamente contribuir para sua aplicação em situações práticas.

- Formalização teórica mais profunda que conduza a propriedades como completude do algorítmo proposto em relação ao algoritmo de unificação já existente (MATCH).

- O aprofundamento da análise deste método de unificação sinaliza que mesmo a grande preocupação com tipos existente em MATCH poderia ser significativamente reduzida. A confirmação desta conjectura pode representar um passo a mais sobre um dos parâmetros que influenciam negativamente sobre o desempenho dos interpretadores.

- Investigar o aproveitamento da análise sintática como regulador e método de conferência dos tipos. Este aspecto da implementação aparenta um melhor entendimento e maior facilidade por parte do usuário na formação de sentenças e termos para a base de conhecimento e tambem da consulta. Se esta pressuposição for verdadeira, desapareceria em grande parte a necessidade de tipificação ao considerar que a maioria das aplicações tem como seus tipos fundamentais as cadeias ou números. Nestes casos algumas variáveis e constantes poderiam ser tipificadas pelo sistema, apenas posteriormente.

Apêndice A

Exemplo de P-Derivação

Tem-se a seguir uma base de dados codificada para o λ-PROLOG. A consulta é sempre colocada como a última cláusula da base.

prefix nil Y

prefix va vy :- prefix X Y

homem jose

homem pedro

homem roque

mulher maria

mulher carlota

mulher debora

pai jose roque

pai debora artemio

pai pedro luiz

mae roque renata

mae debora marcia

mae pedro renata

filho X Y :- homem X, pai X Y

filho X Y :- homem X, mae X Y

filha X Y :- mulher X, pai X Y

filha X Y :- mulher X, mae X Y

ancestral X Y :- pai X Y

ancestral X Y :- mae X Y

ancestral X Y :- pai X Z, ancestral Z Y

ancestral X Y :- mae X Z, ancestral Z Y

mapfun F2 nil nil

mapfun F2 (con X L1) (con (F2 X) L2) :- mapfun F2 L1 L2

t :- filho Z1 Z2

t :- mae X renata, homen X, pai X Y

tt :- mapfun F1 (con 2 nil) (con (G 1 1)(con (g 1 2) nill))

O interpretador ao lê-la gera a tabela de indices abaixo, onde cada número indica a posição da cláusula dentro da base.

prefix *1*2

homen *3*4*5

mulher *6*7*8

pai *9*10*11

mae *12*13*14*15

filho *16*17

filha *18*19

ancestral *20*21*22*23

mapfun *24*25

t 26*27

tt *28

Controles para ver se ainda há cláusulas e quais estão sem ser testadas, são feitos pela manutenção de um ponteiro que se desloca sobre esta cadeia.

De posse desta base que é um programa lógico a linha 1 da pilha é preenchida com informações da consulta. Estes detalhes podem ser acompanhados na Pilha de Derivação adiante. Quando ocorre a unificação alguns de seus campos são ainda preenchidos e cria-se nova linha.

A segunda linha foi criada quando, após falhar a unificação com a cláusula 24, houve sucesso com a cláusula 25. Como houve mais de um unificador para F_1, todo o ambiente foi salvo para posterior retroencadeamento. As quatro soluções foram:

$F_1, \backslash W_1$(g 1 1)

$F_1, \backslash W_1$(g 1 W_1)

$F_1, \backslash W_1$(gW_1 W_1 1)

$F_1, \backslash W_1$(gW_1 W_1)

Dada ao usuário a oportunidade de optar por um deles foi escolhido o unificador número 1. Em parte, a segunda linha foi preenchida.

O unificador escolhido foi aplicado à mapfun F_2 L1 L2 que se torna a nova consulta mapfun $\backslash W_1$ (g 1 1)(con 2 nil)(con (g 1 2) nil) e o processo recomeça.

Observe que não há ponto de retroencadeamento, exceto para testar outros unificadores da linha 1 para a linha 2.

Novamente é testada a consulta contra as cláusulas 24 e 25 e ambas levam ao insucesso. Não há ponto de retroencadeamento em aberto, mas há unificadores não testados na passagem da linha 1 para linha 2. Restaura-se aquele ambiente e deixa-se o usuário optar por um dos três conjuntos unificadores. O escolhido foi $F_1\backslash W_1$(g W_1 1).

Esta solução é testada com a nova consulta mapfun $\backslash W_1$(g W_1 1) (con 2 nil) (con(g 1 2)nil).

Novamente a unificação fracassa para as cláusulas 24 e 25. Oferecida ao usuário uma oportunidade de escolher entre os dois unificadores restantes, foi escolhido $F_1, \backslash W_1$(g W_1 W_1). Outra vez o mecanismo conduz a um fracasso da unificação.

Na quarta opção restou o unificador $F_1\backslash W_1$(g 1 W_1). A consulta agora é feita sobre mapfun W_1 (g 1 W_1) (con 2 nil) (con (g 1 2) nil). A unificação com a cláusula 24 fracassa. No entanto há um sucesso com a clásula 25.

A linha 2 anterior é descartada e sobreposta por novas informações. A solução encontrada devolve o seguinte conjunto unificador:

$\langle(F_2/\ \backslash W_{21}$ (g 1 W_{21}),X/2,L_1/nil, L_2 /nil $\rangle)$

Apenas um conjunto unificador foi encontrado. Abre-se e atualiza-se a linha 3. O unificador é usado para instanciar a parte da direita da cláusula 25. Tem-se a nova consulta: mapfun $\backslash W_{21}$(g 1 W_{21}) nil nil). Esta unificará com a cláusula 24 e deixará um ponto de retroencadeamento que é a cláusula 25. Forma-se a linha 4. Observe que a cláusula 24 é uma asserção. Portanto chega-se a uma solução. Como há ponto de retroencadeamento é preciso testar também a última consulta com a cláusula 25. A unificação falha. Como não há mais ponto de retroencadeamento, nem unificadores sem testar, o sistema tem a prova como terminada. Antes do último retrocesso foi preciso recuperar a solução. Para isto o sistema usa o vetor TT e os índices registrados na pilha.

Pilha de Derivação							
LinhaPai	PontoDe Retorno	PróximaClau Candidata	PtoRetro Anterior	$N°$ Desta Linha	ClausulaAtual Anterior	Posição de P_j na Tabela	$N°$ de Var Livres
-	saida	-	-	1	28	-	1
1	saida	-	-	2	28	6	4
1	saida	-	-	2	28	6	4
2	saida	-	-	3	25	6	4
3	saida	25	-	4	25	6	1

Pilha de Derivação (cont.)									
$N°$ de Soluções Não Usadas	Variaveis Livres							Unificadores Disponiveis	Indice no Vetor TT
	$N°$ 1	$N°$ 2	$N°$ 3	$N°$ 4	$N°$ 5	.	.		
.	F_1	.	.	.	.	.	.	.	0
.	F_1	X	L_1	L_2	.	.	.	*2*3*4	.
.	F_1	X	L_1	L_2	.	.	.	.	3
.	F_1	X	L_1	L_2	.	.	.	.	4
.	F_1	X	L_1	L_2	.	.	.	.	4

Pilha de Derivação (cont.) Espaço para Salvar Ambiente							
lalalalal lala	lalalalal lala	lalalalal lala	lalalalal lala	lalalalal lala	lalalalal lala	lalalalal lala	lalalalal lala
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Figura A.1: Pilha de derivação para controle da P-derivação

Apêndice B

Exemplo de unificação usando o algoritmo MATCH

Na figura anexa é mostrada a árvore de derivação para um par de discórdia $\langle F^1, F^2 \rangle$ gerada pela utilização do algoritmo MATCH e que pode ser utilizada para acompanhar as demonstrações ligadas ao desenvolvimento do algoritmo UNIF, do capítulo 6.

Bibliografia

[Ait-90] Ait-Kaci, H. *The WAM: A (Real) Tutorial.* PRL Research Reports, Digital Equipment Corporation, 5, 1990.

[Andr71] Andrews, P.B. Resolution on Type Theory, *The Journal of Symbolic Logic,* Vol. 36, No. 3, Sept. 1971.

[Andr86] Andrews, P.B., *An Introduction to Mathematical logic and Type Theory: to Truth Through Proof,* Academic Press, Inc. 1986.

[Apt:82] Apt, K.R. , van Emden, M.H. Contribution to the Teory of Logical Programming, *JACM,* Vol 29, 841-862, 1982.

[Bare85] Barendregt, H.P. *The Lambda Calculus.* Elsevier Science Publishers B. V. 1985

[Bled79] Bledsoe, W. W., A Maximal Method for Set Variables in Automatic Theorem-Proving. In: *Machine Intelligence*, 9, 53-100, 1979.

[Carn58] Carnap, R. *Introduction to Symbolic Logic and its Aplication.* Dover Publications, New York, 1958.

[Chan73] Chang, L and Lee, R.C. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Chur40] Church, A. A Formulation of the Simple Theory of Types. *Journal of Simbolic Logic,* 5, 1940, pp 56-68.

[Chur36] Church, A., An Insolvable Problem of the Number Theory. *Amer. J. Math.*, 58, 345-363, 1936

[Colm73] Colmerauer, A. Hanoni, H, Roussel, P. and Pasero. *Un Systeme de Communication Homme-Machine en Français.* Groupe d'Intelligence Artificialle, Université d'Aix-Marseille, 1973

[Curr58] Curry, H. B. and Feys, R. *Combinatory Logic*, Vol 1, North-Holland, Amsterdam, 1958

[DeBr72] De Bruijn, N.G. Lambda Calculus Notation with Nameless Dummies, A Tool for Automatic Formula Manipulation. *Indag. Math.* 34, pp, 381-392.

[Fort83] Fortune, S., Leivant, D. O'Donnell M., The expressiveness os Simple and Second-order Type Structures, *JACM* 30(1), 1983.

[Free83] Frege, G. *Os Pensadores.* Editora Abril,1983.

[Gold81] Goldfab, W.D., The Undecidability of the Second-Order Unification Problem, *Theoretical Computer Science.* 13, 1981, 225-230.

[Hatc68] Hatcher, W.S., *Foundations of Mathematics.* W.B. Saunders Co, 1968.

[Hind86] Hindley, J.R., *Introduction to Combinators and Lambda-Calculus.* Cambridge University Press, 1986.

[Henk63] Henkin, L., A Proposional Types. *Fund. Mathematics,* 52, 1963, pp 223-344.

[Herb30] Recherches sur La Theorie de la Demonstration, *Travaux de la Societê des Sciences et des Lettres de Varsovie*, 33, 128, 1930.

[Hofs80] Hofstadter, D. R. *Gödel, Escher, Bach : an eternal golden braid.* NY, Vintage Books, 1980.

[Hogg84] Hogger, C.J., *Introduction to Logic Progamming.* Accademic Press, 1984.

[Huet73] Huet, G. P. , The Undecibility of Unification in Third Order Logic, *Information and Control,* 22, 1973, 257-267

[Huet73] Huet G.P. , A Mechanization of Type Theory. *Proceedings of Third International Joint Conference on Artificial Intelligence,* 1973, 139-146.

[Huet75] Huet G.P. A Unification Algorithm for Typed Lambda-Calculus. *Theoretical Computer Science,* 1, 1975, 27-57.

[Klee67] Kleene, S.C. *Mathematical Logic.* John Willey e Sons, Inc. 1967.

[Knig89] Knight, K. Unification: A Multidisciplinary Survey. *ACM Computing Surveys*, Vol 21, 1, 1989.

[Kowa79] Kowalsky, R. A. *Logic for Problem Solving.* Amsterdam, North-Holland, 1979.

[Lloy84] Lloyd, J.W. *Foundations of Logic Programing*, Springer-Verlag, 1984

[Lucc72] Lucchesi, C.L. The Indecidability of the Unification Problem for Third Order Languages. *Report C S R R 2059*, Dept. of Applied Analyses and Computer Science, University of Waterloo, 1972.

[Luda92] Ludwig A. and Amaral, W.C. A Higher Order Logic Unification Algorithm *In : Proceedings of 1992 IEEE International Conference on Systems Man and Cybernetics*, Oct. 1992, (to appear).

[Ludb92] Ludwig A. and Amaral, W.C. Um algoritmo para a Unificação na Lógica de Ordem Superior. *In: Anais da XVIII Conferência Latinoamericana de Informática*, Las-Palmas, Espanha, Agosto, 1992, (to appear).

[Mill85] Miller, D. A. and Nadathur, G. Some Uses of Higher Order Logic in Computational Linguistics. *MS-CIS-86-31*, Dept. of Comp. and Inf. Science, April, 1985

[Mill86] Miller D.A. Nadathur, G. Higher-order Logic Programing. *LNOCS* vol. 204, Springer-Verlag, 1986.

[Mill85] Miller D.A. Nadathur, G. A computacional Logic Aproach to Syntax and Semantics. *Technical Report MS-CIS-85-17,* Dep. of Comp. and Info. Science, University of Pennsylvania, 1985.

[Nada87] Nadathur, G. *A Higher Order Logic as The Basis for Logic Programing.* PhD Dissertation, University of Pennsylvania, 1987.

[Nada88] Nadathur, G., A and Miller D. An Overview of λProlog. *In: Proceedings of the 5th International Conference on Logic Programing*, MIT Press, Vol 1, Cambridge, Mass.,1988, pp. 810-827.

[Nada89] Nadathur, G. A and Jayaraman B. Towards a WAM Model for λ-Prolog. *In: Proceedings of The North American Conference on Logic Programing*, 1989, Vol2, MIT Press, Cambridge, Mass, 1989.

[Nada90] Nadathur, G. A and Miller D. Higher-Order Horn Clauses. *JACM*, 37:4, 1990, pp. 777-814.

[Nune89] Souza, J. Nunes de. *Representação do Conhecimento Usando o λ-cálculo Tipado.* Tese de doutorado, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica, 1990.

[Piet73] Pietrzykowski, T. A Complete Mechanization of Second-Order Type Theory, *JACM*, April 1973, 333-365.

[Robi65] Robinson J. A. A Machine-oriented Logic Based on the Resolution Principle. *JACM*, 12, 23-41, 1965.

[Russ03] Russel,B and Whitehead A.N. *Principia Mathematica,* Cambridge Univ Press, 1910, 1912

[Shoe67] Shoenfield, J. R. *Mathematical Logic.* Addison Wesley, 1967.

[Ster86] Sterling, L. Shapiro, E. *The Art of Prolog.* MIT PRESS, 1986.

[Turi36] Turing A.M. On Computable Numbers with an Aplication to the Entscheindungs-problem. *Proc. London Math. Soc.*, 42, 230-65.

[Warr82] Warren, D.H.D. Higher Order Extensions to Prolog: are They Needed. *In : Machine Inteligence*, vol 10. 1982, pp. 441-454 Vol 10.

[Warr83] Warren, D.H.D. An Abstract Prolog Instruction Set. Technical Note, 309, *SRI International*, Menlo Park, CA (October 1983).