UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELETRICA
DEPARTAMENTO DE ENGENHARIA DE COMPUTACAO E
AUTOMACAO INDUSTRIAL

Este exemplar corresponde à redação final da tese defendida por João Nunes de Souza e aprovada pela Comissão Julgadora em 19 de maio de 1989

W Amaral

REPRESENTACAO DE CONHECIMENTO
UTILIZANDO O
λ-CALCULO TIPADO

POR: JOAO NUNES DE SOUZA.

ORIENTADOR: PROF. DR. WAGNER CARADORI DO AMARAL.

Tese apresentada à Faculdade de Engenharia Elétrica FEE-UNICAMP como parte dos requisitos exigidos para a obtenção do título de DOUTOR EM ENGENHARIA.

CAMPINAS 1989

REPRESENTACAO DE CONHECIMENTO

UTILIZANDO O

λ-CALCULO TIPADO

JOAO NUNES DE SOUZA.

Este trabalho contou com o apoio financeiro da CAPES, e da SID informática.

Dedico este trabalho
ao meu pai.

AGRADECIMENTOS

Muitas pessoas contribuiram significativamente para o desenvolvimento deste trabalho e quero agradecer a algumas.

Quero agradecer afetuosamente a minha família, em especial ao meu pai, a minha querida irmã Júlia, ao Carlos, a Juliana, ao Luciano e a Sandra (chimbiquinha), sem os quais teria sido muito difícil a minha estada em Campinas.

Um fruto do tempo que vivi em Campinas é a sincera amizade de Rômulo e Solange Cioffi. Quero agradece-los, lembrando o apoio que sempre me deram e os bons "bate-papos" com o "Senhoooor Rooomulo!". Quero também lembrar do pessoal da "velha" república: Anunciato Storopolli, Carlão, Jaime Portugueis (da Ting), das meninas Shirley e Cecília e, em especial, de Ana Maria da Silva Noca.

Foram também importantes as contribuições que vieram de longe. Mesmo estando em Uberlândia, Nanci Costa Menezes (I L Di) e meu sobrinho Renato Túlio Nunes Côrtes apoiaram-me decisivamente.

Quero lembrar o grande estímulo e apoio que recebi dos professores do Departamento de Computação e Automação da FEC. Os principais responsáveis pela minha vontade de estudar Inteligência Artificial foram (em ordem alfabética) os doutores Fernando Gomide, Luis Gimeno, Léo Pini Magalhães Márcio Andrade Neto e Mário Gino. Agradeço ainda ao Dr. Walter Carnielli por despertar o entusiasmo pelo estudo de Lógica Matemática.

Um agradecimento especial ao Dr. Evandro Araujo pelas suas valiosas sugestões durante a elaboração do texto final deste trabalho.

Finalmente, ao Dr. Wagner Caradori do Amaral, meu orientador, um profundo reconhecimento a sua simplicidade e dedicação. Eu o sempre terei como um exemplo de orientador.

Campinas 07 de Maio de 1989.

ABSTRACT

A knowledge hierarchic representation based on the typed λ-calculus foundations is proposed in this work. The knowledge representation is based in functional programms defined as typed λ-calculus formulas set. The formulas set is associated to derivation criterias that determines the programms functions arguments and the execution function sequence.

A typed λ-calculus formulas complexity relationship based on a typed λ-calculus formulas classification is proposed. The classification is based on the formulas and type simbols sintatic structure. The type simbols orders determines typed λ-calculus formulas order relations wich define the complexity relationship. The complexity relationship is used to represent a sintatic representation of a knowledge system hierarchic relationship complexity.

A necessary condictions set to knowledge deduction from the functional programms, defined before, is demonstrated. The proofs are based on the typed λ-calculus formulas complexity relationship. Let Φ a formula and P a programm, to be possible the deduction of Φ from P, restrictions on the complexity relationship betwen Φ and the programm formulas are determined by the necessaryes condiction.

Finally, a typed λ-calculus formulas hierarchization, based on the complexity relationship defined before, is proposed. An architecture system, called EEO, to treat the knowledge hierarchic representation is defined based on the formulas hierarchization. The main objective of this architecture is to serve as a mechanism to treat knowledge hierarchization representation. The architecture is based on three elementary steps: CHOICE, EXECUTION and ORGANIZATION wich are represented by typed λ-calculus formulas.

RESUMO

Neste trabalho propõe-se um estudo da representação hierárquica de conhecimento a partir dos conceitos básicos do λ-cálculo tipado. A representação do conhecimento é feita por programas funcionais definidos como conjuntos de fórmulas do λ-cálculo tipado. Os conjuntos de fórmulas são associados a critérios de derivação que determinam os argumentos das funções dos programas, estabelecendo o sequenciamento da execução destas funções e possibilitando a derivação de novos conhecimentos.

Propõe-se um conjunto de relações de complexidade entre as fórmulas do λ-cálculo tipado, que se baseia em uma classificação destas fórmulas. A classificação se fundamenta na estrutura sintática dos símbolos para tipo associados às fórmulas. A ordem dos símbolos para tipo determina relações de ordem entre as fórmulas do λ-cálculo tipado, definindo as relações de complexidade. As relações de complexidade são utilizadas em uma representação sintática das relações hierárquicas de complexidade do conhecimento representado em um sistema.

Demonstra-se um conjunto de condições necessárias à dedução de conhecimento a partir dos programas funcionais definidos anteriormente. As demonstrações se fundamentam nas relações de complexidade entre as fórmulas do λ-cálculo tipado. Dada uma fórmula Φ e um programa P, as condições necessárias estabelecem restrições sobre as relações de complexidade entre Φ e as fórmulas de P para que seja possível a dedução de Φ a partir de P.

Finalmente, propõe-se uma hierarquização das fórmulas do λ-cálculo tipado a partir das relações de complexidade definidas anteriormente. A partir da hierarquização das fórmulas, ilustra-se a representação hierárquica de conhecimento definindo a arquitetura de um sistema. A arquitetura é denominada EEO e seu principal objetivo é representar hierarquicamente o conhecimento. Ela se fundamenta em três passos elementares que são: ESCOLHA, EXECUCAO e ORGANIZACAO, que são representados por fórmulas do λ-cálculo tipado.

INDICE

CAPITULO 1

INTRODUCAO.

CAPITULO 2

ELEMENTOS DO λ-CALCULO TIPADO E DE PROGRAMACAO LOGICA DE ORDEM SUPERIOR.

CAPITULO 3

λ-PROGRAMAS COM CRITERIO.

CAPITULO 4

COMPLEXIDADE DAS FORMULAS DO λ-CALCULO

CAPITULO 5

CONDICOES NECESSARIAS PARA DEDUCAO DE CONHECIMENTO

A PARTIR DE λ-PROGRAMAS.

CAPITULO 6

REPRESENTACAO HIERARQUICA DE CONHECIMENTO.

CAPITULO 7

CONCLUSAO DO TRABALHO.

LISTA DE SIMBOLOS

$\mathbb{N}$	Conjunto dos números naturais.
$\mathbb{F}$	Conjunto fundamental (definição 2.2.1).
$\mathbb{C}$	Conjunto construtor (definição 2.2.1).
$\mathbb{T}$	Conjunto dos símbolos para tipo (definição 2.2.1).
ordem[α]	Ordem do símbolo para tipo α (definição 2.2.2).
tipo[A_α]	Símbolo para tipo associado ao símbolo próprio A_α (definição 2.2.3).
VAR_α	Conjunto das variáveis do tipo α (definição 2.2.4).
$CONST_\alpha$	Conjunto das constantes do tipo α (definição 2.2.4).
$FORM_\alpha$	Conjunto da fórmulas do tipo α (definição 2.2.5).
V[Φ,φ]	Significado de Φ conforme φ (definição 2.3.6).
$S^X_G F$	Substituição das ocorrências livres de X em F por G (definição 2.4.1).
T	Verdade.
F	Falso.
$FORM^+$	Conjunto das fórmulas positivas (definição 2.5.4).
λ-P	λ-programa (definição 3.2.1).
θ	Substituição resposta (definição 3.2.3).
$\mathbb{D}$[Φ,λ-P]	Critério de derivação (definição 3.4.2).
$\mathfrak{L}$[A]	Conjunto das listas constituídas pelos elementos de A (seção 3.4).

$\langle \lambda\text{-}P, \mathbb{D} \rangle$	Programa com critério (definição 3.4.3).
$ARV[\alpha]$	Árvore de descrição do símbolo para tipo α (definição 4.3.1).
$dist[f,r]$	Distância do vértice f ao vértice r (definição 4.3.2).
$prof[T]$	Profundidade da árvore T (definição 4.3.2).
$ARV[\Phi_\alpha]^p$	Árvore de descrição parcial da fórmula Φ_α (definição 4.3.4).
$ARV[\Phi_\alpha]^t$	Árvore de descrição total da fórmula Φ_α (definição 4.3.5).
$ordem[\Phi_\alpha]^p$	Ordem parcial da fórmula Φ_α (definição 4.3.6).
$ordem[\Phi_\alpha]^t$	Ordem total da fórmula Φ_α (definição 4.3.6).
$=^n, >^n$	Relações de complexidade quanto ao nível (definição 4.3.7).
$=^{dp}, >^{dp}$	Relações de complexidade quanto à descrição parcial (definição 4.3.7).
$=^{dt}, >^{dt}$	Relações de complexidade quanto à descrição total (definição 4.3.7).
$grau[\Phi]$	Grau da fórmula Φ (definição 4.5.1).
$=^g, >^g$	Relações de complexidade quanto ao grau (definição 4.5.2).
$comp[\Phi]$	Comprimento da fórmula Φ (definição 5.2.1).
k-E	k-meta-escolha (definição 6.4.2).
k-A	k-meta-execução (definição 6.4.3).
k-ORG	k-meta-organização (definição 6.4.4).
$\mathcal{W}_{pc\text{-}\Psi}$	Palavra de controle associada à função Ψ (definição 6.5.1).
$\mathcal{X}_{grau}$	Variável grau ativo do sistema (definição 6.5.2).
$\mathcal{M}_{modo}$	Variável que representa o modo fundamental (seção 6.6.2.3).
$\mathcal{MP}_{modo}$	Variável que representa o modo principal (seção 6.6.2.3).

MODO	Função "modo" (definição 6.8.1).
SOB(T)	Situação objetivo no ciclo t (seção 6.8.2.1).
SAT(T)	Situação atual no ciclo t (seção 6.8.2.1).
TR^j	Transição j (seção A.4).
base[Φ]	Número base da fórmula Φ (definição C.2.5).

CAPITULO I

INTRODUCAO

Neste capitulo e apresentada uma introducao aos principais conceitos propostos neste trabalho. Justifica-se a utilizacao de um sistema logico de ordem superior na representacao de conhecimento.

1.1. INTRODUCAO.

Este trabalho trata do estudo da representação de conhecimento em um dado sistema. O estudo da representação de conhecimento em sistemas, como área de pesquisa em inteligência artificial, iniciou-se nos anos 60 e atualmente divide-se principalmente em quatro subáreas [McCalla, 1983].

(i) Representação em redes semânticas, cujos fundamentos foram inicialmente propostos por M. Quillian e S. Shapiro [Quillian, 1968] [Shapiro, 1971]. Uma rêde semântica é constituída por um conjunto de nós interligados por arcos, que relacionam os objetos representados nos nós. Esta estrutura possibilita a representação de símbolos e apontadores em computação simbólica [Barr, 1981]. Entretanto, ela apresenta várias dificuldades: (a) de implementação computacional e de representação quando a base de dados do sistema é extensa, (b) a identificação do conhecimento que está realmente representado em um nó, (c) a unicidade da representação de um fato, (d) a representação da variável tempo, etc [Barr, 1981], [Simmons, 1973], [Anderson, 1973], [Findler, 1979].

(ii) Representação em Frames, conforme a proposta de M. Minsky [Minsky, 1975]. Esta estrutura permite a representação de uma quantidade extensa de conhecimento necessária no tratamento de tarefas cognitivas. As pesquisas sobre Frames são recentes e numerosos problemas, tal como a heurística da utilização do conhecimento representado, ainda devem ser resolvidos para se ter a sua utilização [Barr, 1981].

(iii) Representação em sistemas de produção, cujos fundamentos foram inicialmente propostos por A. Newell [Newell, 1973]. Este tipo de representação é um esquema procedimental. Apresenta as vantagens dos esquemas de representação declarativa tal como a modularidade. Uma outra vantagem é que as regras de produção são estruturadas como as pessoas falam

sobre a resolução de seus problemas Algumas das desvantagens inerentes a este sistema são: (a) a ineficiência da execução do programa, (b) a dificuldade em acompanhar o fluxo de controle na resolução de um problema. Este fato decorre do isolamento e da uniformidade das regras de produção [Barr, 1981].

(iv) Representação em lógica de primeira ordem, conforme é analisado em [Nilsson, 1980] e em [Robinson, 1965]. A maior qualidade deste formalismo é que ficam explícitos o conhecimento representado no sistema e as suas consequências ou possíveis inferências. A representação em lógica é também precisa, flexível e modular. Como desvantagem, não se tem a distinção entre a representação propriamente dita e o seu processamento [Barr, 1981].

Atualmente o estudo da representação de conhecimento extende-se com a utilização de outras teorias, como Lógica Monotônica [Bobrow, 1980], Lógica Nebulosa [Zadeh, 1975], Lógica Temporal, Lógica de Ordem Superior [Nadathur, 1987], etc. A maior dificuldade, na maioria destes esquemas, é a representação da heurística de controle do sistema. Isto é, determinar como os fatos representados no sistema devem ser utilizados [Barr, 1981].

O principal objetivo deste trabalho é a proposta de um esquema de representação de conhecimento onde a heurística de controle seja facilmente representada. Nesta representação, denominada representação hierárquica de conhecimento, utiliza-se um subconjunto da lógica matemática, denominado λ-cálculo. Existem dois tipos de λ-cálculo: o λ-cálculo sem tipos [Martins, 1988], [Hidley, 1986], [Dickmann, 1988] , e o λ-cálculo tipado [Girard, 1987]. No primeiro não se considera a associação de símbolos para tipo aos elementos da linguagem, enquanto que no λ-cálculo tipado, os símbolos para tipos estão presentes, permitindo uma classificação das fórmulas e consequentemente do conhecimento por elas representado. Assim, pode-se fazer um estudo da representação hierárquica de conhecimento, a partir dos fundamentos do λ-cálculo tipado, enfocando uma heurística de controle que considera a hierárquia do conhecimento represenado.

Finalmente, convém lembrar que a representação de conhecimento é um problema fundamental em todas as ciências [Hessen, 1987], [Kant, 1983] e não é objetivo deste trabalho a comparação entre as diferentes representações possíveis [Woods, 1983], bem como o seu estudo filosófico [Hessen, 1987].

O sistema lógico considerado neste trabalho define um λ-cálculo tipado que se fundamenta no sistema lógico inicialmente definido por Church [Church, 1940] revisto nas versões apresentadas em [Hatcher, 1968], [Gallin, 1975], [Henkin, 1963], e [Andrews, 1971]. A escolha do λ-cálculo tipado proposto por Church [Church, 1940] como o fundamento da representação hierárquica de conhecimento, deve-se às seguintes características:

(i) é um sistema lógico de ordem superior.

(ii) A linguagem deste sistema lógico admite símbolos funcionais como variáveis.

(iii) O domínio das funções é constituído por conjuntos de outras funções.

(iv) As fórmulas do λ-cálculo tipado são associadas a símbolos para tipo. Além disso, as fórmulas utilizadas na representação do conhecimento podem ser consideradas como funções ou argumentos de funções.

(v) Há uma relação de ordem entre estes símbolos, que possibilita estabelecer várias relações de ordem entre as fórmulas do λ-cálculo tipado.

(vi) No λ-cálculo tipado as representações de elementos funcionais e de dados são idênticas. Isto é, há uma uniformidade na representação do conhecimento no sistema.

(vii) O problema da determinação da forma normal de uma fórmula do λ-cálculo tipado é decidível. Assim, os programas considerados são constituídos apenas por fórmulas normais não equivalentes, eliminando redundâncias na representação do conhecimento. Deve-se observar que no λ-cálculo sem tipos, a determinação da forma normal de uma fórmula não é decidível [Girard, 1987], [Hidley, 1986].

(viii) Este sistema lógico é utilizado na definição formal de programação lógica de ordem superior.

Uma extensão do sistema lógico apresentado neste trabalho é o sistema F de Girard [Girard, 1987], [Fortune, 1983]. Neste sistema os símbolos para tipo ocorrem também na forma de variáveis. O estudo destas fórmulas, chamado de λ-cálculo de segunda ordem é utilizado no estudo da correção automática de programas.

Propõe-se uma representação hierárquica de conhecimento a partir da definição de programas funcionais, que são formados por conjuntos de funções do λ-cálculo tipado.Esta representação é obtida a partir dos conceitos fundamentais de lógica de ordem superior. Em linguagens de programação lógica como PROLOG, as estruturas de dados e os mecanismos utilizados na representação e tratamento do conhecimento são, fundamentalmente, termos de lógica de primeira ordem [Andrews, 1986], [Sterling, 1986], [Turbo Prolog, 1986]. A extensão destas estruturas e mecanismos, com a utilização de elementos da lógica de ordem superior, mais especificamente do λ-cálculo tipado, estabelecem novos métodos para a representação de conhecimento. Exemplos desta extensão são:

(i) estudo de linguagem natural utilizando predicados de ordem superior [Montague, 1974], [Walker, A., 1987], [Miller, 1985(a)], [Miller, 1985(b)],

(ii) adição de predicados de ordem superior em versões de programação lógica de primeira ordem [Sterling, 1986], [Robinson, 1988],

(iii) estudo de programação lógica de ordem superior [Miller, 1986(a)], [Nadathur, 1987],

(iv) especificação de provadores de teoremas em lógica de ordem superior [Felty, 1988],

(v) teoria dos módulos para programação lógica [Miller, 1986(b)],

(vi) manipulação de fórmulas e programas em programação lógica [Miller, 1987] e

A programação lógica de ordem superior possui resultados semelhantes àqueles encontrados em programação lógica de primeira ordem, [Casanova, M.A., 1986], [Lloyd, J.W., 1984], [Hogger, C.J., 1984], [Apt, 1982]. Esta teoria foi iniciada em 1971 quando Andrews apresentou a resolução para sistemas lógicos de ordem superior [Andrews, 1971]. Entretanto, na análise de Andrews ficou em aberto o problema da unificação, que é fundamental para programação lógica e que possibilita a mecanização da lógica [Pietrzykowski, 1973], [Huet, 1973(b)], A indecidibilidade da unificação foi estudada por Lucchesi, Huet e Goldfab e um algoritmo para a unificação de termos de ordem superior foi apresentado por Huet em 1975 [Luchesi, 1972], [Huet, 1973(a)], [Goldfab, 1981] [Huet, 1975]. Em 1983 Miller estendeu alguns resultados de programação lógica de primeira ordem, como o teorema de Herbrand, para programação lógica de ordem superior [Miller, 1983]. Interpretadores e linguagens lógicas de ordem superior foram propostos por Huet [Huet, 1985]e por Nadathur [Nadathur, 1987]

A escolha da lógica de ordem superior como mecanismo para a representação e tratamento de conhecimento é um tema controvertido. Os principais argumentos contrários à sua utilização encontrados na literatura [Miller, 1985(b)] são os seguintes:

(i) Foi demonstrado por Godel que lógica de ordem superior é essencialmente incompleta [Shoenfield, 1967]. Em outras palavras, as sentenχas verdadeiras de um sistema lógico de ordem superior não são recursivamente enumeráveis. Neste sentido, um provador de teorema, que utiliza lógica de ordem superior, não é completo, nem mesmo teoricamente.

(ii) Linguagens de programação lógica de primeira ordem, como Prolog, são equivalentes à máquina de Turing. Assim, tudo que é codificado em uma linguagem fundamentada em um sistema lógico de ordem superior e que é computável, também pode ser codificado em linguagem lógica de primeira ordem.

(iii) Há poucas pesquisas sobre provadores de teoremas que utilizam lógica de ordem superior. "Acredita-se" que a prova de teoremas em sistemas lógicos de ordem superior seja difícil.

Os fatos mencionados são utilizados, frequentemente, contra a utilização de lógica de ordem superior na construção de sistemas computacionalmente viáveis. Entretanto, apesar da veracidade das afirmações, nada se conclui em definitivo sobre a inviabilidade da utilização de lógica de ordem superior. A seguir são apresentados alguns argumentos a este respeito.

A indecidibilidade a respeito das sentenças verdadeiras dos sistemas lógicos de ordem superior não possui importância do ponto de vista computacional. Pois, mesmo em programação lógica de primeira ordem, a resolução é indecidível [Casanova, 1986]. Além disso, os sistemas lógicos de ordem superior considerados são sistemas simplificados, nos quais as sentenças verdadeiras são recursivamente enumeráveis. Os sistemas utilizados são versões da teoria dos tipos simplificada de Church, desenvolvida em 1940, [Church, 1940], [Nadathur, 1987].

A segunda observação não é uma argumentação contrária à utilização de lógica de ordem superior. Se o fosse, linguagens como Fortran, Pascal e inúmeras outras, todas equivalentes à máquina de Turing, não seriam utilizadas. Ao se escolher uma linguagem de programação, entre várias, todas equivalentes à máquina de Turing, o que se considera é o poder de representação, a eficiência, a disponibilidade de cada linguagem, etc.

Embora haja um número menor de estudos sobre a representação de conhecimento em lógica de ordem superior, isto também não justifica a sua não utilização. Já se tem vários resultados, tais como a resolução em ordem superior [Andrews, 1971], a unificação de fórmulas do λ-cálculo tipado [Huet, 1975], a construção de interpretadores para programação lógica de ordem superior [Nadathur, 1987], [Miller, 1986(c)], [Huet, 1986], [Coquand, 1975], etc. Há também casos, como por exemplo a demonstração do teorema de Cantor, que é mais simples em lógica de ordem superior [Huet, 1984].

Finalmente, as características da representação hierárquica de conhecimento, indicadas a seguir, justificaram a escolha de lógica de ordem superior.

(i) A presença de conhecimento sobre conhecimento, ou meta conhecimento, é comum em várias áreas, como por exemplo em controle hierárquico, onde se tem diferentes níveis de controle [Findeisen, 1980] e em sistemas especialistas [Buchanan, 1984].

(ii) A representação de meta-conhecimento em lógica de ordem superior é "natural" [Miller, 1985(b)] pois os símbolos funcionais também podem ser variáveis [Nadathur, 1987].

(iii) A utilização da teoria de tipos, comum em lógica de ordem superior, elimina a possibilidade de sistemas paradoxais. Isto é, a utilização de símbolos para tipo possibilita uma representação hierárquica de conhecimento na qual não aparecem paradoxos, como o paradoxo do barbeiro de Bertrand Russel [Hacther, 1968].

Para a classificação do conhecimento são definidas relações de complexidade que permitem ordenar as fórmulas do λ-cálculo tipado. Como linguagem para a representação hierárquica de conhecimento, propõe-se um novo conceito de programa funcional, denominado programa com critério. Esta programação permite explicitar a hierarquia da representação do conhecimento.

Neste trabalho, não é propósito analisar o mecanismo de dedução utilizado em lógica de ordem superior. Pode se dizer que este mecanismo é similar ao da lógica de primeira ordem, na dedução a partir de cláusulas de Horn [Lloyd, 1984]. Substituição, unificação e backtracking são combinados para se construir provadores de teoremas em primeira ordem e em ordem superior. Deve-se também observar que a unificação em ordem superior não é uma simples extensão da unificação que ocorre em lógica de primeira ordem. Dadas duas fórmulas Φ e Ψ, pertencentes a um sistema lógico de ordem superior, nem sempre existe o unificador mais geral de Φ e Ψ. Além disso, a questão sobre a existência do unificador não é decidível [Huet, 1973(a)], [Goldfarb, 1981]. O estudo de subcasos onde a unificação de ordem superior é decidível é discutido em [Huet, 1978] e um algoritmo linear que decide sobre a existência de um unificador para fórmulas de segunda ordem é encontrado em Simon[Simom, 1984].

1.2 DESCRICAO DO TRABALHO.

Este trabalho é composto por 7 capítulos e três apêndices. O capítulos são descritos a seguir.

1.2.1 CAPITULO 2.

Neste capítulo são apresentadas as noções fundamentais do λ-cálculo tipado e de programação lógica de ordem superior. O λ-cálculo tipado é definido a partir de um sistema lógico de ordem superior [Church, 1940], [Henkin, 1963]. Na construção deste sistema são consideradas a definição de símbolos para tipo e a ordem destes símbolos. Também é analisada a caracterização hierárquica dos objetos formados a partir dos elementos de um conjunto. São definidos o alfabeto, as fórmulas bem formadas e a interpretação do sistema lógico. A partir da definição do sistema lógico, são analisadas as operações de λ-conversão e a equivalência de fórmulas do λ-cálculo tipado. São apresentados os teoremas da forma

normal, da unicidade da forma normal e de Church-Rosser, que garantem a existência e unicidade da forma normal de uma fórmula do λ-cálculo tipado.

Finalmente, são introduzidas as definições básicas de programação lógica de ordem superior. Estas definições baseiam-se no sistema lógico apresentado, sendo portanto uma aplicação do λ-cálculo tipado. São apresentados exemplos que ilustram as diferenças entre programação lógica de ordem superior e programação lógica de primeira ordem.

1.2.2 CAPITULO 3.

Os conceitos propostos neste capítulo iniciam-se pela definição de λ-programa e ordem de um λ-programa. São apresentados os conceitos de lista compatível, substituição resposta, derivação em uma fórmula e sua representação gráfica. A partir destas definições é analisado o conceito de programa com critério. Também é analisada a derivação de conhecimento a partir de um programa com critério. São consideradas duas formas de derivação de conhecimento a partir de um λ-programa. Uma derivação "forward" e uma derivação "backward" ou por retrocesso. A primeira é obtida a partir das definições de sequências de derivação com memória e sem memória. Já a segunda, é analisada a partir da definição de sequência de derivação por retrocesso. Finalmente, os conceitos apresentados são ilustrados pela apresentação de um λ-programa com critério, que representa o modelo de uma rede neural. Apresenta-se também um λ-programa, que é a tradução de um programa lógico de ordem superior, apresentado no capítulo 2, para a linguagem dos λ-programas.

1.2.3 CAPITULO 4.

Neste capítulo são definidas relações de complexidade entre as fórmulas do λ-cálculo tipado, [Souza, 1989]. São introduzidas diversas relações de complexidade que traduzem sintaticamente as "complexidades" do conhecimento. Para definir formalmente as relações de complexidade é apresentada a árvore de descrição de um símbolo para tipo. Em seguida são definidas as árvores de descrição total e parcial de uma fórmula. Tais definições, juntamente com a definição de ordem dos símbolos para tipo associados às fórmulas, determinam relações de complexidade entre as fórmulas do λ-cálculo tipado. Definem-se as relações de complexidade quanto à descrição total, quanto à descrição parcial, quanto ao nível e quanto ao grau.

1.2.4 CAPITULO 5.

Analisam-se as condições necessárias para que uma fórmula Φ seja uma dedução por derivação a partir de um programa com critério P. Estas condições determinam restrições sobre a fórmula Φ para que seja possível a construção de uma sequência de derivação neutra e restrita a partir de P. As restrições, que definem as condições necessárias, utilizam as relações de complexidade entre Φ e as fórmulas do programa P.

1.2.5 CAPITULO 6.

Ilustra-se a representação hierárquica de conhecimento a partir de programas com critério P. Considera-se uma hierarquização das fórmulas do λ-cálculo tipado, o que determina uma ordenação das fórmulas do programa P. Inicialmente, propõe-se um conjunto de hierarquias em uma lista de fórmulas. São definidas hierarquias quanto ao grau, quanto ao nível e quanto à descrição total. Em seguida é definida uma hierarquia completa a partir da composição de hierarquias quanto ao grau, quanto ao nível e quanto à descrição total. A partir das definições de hierarquias, é proposto o algoritmo da hierarquia completa. O resultado da aplicação deste algoritmo sobre um conjunto de fómulas determina uma lista de fórmulas ordenada segundo uma hierarquia completa.

Em sequência apresenta-se uma proposta de uma arquitetura de um sistema para a representação hierárquica de conhecimento. Esta arquitetura, denominada EEO, baseia-se nos passos elementares ESCOLHA, EXECUÇÃO e ORGANIZAÇÃO [Souza, 1988]. Os passos elementares são definidos utilizando-se os elementos do λ-cálculo tipado. São definidas as funções do tipo escolha, do tipo execução e do tipo organização. O resultado da aplicação do algoritmo da hierarquia completa sobre o conjunto das fórmulas que representam o conhecimento do sistema determina uma ordenação em P. Em seguida é definido um critério de derivação, que se fundamenta nos símbolos para tipo associados às fórmulas do λ-cálculo tipado. Após a definição do critério de derivação é analisado o sequenciamento das execuções das funções que representam o conhecimento de um sistema com arquitetura EEO. É proposto um sequenciamento para os passos elementares de um mesmo grau de complexidade e para as funções com graus de complexidade diferentes. Finalmente, é considerado o controle do sequenciamento dos passos elementares e é proposto um algoritmo para o controle dos modos de operação do sistema.

1.2.6 CAPITULO 7.

Apresenta-se a conclusão do trabalho.

CAPITULO 2

ELEMENTOS DO λ-CALCULO TIPADO E DE PROGRAMACAO LOGICA DE ORDEM SUPERIOR

Neste capitulo sao apresentadas as nocoes fundamentais do λ-calculo tipado e de programacao logica de ordem superior. Inicialmente e introduzido um sistema logico de ordem superior, que define a sintaxe e a semantica do λ-calculo tipado. Sao analisadas as operacoes de λ-conversao e a equivalencia entre as formulas deste sistema. Finalmente, a partir do sistema logico apresentado, sao apresentadas as definicoes basicas de programacao logica de ordem superior.

2.1 INTRODUCAO.

O λ-cálculo tipado apresentado neste trabalho, é o resultado do desenvolvimento da teoria dos tipos [Whitehead, 1927], iniciado no final do século passado. O primeiro passo foi a tentativa de Frege em estabelecer os fundamentos da matemática, baseada em uma axiomatização da teoria dos conjuntos [Frege, 1983]. Entretanto, esta primeira axiomatização, apresentada por Frege, continha falhas, como foi demonstrado por Bertrand Russell em [Whitehead, 1927]. A idéia de Russel foi a introdução de símbolos para tipo, com a finalidade de evitar o comportamento paradoxal de conjuntos tais como:

$$A = \{x;\ x \notin x\},$$

onde se tem $x \notin x$ [Hatcher, 1968]. Neste sentido, Russell é um dos precursores da análise da representação hierárquica de conhecimento. A teoria dos tipos do "Principia Matemática" de Russell foi utilizada mais tarde, em 1928, por Hilbert e Ackermann para definir o cálculo de segunda ordem e de ordem superior, onde os símbolos funcionais podem ser constantes ou variáveis [Hilbert, 1928]. Em 1940 Church apresentou uma teoria dos tipos, denominada "teoria dos tipos simplificada" ou "λ-cálculo tipado" e que é um cálculo de ordem superior [Church, 1940]. O λ-cálculo tipado introduzido neste capítulo baseia-se na teoria apresentada por Church [Church, 1940].

Na seção 2.2 é definido um sistema lógico de ordem superior baseado nas versões apresentadas em [Hatcher, 1968], [Gallin, 1975], [Henkin, 1963], [Church, 1940] e [Andrews, 1971]. Após as definições dos elementos fundamentais do sistema lógico, são apresentados comentários informais que relacionam as definições propriamente ditas e a representação hierárquica de conhecimento. Inicialmente, nas seções 2.2.1 e 2.2.2, são definidos, respectivamente, os símbolos para tipo e a ordem de um símbolo para tipo. Na seção 2.2.3 é definida a relação tipada baseada em um conjunto D, que determina uma hierarquização dos elementos do conjunto D. O alfabeto do sistema lógico é definido na seção 2.2.4 e as fórmulas na seção 2.2.5.

Na seção 2.3 considera-se a interpretação do sistema lógico. Esta interpretação estabelece a semântica do sistema lógico. Inicia-se a análise apresentando os conceitos de subfórmula, abstração e ligação de variáveis e ocorrência livre de uma variável indicados, respectivamente, nas seções 2.3.1, 2.3.2 e 2.3.3. Em seguida, considera-se a interpretação propriamente dita. Na seção 2.3.4 é definido o conceito de designação, na seção 2.3.5 é definida uma estrutura padrão baseada em D_i $1 \leq i \leq n$ e finalmente, na seção 2.3.6 é definida a interpretação padrão baseada em D_i $1 \leq i \leq n$.

Na seção 2.4 considera-se a análise dos conceitos de λ-conversão entre fórmulas do λ-cálculo tipado. Inicialmente, nas seções 2.4.1 e 2.4.2, são apresentadas as definições básicas de substituição e "G_α é livre para X_α em F". Em seguida, na seção 2.4.3 são definidas as operações de λ-conversão e na seção 2.4.4 são definidos os conceitos de β-redex e η-redex. Em seguida, são definidos, respectivamente, nas seções 2.4.5 e 2.4.6 as λ-conversoes de fórmulas do λ-cálculo tipado e o conceito de fórmula normal. Finalmente, são indicados, respectivamente, nas seções 2.4.8, 2.4.9 e 2.4.10 os teoremas da forma normal, da unicidade da forma normal e de Church-Rosser.

A partir do sistema lógico, são introduzidos, na seção 2.5, os fundamentos de programação lógica de ordem superior [Nadathur, 1987]. Programação lógica de ordem superior é uma aplicação da teoria do λ-cálculo tipado apresentada nas seções 2.2, 2.3 e 2.4, o mesmo ocorrendo com a representação hierárquica de conhecimento proposta neste trabalho. São apresentados as definições básicas dos elementos que constituem esta programação. Da seção 2.5.2 até a seção 2.5.10 são apresentadas, respectivamente, as definições de predicado, proposição, fórmulas positivas, fórmulas objetivo, átomo, átomo rigido, fecho universal, cláusulas definidas de ordem superior e programa lógico de ordem superior.

2.2 O SISTEMA LOGICO.

O sistema lógico considerado baseia-se no sistema lógico inicialmente definido por Church [Church, 1940] e sua escolha deve-se aos seguintes fatos:

(i) Este sistema lógico é um sistema lógico de ordem superior.

(ii) Lógica de ordem superior possui uma linguagem que admite predicados e símbolos funcionais como variáveis.

(iii) O domínio das funções e predicados é constituido por um conjunto de relações arbitrárias.

(iv) Este sistema lógico é utilizado na definição formal de programação lógica de ordem superior.

(v) Neste sistema todos os elementos do alfabeto estão associados a símbolos para tipo.

O sistema lógico a ser utilizado neste trabalho, no estudo da representação hierárquica de conhecimento, é o mesmo que fundamenta a programação lógica de ordem superior [Nadathur, 1987]. Além disso, há uma hierarquia entre os símbolos para tipo associados aos elementos do alfabeto deste sistema, o que determina os fundamentos da representação hierárquica de conhecimento a ser considerada neste trabalho.

2.2.1 DEFINICAO. Simbolos para tipos.

Seja $\mathbb{F}$ um conjunto com pelo menos dois elementos "$\bullet$" e "ι", que não são n-uplas ordenadas onde $n \geq 2$. Seja também um conjunto $\mathbb{C}$ de pares ordenados $\langle c,n \rangle$, onde "c" é um símbolo qualquer, $n \in \mathbb{N}$. Para cada "c" existe no máximo um par $\langle c,n \rangle$ tal que $\langle c,n \rangle \in \mathbb{C}$. Os conjuntos $\mathbb{F}$ e $\mathbb{C}$ são denominados conjunto fundamental e conjunto construtor, respectivamente.

O conjunto dos símbolos para tipos é o menor conjunto $\mathbb{T}$ tal que:

(i) $\mathbb{F} \subset \mathbb{T}$

(ii) Se $\langle c,n \rangle \in \mathbb{C}$ e $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{T}$, então
$(c\ \alpha_1 \alpha_2 \ldots \alpha_n) \in \mathbb{T}$

(iii) Se $\alpha \in \mathbb{T}$ então $(\alpha) \in \mathbb{T}$

(iv) Se $\alpha, \beta \in \mathbb{T}$ então $(\alpha \succ \beta) \in \mathbb{T}$

***

EXEMPLO 2.1

Considere

$$\mathbb{F} = (\bullet, t, int),$$

$$\mathbb{C} = (\langle list, 1 \rangle),$$

então tem-se, por exemplo, os seguintes símbolos para tipo.

$$(\bullet \succ t),\ (\bullet),\ ((\bullet \succ t) \succ (\bullet)),\ (t \succ \bullet),$$

$$(((\bullet \succ t) \succ (\bullet)) \succ (t \succ \bullet))$$

Observe que o símbolo $((\bullet \succ t) \succ (\bullet))$ é formado a partir dos símbolos $(\bullet \succ t)$ e $(\bullet)$ utilizando a regra (iii) da definição anterior. Tem-se também os símbolos para tipo, formados utilizando o conjunto construtor.

$$(list\ \bullet),\ (list\ t)\ (list\ int)$$
$$((list\ int) \succ (int \succ int))$$

***

NOTACAO: Quando não houver possibilidade de interpretação errônea, será utilizada a seguinte notação:

$\alpha\beta$ no lugar de $(\alpha \succ \beta)$

Além disso, considera-se a associatividade à direita. Tem-se, por exemplo:

$(\alpha \succ \beta \succ \gamma)$ no lugar de $(\alpha \succ (\beta \succ \gamma))$

***

A seguir são apresentados exemplos que utilizam, informalmente, a relação entre a definição de símbolos para tipo e a representação de conhecimento. Nestes exemplos são utilizados objetos ainda não definidos no texto, tais como as noções de variáveis, constantes, relações e fórmulas. Estas noções são aquelas usualmente utilizadas em lógica matemática.

EXEMPLO 2.2

Seja um sistema que considera as seguintes medidas de um processo,

(temperatura, pressao, volume, densidade)

associadas a um elemento gasoso. Naturalmente, há várias formas de se representar o conhecimento envolvido neste processo. Pode-se utilizar lógica de ordem superior, ou mesmo de primeira ordem. No caso em que se considera lógica de ordem superior, o conhecimento do sistema é classificado quanto a sua "complexidade". Pode-se considerar, por exemplo, que as medidas são os objetos mais "elementares" sobre os quais o sistema determina resultados. Neste caso, considera-se o

símbolo para tipo associado às medidas como um símbolo pertencente ao conjunto fundamental $\mathbb{F}$. Considere o símbolo "med". Assim, o símbolo para tipo associado a um objeto como **temperatura** é o símbolo "med". Observe que tais escolhas são inteiramente arbitrárias e dependem de como o conhecimento está sendo representado no sistema. Símbolos pertencentes ao conjunto fundamental são associados aos objetos, mais "elementares", sobre os quais o sistema está atuando.

Na representação do conhecimento do sistema, há os conjuntos de medidas, ou conjunto de objetos "elementares". Se os elementos de um conjunto estão associados ao símbolo para tipo "α", então o símbolo para tipo associado a este conjunto é

$$(\alpha).$$

Logo, se

A = (pressao, densidade)

então o símbolo para tipo associado a **A** é **(med)**.

Além dos conjuntos de objetos, há também as relações entre os objetos do sistema. Há, por exemplo, relações entre as medidas do processo. Como se sabe, pela lei de Boyle, dado um elemento gasoso sob condições isovolumétricas, então

$$P = K\ T$$

onde P é a pressão, T a temperatura e K uma constante. Considere então a relação **(boyle X Y)**, que é assumida como verdadeira quando as variáveis X e Y são substituidas por medidas relacionados pela lei de Boyle. Assim, pode-se perguntar ao sistema sobre a veracidade da fórmula,

(boyle temperatura, pressao)

As fórmulas que possuem valor de verdade são associadas a um símbolo para tipo pertencente ao conjunto fundamental $\mathbb{F}$, diferente do símbolo associado às medidas. Neste trabalho utiliza-se o símbolo

"t"

Assim, **(boyle temperatura, pressao)** é um objeto do tipo "t", que pode ser verdadeiro ou falso.

***

EXEMPLO 2.3

Na representação do conhecimento em um sistema, além da representação dos objetos "elementares", dos conjuntos de objetos, há também funções que relacionam objetos e conjuntos. Associa-se aos símbolos funcionais os símbolos para tipo $(\alpha \succ \beta)$. Neste caso, se um símbolo funcional f está associado ao símbolo para tipo $(\alpha \succ \beta)$, então os elementos do domínio de f são objetos associados ao símbolo para tipo α e os elementos do contradomínio são objetos associados ao símbolo para tipo β.

O predicado **boyle**, indicado acima, é um predicado associado ao símbolo para tipo

$$(\textbf{med} \succ (\textbf{med} \succ t)).$$

e a aplicação deste predicado às constantes **temperatura** e **pressão** é indicada como se segue: inicialmente tem-se a aplicação de (**boyle** X Y) à constante **temperatura**, isto é,

$$(\textbf{boyle } X\ Y)\ \textbf{temperatura} = (\textbf{boyle temperatura } Y)$$

onde

$$(\textbf{boyle temperatura } Y)$$

é uma fórmula do tipo

$$(\textbf{med} \succ t)$$

Em seguida, aplicando (**boyle temperatura** Y) à constante **pressao**, tem-se o resultado

$$(\textbf{boyle temperatura } Y)\ \textbf{pressao} = (\textbf{boyle temperatura pressao})$$

sendo

$$(\textbf{boyle temperatura pressao})$$

uma fórmula do tipo "t". Neste exemplo, é introduzida, de uma maneira informal, a notação utilizada pelo λ-cálculo. Isto é, dada uma função ou um predicado com mais de um argumento, como, por exemplo, f(X,Y), então a aplicação de f às constantes A e B é feita por etapas. Consideradando-se a notação a seguir,

$$f(X,Y) \equiv (f\ X\ Y)$$

a aplicação de f à constante A é dada por:

$$(f\ X\ Y)\ A = (f\ A\ Y)$$

e em seguida, a aplicação do resultado anterior à constante B é,

$$(f\ A\ Y)\ B = (f\ A\ B)$$

***

EXEMPLO 2.4

Nos exemplos anteriores, associa-se os símbolos para tipo, pertencentes ao conjunto fundamental $\mathbb{F}$, aos objetos mais "elementares" do sistema. Já um símbolo para tipo

$$\langle\alpha\rangle$$

é associado a um conjunto de objetos que estão associados ao símbolo para tipo α. Tem-se também que símbolos para tipo como $\langle\alpha \succ \beta\rangle$ são associados às funções e aos predicados. Considere agora $\langle list,1\rangle$ um elemento do conjunto construtor $\mathbb{C}$ e os inteiros como sendo constantes associadas ao símbolo para tipo "int", tal que $int \in \mathbb{F}$. Associa-se as listas de inteiros ao símbolo para tipo

$$(list\ int)$$

Intuitivamente, esta é a razão da denominação do conjunto $\mathbb{C}$ como sendo construtor. Cada elemento de $\mathbb{C}$, $\langle list,1\rangle$ por exemplo, determina uma construção, que no presente caso é uma lista de inteiros. Considere também $\langle bag,2\rangle \in \mathbb{C}$, "char" $\in \mathbb{F}$, então $(bag\ int\ char)$ é o símbolo para tipo associado a "bags" de inteiros e de caracteres ("bag" é um conjunto com repetições de elementos).

Deve-se observar que se $\langle conj\ 1\rangle \in \mathbb{C}$ então um conjunto de números inteiros pode estar associado a qualquer um dos seguintes símbolos para tipo:

$$(int),\ (conj\ int)$$

Esta redundância é mantida no sistema com o objetivo de simplificar a representação de outras estruturas como, por exemplo, as listas.

***

2.2.2 DEFINICAO. Ordem de um simbolo para tipo.

A ordem de um símbolo para tipo é definida recursivamente como se segue. Dado $\alpha \in \mathbb{T}$, a sua ordem é indicada por $ordem[\alpha]$.

(i) Se $\alpha \in \mathbb{F}$, então $ordem[\alpha] = 0$.

(ii) Se $\alpha = (c\ \alpha_1 \ldots \alpha_n)$, tal que $\langle c\ n\rangle \in \mathbb{C}$, $\alpha_i \in \mathbb{T}\ \forall\ i$, então

$$ordem[\alpha] = k+1$$

onde

$$k = \max_{1 \le i \le n} \{\ ordem[\alpha_i]\ \}$$

(iii) Se $\alpha \in \mathbb{V}$ e ordem[α] = k, então

$$ordem[(\alpha)] = k+1.$$

(iv) Se $\alpha = \langle \tau \succ \beta \rangle$, então

$$ordem[\alpha] = k+1$$

onde

$$k = max\ (\ ordem[\tau],\ ordem[\beta]\ )$$

***

Observe que a ordem de um símbolo para tipo caracteriza uma hierarquia entre os diferentes símbolos para tipo.

EXEMPLO 2.5

Considerando que $\alpha, t \in \mathbb{F}$, então o símbolo a seguir possui ordem igual a 3.

$$\langle\langle\alpha \succ t\rangle \succ \langle\alpha\rangle\rangle \succ \langle t \succ \alpha\rangle\rangle$$

***

2.2.2.2 Metodo para determinacao da ordem.

Um método para a deteminação da ordem de um símbolo para tipo é o seguinte:

(i) Faça k=0.

(ii) Percorra o símbolo para tipo da esquerda para direita e faça:

(a) k = k+1 para cada parêntese aberto "<".

(b) k = k-1 para cada parêntese fechado ">".

(iii) A ordem do símbolo para tipo é igual a valor máximo atingido por k.

Observe que a ordem de um símbolo para tipo caracteriza uma hierarquia entre os diferentes símbolos para tipo. As medidas indicadas nos exemplos anteriores estão associadas a um símbolo para tipo cuja ordem é zero. As listas de inteiros estão associadas a um símbolo para tipo cuja ordem é 1, pois ordem[int] = 0, logo ordem[<list int>]=1. Já o predicado (boyle X Y) está associado a um símbolo para tipo cuja ordem é 2,

$$ordem[\ (med \succ (med \succ t))\ ] = 2$$

Observe que é exatamente este fato que impede o tratamento de predicados como (boyle X Y), como variáveis livres em programação lógica de primeira ordem. Em lógica de primeira ordem, as únicas variáveis permitidas são objetos

cuja ordem é zero [Shoenfield, 1967], [Lloyd, 1986].

A hierarquia entre os diferentes símbolos para tipo caracteriza também uma hierarquização do conhecimento representado, como é analisado a seguir.

2.2.3 DEFINICAO. Relacao tipada baseada em um conjunto D.

Uma relação tipada baseada em um conjunto não vazio D é definida recursivamente como se segue. Seja $\alpha \in \mathbb{F}$ o símbolo para tipo associado aos elementos de D.

(i) Se $x \in D$, então x é uma relação que está associada ao símbolo para tipo α.

(ii) Se $A=(x_1,x_2,\ldots,x_n)$, tal que $x_i \in D$ $i=1,2,\ldots n$, então A é uma relação que está associada ao símbolo para tipo $\langle\alpha$.

(iii) Se B é uma lista, árvore, bag, etc, formada por elementos pertencentes a D, então B é uma relação que está associada ao símbolo para tipo $\langle c\ \alpha$, onde "c" é o identificador de B. Isto é, "c" identifica se B é uma lista, uma árvore, um bag, etc.

(iv) Uma relação R associada ao símbolo para tipo $(\alpha_1 \succ \alpha_2)$ é um conjunto $\mathbb{R}$ de pares ordenados tais que:

$$\langle r_1, r_2 \rangle \in \mathbb{R}$$

se e somente se r_i é uma relação associada ao símbolo para tipo α_i.

***

NOTACAO: Dada uma relação R associada a um símbolo para tipo α, então diz-se que R é do tipo α, ou que o tipo de R é α. Além disso, utiliza-se notação

$$\text{tipo}[R] = \alpha$$

ou associa-se um subíndice a R, isto é, R_α.

***

Se os elementos de D estão associados ao símbolo para tipo $\alpha \in \mathbb{F}$, cuja ordem é 0, então os conjuntos formados a partir de tais elementos estão associados ao símbolo para tipo $\langle\alpha$, cuja ordem é 1. Da mesma forma, os conjuntos de conjuntos de tais elementos estão associados a símbolos para tipo cuja ordem é 2. Além disso, uma relação de pares $\langle r_1, r_2 \rangle$, tais que α_i é o símbolo para tipo associado a r_i, $1 \leq i \leq 2$, é um conjunto associado a um símbolo para tipo $\langle\alpha_1 \succ \alpha_2\rangle$, cuja ordem é:

$$ordem[(\alpha_1 \succ \alpha_2)] = 1 + max\ (ordem[\alpha_1],\ ordem[\alpha_2])$$

Desta forma, a relação **ordem**[α], que fornece a ordem do símbolo para tipo α, estabelece uma ordenação do conjunto $\mathbb{T}$ de todos os símbolos para tipo. Tem-se também uma ordenação do conjunto das relações tipadas baseadas em um conjunto D. Dadas duas relações R_α e Q_β, então

$$R_\alpha < Q_\beta$$

se e somente se

$$ordem[\alpha] < ordem[\beta]$$

A extensão da ordenação dos símbolos para tipo, aos objetos formados a partir dos elementos de D caracteriza tais objetos hierarquicamente.

EXEMPLO 2.6

Dado um conjunto D, uma caracterização hierárquica dos objetos formados a partir dos seus elementos é obtida associando símbolos para tipo a tais objetos. Considerando

D = (**temperatura, pressao, volume, densidade**),

constrói-se uma relação tipada baseada em D. Tem-se, por exemplo, que o conjunto (**temperatura, volume**) é uma relação associada ao símbolo para tipo (**med**), e a lista [**temperatura, volume**] é uma relação associada ao símbolo para tipo (**list med**), já que seus elementos estão associados ao símbolo para tipo "**med**". O predicado (**boyle** X Y) é uma relação associda ao símbolo para tipo

(**med** ≻ **med** ≻ **t**)

Observe então que há uma hierarquia entre (**temperatura, volume**), [**temperatura, volume**] e (**boyle** X Y) e todos os outros objetos formados a partir dos elementos do conjunto D. Tem-se uma hierarquia caracterizada pela ordem dos símbolos para tipos associados.

***

2.2.4 DEFINICAO. Alfabeto.

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

(I) Simbolos próprios.

(a) Associado a cada símbolo para tipo α existe um conjunto enumerável de variáveis do tipo α.

(b) Associado a cada símbolo para tipo α, também existe um conjunto enumerável de constantes do tipo α, disjunto do conjunto das variáveis.

(II) Símbolos impróprios:

Como símbolos impróprios tem-se os parentêses, ")", "(" e o símbolo lambda "λ".

***

NOTACAO: Dado um símbolo próprio qualquer, serão utilizados subíndices para a identificação do símbolo para tipo associado ao símbolo próprio. Assim, se A_α é um o símbolo próprio, então o símbolo para tipo associado a A_α é α. Será utilizada também a notação

$$tipo[A_\alpha] = \alpha$$

indicando que o símbolo para tipo associado a A_α é α. Os conjuntos das variáveis e das constantes associadas ao símbolo para tipo α serão indicados por VAR_α e $CONST_\alpha$, respectivamente.

***

2.2.4.1 Constantes lógicas.

As constantes lógicas formam uma classe de constantes do sistema lógico. Tais constantes são indicadas a seguir, com os respectivos símbolos para tipos associados. Considera-se que $\iota \in \mathbb{T}$.

constantes	tipo
$\wedge$	$(\iota \succ \iota \succ \iota)$
$\vee$	$(\iota \succ \iota \succ \iota)$
$\Rightarrow$	$(\iota \succ \iota \succ \iota)$
$\neg$	$(\iota \succ \iota)$
T	ι
Π_α	$((\alpha \succ \iota) \succ \iota)$
Σ_α	$((\alpha \succ \iota) \succ \iota)$

Informalmente, pode-se dizer que o significado semântico das constantes lógicas é aquele indicado no estudo da lógica de primeira ordem. A constante lógica "$\vee$", por exemplo, é o conectivo OU, cujo tipo é:

$$(\iota \succ (\iota \succ \iota))$$

Isto significa que aplicando $\vee$ a uma constante A_ι, tem-se como resultado $(\vee\ A_\iota)$, que é do tipo $(\iota \succ \iota)$. Em seguida aplicando $(\vee\ A_\iota)$ a uma outra constante B_ι, tem-se

$$((\vee\ A_\iota)\ B_\iota)$$

que é do tipo "ι". Finalmente, para simplificar, utiliza-se a notação infixa

$(A \vee B)$ no lugar de $((\vee\ A)\ B)$.

isto é,

$$(A \vee B) \equiv ((\vee\ A)\ B)$$

$$(A \wedge B) \equiv ((\wedge\ A)\ B)$$

$$(A \Rightarrow B) \equiv ((\Rightarrow\ A)\ B)$$

As constantes Π_α e Σ_α são utilizadas para definir os quantificadores universal e existencial respectivamente. Tem-se por definição as seguintes correspondências:

$$\Pi_\alpha(\lambda\ X_\alpha\ A_t) \equiv \forall X_\alpha\ (A_t)$$

$$\Sigma_\alpha(\lambda\ X_\alpha\ A_t) \equiv \exists X_\alpha\ (A_t)$$

EXEMPLO 2.7

Considere o caso em que

$$A_t \equiv (2_{int} = X_{int})$$

onde o símbolo de igualdade "=" é um predicado de tipo apropriado, cuja "interpretação" é a usual. Neste caso, a fórmula

$$\Pi_{int}(\lambda\ X_{int} A_t)$$

é falsa, pois

$$\Pi_{int}(\lambda\ X_{int} A_t) \equiv \forall X_{int}(A_t)$$

Por outro lado,

$$\Sigma_{int}(\lambda\ X_{int} A_t)$$

é verdadeira, pois

$$\Sigma_{int}(\lambda\ X_{int} A_t) \equiv \exists X_{int}(A_t)$$

A decisão sobre a veracidade das fórmulas acima é informal, pois ainda não foi definido o conceito de interpretação. Esta definição é analisada mais adiante.

***

2.2.5 DEFINICAO. Formulas.

O conjunto das fórmulas do tipo α, $\alpha \in \mathbb{T}$, indicado por $FORM_\alpha$, é definido indutivamente pelas seguintes regras.

(i) Variáveis ou constantes do tipo α são fórmulas do tipo α.

(ii) Seja X_α uma variável e H_β uma fórmula, então $(\lambda\ X_\alpha H_\beta)$ é uma fórmula do tipo

$$(\alpha \succ \beta)$$

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

***

NOTACAO: Quando não ocorrer dúvida na interpretação das fórmulas

$$(\lambda\ A_{\alpha} H_{\beta})$$

e

$$(F_{\alpha} G_{\beta}),$$

os parênteses serão omitidos. Além disso, as equivalências a seguir serão consideradas.

$$(\lambda X_1\ \lambda X_2\ \ldots\ \lambda X_n\ F) \equiv (\lambda X_1\ (\lambda X_2\ (\ldots\ (\lambda X_n\ F)\ldots)$$

e

$$F\ A_1\ A_2\ \ldots\ A_m \equiv (\ldots(F\ A_1)\ A_2)\ \ldots\ A_m)$$

***

Informalmente, fórmulas construídas utilizando o símbolo "λ", determinam novas funções no contexto da teoria. Em outras palavras, a partir de uma variável X_{α}, e de uma fórmula H_{β}, obtém-se a fórmula

$$(\lambda\ X_{\alpha} H_{\beta}),$$

que é do tipo $(\alpha \succ \beta)$. A interpretação formal de tais fórmulas é considerada a seguir.

2.3 INTERPRETACAO.

As fórmulas do tipo $(\alpha \succ \beta)$ são interpretadas como funções cujo domínio é formado por objetos do tipo "α" e o contradomínio por objetos do tipo "β". Já as fórmulas $(F_{(\alpha\succ\beta)} G_{\alpha})$ são interpretadas como a aplicação de um símbolo funcional a uma fórmula. A seguir é analisada a interpretação formal das fórmulas do sistema logico. Com esta interpretação, as fórmulas $(\lambda\ X_{\alpha} H_{\beta})$ e $(F_{(\alpha\succ\beta)} G_{\alpha})$ possuem a semântica indicada acima. Isto é, tem-se uma função e a aplicação de uma função respectivamente. Inicia-se a análise pela definição de subfórmula.

2.3.1 DEFINICAO Subformula.

Sejam as fórmulas F e H. H é uma subfórmula de F se e somente se:

(i) H = F.

(ii) F = (AB), onde A e B são fórmulas e H é uma subfórmula de A ou de B.

(iii) F = (λ X A), onde X é variável, A é fórmula e H é uma subfórmula de A, ou H = X.

***

2.3.2 DEFINICAO Abstracao e ligacao de variaveis.

Dada uma variável X_α e uma fórmula C_β; então a fórmula (λ X_α C_β) é uma abstração de C_β por X_α. Neste caso, a abstração é ligada por X_α e seu escopo é C_β.

***

2.3.3 DEFINICAO Ocorrencia livre de uma variavel.

Uma variável X_α ocorre livre em uma fórmula B_γ se e somente se B_γ não possui uma subfórmula que é uma abstração ligada por X_α.

***

2.3.4 DEFINICAO. Designacao.

Uma designação é uma função φ cujo domínio é o conjunto de todas as variáveis e o contradomínio é o conjunto de todas as fórmulas, isto é,

$$\varphi: \bigcup_{\alpha\in\mathbb{T}} (VAR_\alpha) \longrightarrow \bigcup_{\beta\in\mathbb{T}} (FORM_\beta)$$

Além disso, a aplicação de uma designação a uma variável X_β é uma fórmula do tipo β,

$$tipo[\varphi(X_\beta)] = tipo[X_\beta] = \beta$$

***

Informalmente, uma designação é uma função que associa a cada variável uma fórmula do mesmo tipo. Tem-se uma substituição da variável pela fórmula conforme a designação.

2.3.5 DEFINICAO. Estrutura padrao baseada em D_i, $1 \leq i \leq n$.

Uma estrutura padrão baseada em uma família finita de conjuntos não vazios D_i é uma família

$$(M_\gamma)_{\gamma \in \mathbb{T}}$$

tal que:

(i) $M_{\alpha_i} = D_i$ onde α_i é o símbolo para tipo associado aos elementos de D_i.

(ii) Dado $M_{\langle\delta\rangle}$, $\delta \in \mathbb{T}$, então $M_{\langle\delta\rangle}$ é o conjunto das partes de M_δ.

(iii) Dado M_η, $\eta \in \mathbb{T}$ tal que $\eta = (c\ \sigma_1 \sigma_2 \ldots \sigma_n)$ onde $\langle c,n\rangle \in \mathbb{C}$, $\sigma_i \in \mathbb{T}$, $1 \leq i \leq n$, então M_η é o conjunto de todas as estruturas formadas com objetos associados aos símbolos para tipo

$$\alpha_1, \ldots, \alpha_n$$

e definidos conforme o símbolo "c".

(iv) $M_{\alpha\beta} = \{f$ tal que $f: M_\alpha \dashrightarrow M_\beta\}$

***

Assim, um elemento de $M_{\alpha\beta}$ é uma função de M_α em M_β. Informalmente, se

D = (temperatura, pressao, volume, densidade)

e o símbolo para tipo associado aos elementos de D é "med", então $M_{med} = D$ e $M_{(med)med}$ é o conjunto das funções cujo domínio e contradomínio é D. $M_{(list med)}$ é o conjunto de todas as listas cujos elementos são objetos de D.

2.3.6 DEFINICAO. Interpretacao padrao baseada em D_t, $1 \leq i \leq n$.

Uma interpretação padrão baseada em uma família finita de conjuntos não vazios D_t é uma família

$$(M_\alpha, V)_{\alpha \in \mathbb{T}}$$

tal que:

(i) $(M_\alpha)_{\alpha \in \mathbb{T}}$ é uma estrutura padrão baseada na família finita de conjuntos D_t, $1 \leq i \leq n$.

(ii) V é uma função significado tal que, dada uma designação φ tem-se:

(a) $V(A, \varphi) = A$, se $A \in CONST$.

(b) $V(X, \varphi) = \varphi(X)$, se $X \in VAR$.

(c) $V(F_{\alpha\beta}\, G_\alpha, \varphi) = V(F_{\alpha\beta}, \varphi)(V(G_\alpha, \varphi))$

(d) $V(\lambda X_\alpha H_\beta, \varphi) = f$, onde f é tal que:

$$f \in M_{\alpha\beta}$$

isto é,

$$f: M_\alpha \longrightarrow M_\beta$$

e para toda fórmula B_α,

$$f(B_\alpha) = V(H_\beta, \varphi_{B_\alpha})$$

onde

$$\varphi_{B_\alpha}(X) = \begin{cases} B_\alpha \text{ se } X = X_\alpha \\ \varphi(X) \text{ se } X \neq X_\alpha \end{cases}$$

***

A interpretação formal da fórmula $(\lambda X_\alpha H_\beta)$ é uma função f cujo domínio é formado pelo conjunto das fórmulas do tipo α, que é o mesmo tipo da variável X_α. Já o contradomínio é formado por fórmulas do tipo β, que é o mesmo tipo da fórmula H_β. O resultado da aplicação de f a uma fórmula B_α é dado por:

$$f(B_\alpha) = V(H_\beta, \varphi_{B_\alpha})$$

Assim, a aplicação de f em B_α é o significado da fórmula H_β, no caso em que a designação considerada é φ_{B_α}.

Informalmente, $f(B_\alpha)$ é o significado de H_β no caso em que cada ocorrência livre de X_α em H_β é substituída por B_α.

Considere os exemplos a seguir, nos quais as operações algébricas "+" e "-" são as usuais.

EXEMPLO 2.8

Neste exemplo considera-se a interpretação de uma fórmula do tipo

$$(\lambda\ X_{int}\ H_{int}),$$

$int \in \mathbb{T}$. O objetivo desta análise é indicar todo o desenvolvimento da aplicação da função

$$(\lambda\ X_{int}\ H_{int})$$

ao seu argumento. Considere o caso em que:

$$H_{int} = X_{int} + 2_{int}$$

onde X_{int} é uma variável do tipo "int" (inteira) e 2_{int} é uma constante. Neste caso, escrevendo H_{int} na forma prefixa tem-se,

$$H_{int} = ((+\ X_{int})\ 2_{int})$$

e obtem-se

$$(\lambda\ X_{int}H_{int}) \equiv [\lambda\ X_{int}\ ((+\ X_{int})\ 2_{int})]$$

onde se utiliza os símbolos "[" e "]" no lugar de parênteses para facilitar a leitura. A interpretação desta fórmula é uma função cujo domínio e contradomínio é o conjunto dos inteiros. Isto é, dada uma designação φ, e uma interpretação padrão

$$M = (M_\alpha\ ,\ V)_{\alpha\in\mathbb{T}}$$

então o significado de $(\lambda\ X_{int}H_{int})$ em relação à interpretação M é dada por:

$$V[(\lambda\ X_{int}H_{int}),\ \varphi] = f$$

onde

$$f:\ M_{int} \longrightarrow M_{int}$$

A interpretação de

$$(\lambda\ X_{int}H_{int})\ 4_{int}$$

é a dada por:

$$V[(\lambda\ X_{int}H_{int})\ 4_{int}, \varphi]$$

Por definição,

$$V[F_{\alpha\beta}G_{\alpha}, \varphi] = V[F_{\alpha\beta}, \varphi]\ V[G_{\alpha}, \varphi] =$$

logo,

$$V[(\lambda\ X_{int}H_{int})\ 4_{int}, \varphi] =$$

$$= V[(\lambda\ X_{int}H_{int}), \varphi]\ \ V[4_{int}, \varphi]$$

Novamente, por definição,

$$V[4_{int}, \varphi] = 4_{int}$$

$$V[\lambda\ X_{int}H_{int}, \varphi] = f$$

Portanto,

$$V[(\lambda\ X_{int}H_{int})\ 4_{int}, \varphi] =$$

$$= f\ V[4_{int}, \varphi]$$

$$= f\ 4_{int}$$

isto é, tem-se a aplicação de f em 4_{int}. Aplicando a definição da função significado V obtem-se:

$$
\begin{aligned}
f\ 4_{int} &= \\
&= V[(\lambda\ X_{int}\ H_{int})\ ,\ \varphi]\ 4_{int} \\
&= V[H_{int},\ \varphi_{4_{int}}] \\
&= V[((+\ X_{int})\ 2_{int})\ ,\ \varphi_{4_{int}}] \\
&= V[+\ X_{int},\ \varphi_{4_{int}}]\ V[2_{int},\ \varphi_{4_{int}}] \\
&= V[+\ X_{int},\ \varphi_{4_{int}}]\ 2_{int} \\
&= V[+\ ,\ \varphi_{4_{int}}]\ V[X_{int},\ \varphi_{4_{int}}]\ 2_{int} \\
&= +\ \ \varphi_{4_{int}}[X_{int},\ \varphi_{4_{int}}]\ \ 2_{int} \\
&= +\ \ 4_{int}\ \ 2_{int} = 6_{int}
\end{aligned}
$$

Em outras palavras, a interpretação de

$$(\lambda\ X_{int}(+\ X_{int}\ 2_{int}))\ 4_{int}$$

é $4_{int}+\ 2_{int}$, que equivale à substituição de 4_{int} no lugar de X_{int} em H_{int}.

***

EXEMPLO 2.9

O resultado do exemplo anterior é geral, isto é, a interpretação da fórmula

$$(\lambda\ X\ J)\ A$$

é a substituição das ocorrências livres de X em J pela fórmula A. Considere, por exemplo,

$$J_{(list\ int)} = [X_{int},1,2,X_{int},3]$$

Seguindo o mesmo raciocínio do exemplo anterior, tem-se

$$(\lambda\ X_{int}J_{(list\ int)})0 = [0,1,2,0.3]$$

onde fica implícita a aplicação da função significado e a designação.

OBSERVACAO: A lista de inteiros indicada neste exemplo é formada utilizando o predicado "CONS" do tipo

$$\langle int \succ \langle list\ int \rangle \succ \langle list\ int \rangle \rangle$$

cuja interpretação é análoga à do predicado "CONS" em LISP, [Winston, 1984]. Assim,

$$[X_{int}, 1, 2, X_{int}, 3] =$$

$$= (\text{CONS } X_{int} (\text{CONS } 1\ (\text{CONS } 2\ (\text{CONS } X_{int} (\text{CONS } 3\ \text{NIL})))))$$

***

EXEMPLO 2.10

Quando ocorrem vários símbolos λ em uma fórmula, como por exemplo,

$$(\lambda\ X_{int}\ (\lambda\ Y_{int}\ K_{\langle list\ int \rangle}))\ H_{int}\ 7$$

a sequência de substituições das variáveis X_{int} e Y_{int} deve ser observada. Neste caso, X_{int} é substituída por H_{int} nas ocorrências livres de X_{int} em $K_{\langle listint \rangle}$ e Y_{int} é substituída por 7_{int} nas ocorrências livre de Y_{int} em $K_{\langle listint \rangle}$. Considere,

$$K_{\langle list\ int \rangle} = [X_{int}, Y_{int}, 2, Z_{int}, 3]$$

Neste caso,

$$(\lambda\ X_{int} (\lambda\ Y_{int} K_{\langle list\ int \rangle}))$$

é interpretada como uma função cujo domínio e contradomínio são, respectivamente,

$$M_{int},\ M_{\langle int \succ \langle list\ int \rangle \rangle}$$

Considere também que,

$$H_{int} = X_{int} + 2_{int}$$

logo,

$$(\lambda\ X_{int} (\lambda\ Y_{int} K_{\langle list\ int \rangle}))\ H_{int}\ 7 =$$

$$= \lambda\ Y_{int} [H_{int}, Y_{int}, 2, Z_{int}, 3]\ 7$$

$$= \lambda\ Y_{int} [X_{int} + 2, Y_{int}, 2, Z_{int}, 3]\ 7$$

$$= [X_{int} + 2, 7, 2, Z_{int}, 3]$$

Novamente, fica implícita a aplicação da função significado e da função designação.

***

OBSEVACAO: Neste trabalho, as fórmulas e suas interpretações se confundem. Assim, a menos que seja necessário, a interpretação de uma fórmula não é indicada explicitamente.

***

EXEMPLO 2.11

Dada uma fórmula $(\lambda\ X\ T)$, então a interpretação de

$$(\lambda\ X\ T)\ F$$

é a substituição das ocorrências livres de X em T pela fórmula F. Deve-se enfatizar que tais substituições levam em conta apenas as ocorrências de X em T, que são livres. Considere o caso específico em que,

$$T_{(list\ \alpha)} = [\lambda\ Y_{int}(X_{int} + Y_{int}), \lambda\ X_{int}(X_{int} + Y_{int})]$$

onde $\alpha = (int \succ int)$. Neste caso,

$$(\lambda\ X_{int} T_{(list\alpha)})\ 2 =$$

$$= [\lambda\ Y_{int}(2 + Y_{int}), \lambda\ X_{int}(X_{int} + Y_{int})$$

Observe que apenas a variável X_{int} da fórmula

$$\lambda\ Y_{int}(X_{int}+Y_{int})$$

é substituída por 2. Isto se deve ao fato de que X_{int} é livre nesta fórmula e ligada na fórmula

$$\lambda\ X_{int}(X_{int} + Y_{int})$$

Omitindo os subíndices que determinam os símbolos para tipo, tem-se que,

$$((\lambda\ X\ (\lambda\ Y\ T))\ 2\ )\ 3 =$$

$$= (\lambda\ X\ (\lambda\ Y\ \ [\lambda\ Y\ (X + Y), \lambda\ X\ (X + Y)]\ ))\ 2\ 3$$

$$= (\lambda\ Y\ \ [\lambda\ Y\ (2 + Y), \lambda\ X\ (X + Y)]\ )\ 3$$

$$= [\lambda\ Y\ (2 + Y), \lambda\ X\ (X + 3)]$$

Portanto, apenas as ocorrências livre de X e Y em T são substituídas por 2 e 3 respectivamente.

***

OBSERVACAO: Para toda variável X e fórmulas A, B, tem-se que,

$$(\lambda\ X(\lambda\ X\ A))\ B\ =\ \lambda\ X\ A$$

Isto se deve ao fato de que X não ocorre livre em $(\lambda\ X\ A)$.

***

Finalmente, as constantes lógicas Π_α e Σ_α, utilizadas para representar os quantificadores universal e existencial respectivamente, são interpretadas formalmente como se segue.

$$V(\Pi_\alpha,\varphi) \in M_{\langle\alpha \succ \iota\rangle \succ \iota}$$

$$V(\Pi_\alpha,\varphi)(\lambda\ Y_\alpha A_\iota) \in M_\iota$$

$$V(\Pi_\alpha,\varphi)(\lambda\ Y_\alpha A_\iota) \equiv \forall Y_\alpha(A_\iota)$$

$$V(\Sigma_\alpha,\varphi) \in M_{\langle\alpha \succ \iota\rangle \succ \iota}$$

$$V(\Sigma_\alpha,\varphi)(\lambda\ Y_\alpha A_\iota) \in M_\iota$$

$$V(\Sigma_\alpha,\varphi)(\lambda\ Y_\alpha A_\iota) \equiv \exists X_\alpha(A_\iota)$$

2.4 λ-CONVERSAO.

Nesta seção são analisadas as operações de λ-conversão, que especificam quando duas fórmulas são equivalentes. A importância desse estudo é que fórmulas equivalentes, determinadas por operações de λ-conversão, representam um mesmo conhecimento e estas operações eliminam redundâncias na representação de conhecimento. Além disso, dado um conjunto de fórmulas do λ-cálculo tipado, todas equivalentes, é possível a determinação de uma fórmula que represente este conjunto. A fórmula encontrada é única, a menos de renomeação de variáveis, e para todo conjunto de fórmulas equivalentes sempre é possível determiná-la.

2.4.1 DEFINICAO Substituicao.

Sejam X_α uma variável e G_α, F fórmulas. Utiliza-se

$$S^X_G\ F$$

para denotar o resultado da substituição de todas as ocorrências livres de X_α em F por G_α. Esta operação é definida recursivamente a seguir.

(i) Se F é uma variável ou constante, tem-se:

caso 1. Se $F = X_\alpha$, então: $S^X_\alpha F = G$

caso 2. Se $F \neq X_\alpha$, então: $S^X_\alpha F = F$

(ii) Se $F = (\lambda\ YC)$, tem-se:

caso 1. Se $X_\alpha = Y$, então: $S^X_\alpha F = F$

caso 2. Se $X_\alpha \neq Y$ então: $S^X_\alpha F = (\lambda\ Y\ (S^X_\alpha C))$

(iii) Se $F = (C\ D)$, então: $S^X_\alpha F = ((S^X_\alpha C)\ (S^X_\alpha D))$

***

EXEMPLO 2.12

Como é definido acima, dadas as fórmulas F e G, então

$$S^X_\alpha F$$

é a substituição das ocorrências livres de X em F por G. Naturalmente, tal substituição só é possível quando X e G estão associadas ao mesmo símbolo para tipo. Considere o caso específico em que

$$F = (\lambda\ Y_{int}(\lambda\ Z_{int}((Z_{int}+Y_{int}) = W_{int})))$$

$$G_{int} = Y_{int}+4_{int}$$

Observe que o símbolo para tipo associado a G e W são os mesmos, logo é possível escrever $S^W_\alpha F$. Além disso, como W ocorre livre em F, tem-se que,

$$S^W_\alpha F = (\lambda\ Y\ (\lambda\ Z\ ((Z + Y) = (Y + 4))))$$

onde, para simplificar a notação, os símbolos para tipo são omitidos.

Finalmente, observe que Y_{int} é uma variável livre em G e que passa a ser ligada em $S_G^{Y_{int}}$ F. Substituições como esta modificam a classificação das variáveis das fórmulas. Variáveis livres em G podem se tornar ligadas no resultado da substituição S_G^Y F. Para evitar tal fato, consideram-se a seguir casos específicos de substituicoes, onde a classificação das variáveis é invariante quando se realiza uma substituição.

***

A próxima definição relaciona as fórmulas G_α, X_α e F de tal forma que o resultado da substituição de X_α por G_α em F não modifica a classificação das variáveis envolvidas.

2.4.2 DEFINICAO. "G_α e livre para X_α em F".

Sejam X_α uma variável e as fórmulas G_α e F. Diz-se que G_α é livre para X_α em F se e somente se F **NAO** possui uma subfórmula $(\lambda\ YC)$, que satisfaça ao mesmo tempo as duas condições a seguir.

(a) - X_α é livre no escopo da abstração $(\lambda\ YC)$.

(b) - A variável Y ocorre livre em G.

***

Observa-se que na substituição da variável X_α por G_α em F, no caso em que "G_α é livre para X_α em F", as variáveis livres em G_α continuam livres no resultado. Isto é, se Y ocorre livre em G, então Y também ocorre livre em S_G^X F.

NOTACAO: No que se segue, quando for escrito

$$S_G^X\ F$$

estará implícito que

$$tipo[X] = tipo[G]$$

e que G é livre para X em F.

***

Dadas duas fórmulas do mesmo tipo F_α e G_α pode ocorrer que elas sejam diferentes apenas por uma nomeação de suas variáveis livres. Em outras palavras, são utilizados símbolos diferentes para nomear suas variáveis livres. Neste caso, as fórmulas F_α e G_α são equivalentes. Os outros casos em que duas fórmulas do mesmo tipo são equivalentes são analisados a partir das operações de λ-conversão definidas a seguir.

2.4.3 DEFINICAO. Operacoes de λ-conversao.

As operações de λ-conversão são,

(i) α-conversão: é a substituição de $(\lambda\ XF)$ por

$$(\lambda\ Y\ (S^X_Y F))$$

ou a substituição de (AB) por

$$(S^X_Y A)(S^X_Y B)$$

desde que $Y \in VAR$, Y seja livre para X em F, em A e em B.

(ii) β-redução: é a substituição de

$$(\lambda\ XF)G$$

por

$$S^X_G F$$

desde que G seja livre para X em F.

(iii) β-expansão: é a operação inversa da operação realizada no item (ii). Isto é, se N é o resultado da aplicação de β-redução sobre a fórmula R, então substitua N por R.

Há sistemas lógicos de ordem superior, aos quais são adicionadas mais duas operações de λ-conversão. Tais operações caracterizam o axioma da extensionalidade [Andrews, 1986] e são apresentadas a seguir.

(iv) η-redução: é a substituição da fórmula

$$(\lambda\ X(FX))$$

por F, desde que X não ocorra livre em F.

(v) η-expansão: é o inverso de (iv). Isto é, substituir uma fórmula F do tipo $(\alpha \succ \beta)$ por uma fórmula

$$(\lambda\ X_\alpha (F_{(\alpha \succ \beta)} X_\alpha))$$

onde X_α não ocorre livre em $F_{(\alpha \succ \beta)}$.

***

2.4.4 DEFINICAO. β-redex, η-redex.

Diz-se que uma fórmula é um β-redex se ela é uma fórmula do tipo

$$(\lambda X\ H)F$$

onde F é livre para X em H.

Da mesma forma, diz-se que uma fórmula é um η-redex se ela é do tipo

$$\lambda X\ (FX)$$

onde X não ocorre livre em F.

***

Observe que é possivel aplicar uma β-redução sobre um β-redex, como também é possivel aplicar uma η-redução sobre um η-redex. Por outro lado, diz-se que uma fórmula não possui β-redex (η-redex), quando ela não possui uma subfórmula que é um β-redex (η-redex).

EXEMPLO 2.13

Na definição acima, a α-conversão determina a renomeação das variáveis que ocorrem livres em uma fórmula. Assim, a substituição de

$$\lambda X\ (YZX)$$

por

$$\lambda X\ (WSX)$$

é uma α-conversão. A β-redução de uma fórmula do tipo $(\lambda\ XF)G$ é a determinação do resultado da aplicação da função $(\lambda\ XF)$ ao seu argumento G. Considere os dois casos a seguir,

(1) Sejam as fórmulas

$$F \equiv X_{int} + 6_{int}$$

$$G \equiv 8_{int}$$

então o resultado da aplicação de uma β-redução sobre $(\lambda\ XF)G$ é dada pela fórmula

$$(8_{int} + 6_{int})$$

(ii) Sejam agora as fórmulas

$$F \equiv Y_{int} + 6_{int}$$

$$G \equiv 8_{int}$$

então o resultado da aplicação de uma β-redução sobre $(\lambda\ XF)G$ é

$$(Y_{int} + 6_{int})$$

Observe que no primeiro caso há uma substituição da variável X_{int}, o que não ocorre no segundo caso.

***

A operação de η-redução determina uma simplificação da fórmula

$$\lambda X\ (FX)$$

para a fórmula

$$F$$

Observe que para toda fórmula B tal que

$$tipo[B] = tipo[X]$$

entao,

$$(\lambda X\ (FX))B \equiv FB$$

Isto é, a fórmula $\lambda X\ (FX)$ comporta como se fosse a função F, o que determina as razões para a substituição acima. Entretanto, deve-se observar que há sistemas lógicos onde se tem

$$HB \equiv GB$$

para toda fórmula B, mas que $F \neq G$ [Martins, 1988].

2.4.5 DEFINICAO λ-conversao de formulas.

Considere duas fórmulas F e G. Diz-se que G é uma λ-conversão de F, se é possível obter G a partir de F utilizando uma série de operações de λ-conversão aplicadas sobre subformulas de F. Isto é,

$$F \xleftrightarrow{\text{operacoes de } \lambda\text{-convercao}} G$$

No caso em que se utiliza somênte as operações α-conversão, β-redução e β-expansão, diz-se que G é uma β-conversão de F. Isto é,

$$F \xleftrightarrow[\beta\text{-reducao}]{\alpha\text{-conversao}}{}^{\beta\text{-expansao}} G$$

Quando se utiliza apenas α-conversão tem-se que G é uma α-conversão de F. Isto é

$$F \xleftrightarrow{\alpha\text{-conversao}} G$$

***

LEMA 2.1.

As operações de λ-conversão possuem inversas. Isto é, se F é uma λ-conversão de G, então G é uma λ-conversão de F.

DEMONSTRACAO.

Decorre da definição anterior.

***

A seguir define-se o conceito de relação de equivalência, conforme [Lipschutz, 1972], [Simis, 1977].

2.4.6 DEFINICAO. Relacao de equivalencia [Lipschutz, 1972].

Dada uma relação R em um conjunto D, então R é uma relação de equivalência se e somente se, para todo x, y, z $\in$ D,

(i) x R x.

(ii) x R y, então y R x.

(iii) x R y, y R z, então x R z.

***

LEMA 2.2.

As operações de λ-conversão determinam uma relação de equivalência no conjunto das fórmulas do λ-cálculo tipado. Isto é, se R é a relação definida por

$$[\Phi \; R \; \Psi] \text{ se e somente se } [\Phi \xleftrightarrow{\lambda\text{-conversao}} \Psi]$$

onde Φ, $\Psi \in$ FORM, então R é uma relação de equivalência.

DEMONSTRACAO.

Segue do lema 2.1 e da definição anterior.

***

Informalmente, a noção de equivalência entre fórmulas é utilizada para simplificar a representação d conhecimento a partir de fórmulas do λ-cálculo. Em outras palavras, fórmulas equivalentes em um programa representam informações redundantes, isto é,

$$F \equiv H \leftrightarrow F \longleftrightarrow [\ \lambda\text{-conversão}\ ] \longleftrightarrow H$$

Neste caso, basta escrever apenas uma fórmula para cada classe de equivalência, [Lipschutz, 1972], [Simis, 1977]. Define-se a seguir, fórmula normal, que é utilizada como o representante de cada classe de equivalência.

2.4.7 DEFINICAO Formulas normais.

(i) Uma fórmula Φ é β-normal se não é possível aplicar sobre ela a regra de β-redução.

(ii) Uma fórmula Φ é λ-normal se não é possível aplicar sobre ela as regras de β-redução nem de η-redução.

***

Uma fórmula Φ é λ-normal quando ela possui uma das formas a seguir,

(i) $\Phi = (A\ F_1 \ldots F_m)$.

(ii) $\Phi = \lambda\ X_1 \ldots X_n (A\ F_1 \ldots F_m)$

(iii) $\Phi = A$

(iv) $\Phi = \lambda\ X_1 \ldots X_n\ A$

onde A é uma constante ou variável e as fórmulas $F_1,\ldots,F_m$ também possuem a mesma forma. As variáveis $X_1,\ldots,X_n$ são as ligações, "A" é denominado a cabeca e $F_1,\ldots,F_m$ são os argumentos da fórmula.

Informalmente, a forma β-normal de uma fórmula é obtida após repetidas aplicações das operações α-conversão e β-redução. Já a forma λ-normal é obtida após repetidas aplicações das operações α-conversão, β-redução e η-redução. Tais operações são aplicadas até não ser mais aplicável nenhuma redução. Dada uma fórmula Φ, a existência e unicidade de sua forma β-normal, ou λ-normal, é desmonstrada formalmente em [Girard, 1987], [Andrews, 1971] e [Fortune, 1983] e se constitui nos teoremas a seguir.

2.4.8 TEOREMA DA FORMA NORMAL [Girard, 1987].

Dada uma fórmula Φ, então existe uma fórmula λ-normal (β-normal) Ψ, obtida aplicando sobre Φ uma sequência de operações β-redução, η-redução e α-conversão (β-redução e α-conversão).

***

2.4.9 TEOREMA DA UNICIDADE DA FORMA NORMAL [Girard, 1987].

Dado uma fórmula Φ, tem-se a unicidade de sua forma λ-normal (β-normal), a menos de renomeação das variáveis ligadas.

***

2.4.10 TEOREMA DE CHURCH-ROSSER [Girard, 1987].

Se Φ, Ψ são fórmulas λ-normais tais que Ψ é uma λ-conversão de Φ, então Φ é uma α-conversão de Ψ.

***

Os teoremas da forma normal, da unicidade da forma normal e de Church-Rosser garantem não só a existência como também a unicidade do representante de um conjunto de fórmulas equivalentes. Além disso, a decidibilidade sobre a existência da forma normal de uma fórmula possibilita representar informações sem redundâncias. Dada uma fórmula Φ, considera-se a sua forma λ-normal. Outras fórmulas equivalentes a Φ são consideradas iguais a Φ. Neste sentido, tais teoremas são fundamentais na representação de conhecimento a partir de fórmulas do λ-cálculo tipado.

Deve-se observar também que os teoremas acima não são válidos para o λ-cálculo sem tipos, conforme é apresentado em [Dickmann, 1988]. O exemplo a seguir considera a forma λ-normal de uma fórmula.

EXEMPLO 2.14

Neste exemplo, considera-se a obtenção da forma λ-normal de uma fórmula. Sejam $\mathbb{F}=\{int\}$, $\mathbb{C}=\{\langle list,1\rangle\}$ e a fórmula

$$(\lambda\ X.(CONS_{\alpha}\ 1\ (CONS_{\alpha}\ 2\ X)))\ (CONS_{\alpha}\ 3\ NIL_{(list\ int)})$$

onde $NIL_{(list\ int)}$ é uma constante que representa a lista vazia e $CONS_{\alpha}$ é tal que,

$$\alpha = (int \succ (list\ int) \succ (list\ int))$$

A fórmula acima possui a seguinte forma λ-normal:

$$(CONS_{\alpha}\ 1(CONS_{\alpha}\ 2(CONS_{\alpha}\ 3\ NIL_{(list\ int)})))$$

Neste caso não há variáveis de ligações, a cabeca da fórmula é $CONS_{\alpha}$ e seus argumentos são 1 e

$$(CONS_{\alpha}\ 2(CONS_{\alpha}\ 3\ NIL_{(list\ int)})).$$

Observe ainda que a forma λ-normal indicada é representada pela lista

$$[1,2,3]$$

***

EXEMPLO 2.15

Este exemplo também considera a obtenção da forma λ-normal de uma fórmula. Seja

$$\Phi = \lambda\ X_{int}[\lambda\ Y_{int}(X_{int}+2+Y_{int})(4),\lambda\ Z_{int}[X_{int},Z_{int}]]\ 5$$

então a forma λ-normal de Φ e:

$[11,\lambda\ Z_{int}[5,Z_{int}]] =$

$$= (CONS_{\alpha}\ 11\ (CONS_{\alpha}\ \lambda\ Z_{int}[5,Z_{int}]\ NIL_{(listint)}))$$

Neste caso, não há variável de ligação e $CONS_{\alpha}$ é a cabeca. Já o número 11 e a fórmula

$$(CONS_{\alpha}\ \lambda\ Z_{int}[5,Z_{int}]\ NIL_{(list\ int)})$$

são argumentos. Observe que esta última fórmula também está na forma λ-normal.

***

CONVENCAO: Dada uma fórmula Φ, considera-se neste trabalho apenas a sua forma λ-normal. Assim, a referência a uma fórmula significa implicitamente a referência a sua forma λ-normal.

***

2.5 PROGRAMACAO LOGICA DE ORDEM SUPERIOR.

2.5.1 INTRODUCAO.

Nesta seção apresenta-se a progamação lógica de ordem superior. Esta programação e a representação hierárquica de conhecimento, a ser considerada neste trabalho, são aplicações do λ-cálculo tipado à representação de conhecimento. Isto é, são aplicações de lógica de ordem superior à representação de conhecimento. Inicia-se esta análise pela definição de predicado.

2.5.2 DEFINICAO. Predicado.

Uma fórmula Φ_α é um predicado se e somente se

$$\alpha = \langle \beta_1 \succ \beta_2 \succ \ldots \succ \beta_n \succ \iota \rangle,$$

onde $\iota \in \mathbb{F}$, "ι" é um símbolo para tipo associado aos elementos do conjunto de valores verdade (T,F).

***

O significado de um predicado com relação a uma designação φ é um elemento de $M_{\beta\iota}$. Isto é,

$$V[\Phi,\varphi] \in M_{\beta\iota}$$

Ou em outras palavras

$$V[\Phi,\varphi] : M_\beta \longrightarrow M_\iota$$

para qualquer designação φ.

Informalmente, predicados representam conjuntos e relacões. Um predicado do tipo $\langle \alpha \succ \iota \rangle$, por exemplo, representa um conjunto de objetos do tipo α. O exemplo a seguir esclarece este fato.

EXEMPLO 2.16

Considere o conjunto $\mathbb{N}$ dos números naturais e o predicado a seguir, que tem como domínio os elementos de $\mathbb{N}$.

$$(\lambda\ X\ (\text{par}\ X)\ N) \equiv \begin{cases} T & \text{se N e numero par} \\ F & \text{se N e numero impar} \end{cases}$$

Neste caso, o predicado (λX (par X)) representa o conjunto dos números pares. Já um predicado do tipo $(\beta \succ \iota)$ representa um conjunto de objetos do tipo β. Um predicado do tipo

$$((\beta \succ \iota) \succ \iota),$$

por exemplo, representa uma família de conjuntos de objetos do tipo β. Já um predicado do tipo

$$(\alpha \succ (\alpha \succ \iota) \succ \iota)$$

representa uma relação binária entre objetos do tipo α e conjuntos destes objetos.

***

São introduzidas a seguir uma série de definições que estão relacionadas de acordo com o diagrama apresentado na figura 2.1. Nesta figura as setas determinam as regras de formação dos conceitos envolvidos. Inicia-se com as definições de proposição e de fórmulas positivas. Utilizando tais definições obtém-se as definições de fórmula objetivo e de átomo. Em seguida, a partir da definição de átomo, tem-se a definição de átomo rígido. A definição de fórmula objetivo juntamente com a definição de átomo rígido determina a definição de cláusula definida. Finalmente, define-se programa lógico a partir da definição de cláusula objetivo. Observe que tais conceitos estão ligados na figura 2.1.

2.5.3 DEFINICAO. Proposicao.

Uma proposição é uma fórmula do tipo ι, $\iota \in \mathbb{F}$, "ι" é um símbolo para tipo associado aos elementos do conjunto de valores de verdade (T,F).

***

Como o símbolo para tipo associado a uma proposição é também um símbolo associado ao conjunto dos valores de verdade, cada proposição possui um valor de verdade. Tal fato confere às proposições um papel singular em programação lógica, na medida em que se procura provar a veracidade, ou não, de fórmulas. Isto é, quer se provar se uma fórmula é verdadeira ou não com relação a base de conhecimento do sistema, ou em outras palavras, se uma fórmula é, ou não, uma consequência lógica da base de conhecimento do sistema. Além disso, a obtenção de proposições se dá pela aplicação de predicados aos elementos de seu domínio. A partir do predicado (λX λY (boyle X Y)), por exemplo, obtém-se a proposição.

(boyle temperatura pressao)

que é forma normal de

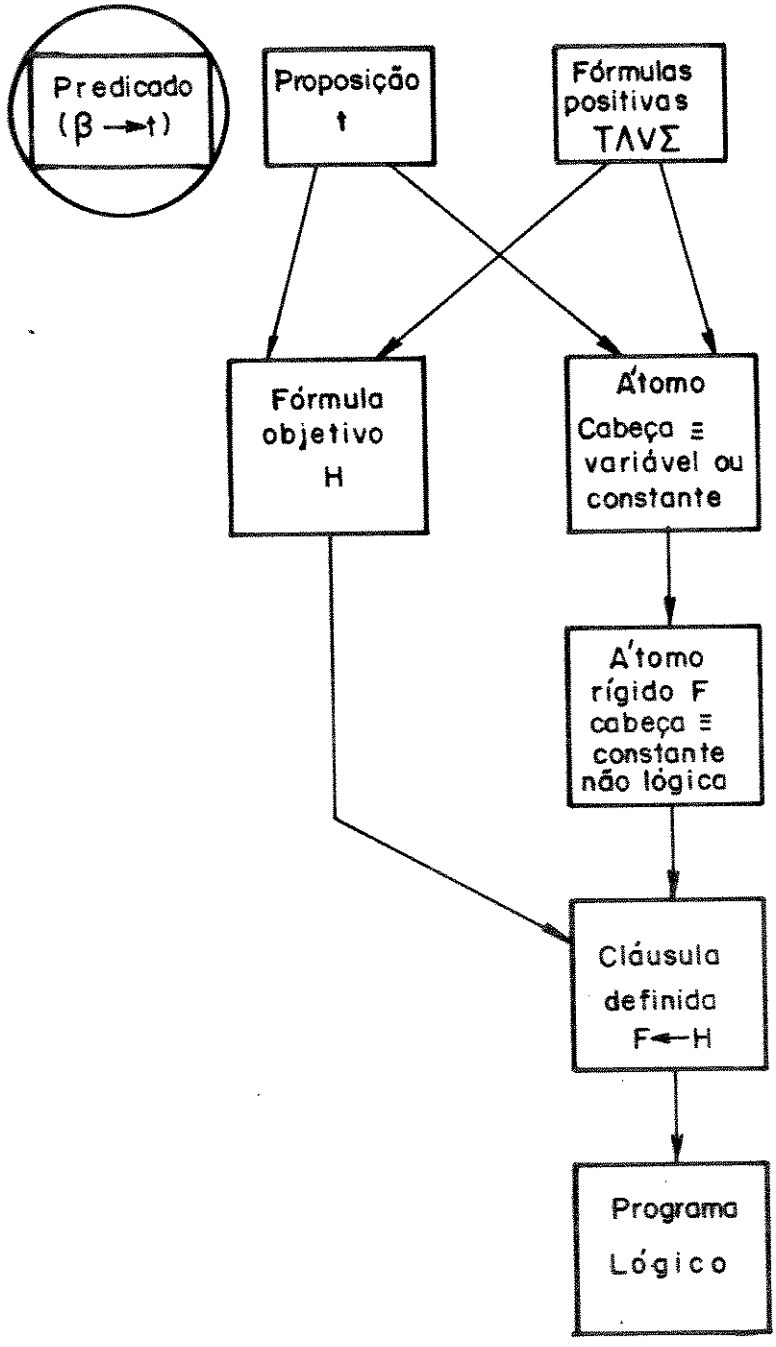
(λX λY (boyle X Y)) temperatura pressao)

Observe que o tipo do predicado (λX λY (boyle X Y)) é

(med ≻ med ≻ t)

e o tipo de (boyle temperatura pressao) é "t".

A partir dos conceitos de predicado e proposição indroduz-se a noção de cláusula, que é um componente dos programas lógicos. Entretanto, antes de definir cláusula de ordem superior são introduzidas algumas definições.



Conceitos fundamentais e
programacao logica de ordem superior.
FIGURA 2.1

2.5.4 DEFINICAO. Formulas positivas.

Uma fórmula F é positiva se ela é formada a partir de variáveis, de constantes não lógicas e das seguintes constantes lógicas:

$$T,\ \vee,\ \wedge,\ \Sigma_\alpha$$

***

Portanto, o conjunto $FORM^+$ das fórmulas positivas não contém as constantes lógicas a seguir:

$$\rightarrow,\ \neg\ \Pi_\alpha$$

e é formado como se segue:

(i) Se $X_\alpha \in VAR_\alpha$, então $X_\alpha \in FORM^+$.

(ii) Se $A_\alpha \in CONST_\alpha$, $A_\alpha \neq \neg$, então $A_\alpha \in FORM^+$.

(iii) Se $F,G,H \in FORM^+$, então as fórmulas

$$\lambda\ X\ F,\ GH,\ \Sigma_\alpha(\lambda\ X_\alpha F),\ F \wedge G,\ F \vee G$$

pertencem a $FORM^+$.

2.5.5 DEFINICAO. Formula objetivo.

Uma fórmula G_t é uma fórmula objetivo se $G_t \in FORM^+$ e G_t é uma proposição. Uma fórmula objetivo é uma proposição positiva.

***

2.5.6 DEFINICAO. Atomo.

Uma fórmula A_t, $t \in \mathbb{T}$, é um átomo quando ela satisfaz as seguintes condições:

(i) A_t é uma proposição.

(ii) $A_t \in FORM^+$.

(iii) A cabeca da forma normal de A_t é formada apenas de variáveis ou constantes não lógicas.

***

Os átomos, conforme esta definição, não são formados com as constantes lógicas

$$\Rightarrow, \neg, \Pi_\alpha$$

Um fato analogo ocorre em programação lógica de primeira ordem, conforme é analisado em [Lloyd, 1984]. Nesse caso não há átomos formados com a implicação, a negação e o quantificador universal. Entretanto, em programação lógica de ordem superior são definidos átomos como fórmulas que contém o quantificador existencial Σ_α. Isto difere do caso da primeira ordem, tornando a linguagem mais flexível para a representação de conhecimento. Os exemplos a seguir, neste capítulo, ilustram este fato.

2.5.7 DEFINICAO. Atomo rigido.

Um átomo é rígido se e somente se sua cabeca é uma constante.

***

EXEMPLO 2.17

Naturalmente, há várias maneiras de se obter um átomo rígido. Uma delas é considerando a aplicação de um predicado ao seu argumento. Considere, por exemplo, um predicado P_α,

$$\alpha = (\beta \succ \iota),$$

tal que

$$P_\alpha \in CONST_\alpha,$$

e uma fórmula normal C_β, então:

$$P_\alpha\ C_\beta$$

é um átomo rígido.

***

2.5.8 DEFINICAO. Fecho universal.

O fecho universal de uma fórmula Φ é a fórmula

$$\forall X_1 \forall X_2 \ldots \forall X_n \Phi$$

onde $\{X_1, X_2, \ldots, X_n\}$ é o conjunto das variáveis que ocorrem livres em Φ.

***

NOTACAO: O fecho de uma fórmula Φ é indicado por $\forall[X]\Phi$.

***

2.5.9 DEFINICAO. Clausulas definidas de ordem superior.

Uma cláusula definida de ordem superior é uma fórmula do tipo

$$\forall[X]\ G \rightarrow A$$

onde G é uma fórmula objetivo e A um átomo rígido.

***

NOTACAO: Quando não ocorrer dúvida na interpretação, o quantificador universal, que determina o fecho da cláusula, é omitido. Além disso, seguindo [Lloyd, 1984], utiliza-se a notação invertida para cláusula, escrevendo ":-" no lugar de "←" e a virgula "," no lugar do conectivo "∧". Tem-se a equivalência entre as fórmulas a seguir:

$$\forall[X]\ G \rightarrow A$$

$$A :- G$$

***

2.5.9.1 DEFINICAO. Cabeca e cauda de uma clausula.

Dado uma cláusula A :- G, então "A" é a sua cabeca e "G" a sua cauda.

***

2.5.10 DEFINICAO. Programa logico de ordem superior.

Um programa lógico de ordem superior é um conjunto de cláusulas de ordem superior.

***

Como os átomos e as fórmulas objetivo são fórmulas positivas, é possível formá-los utilizando o quantificador existencial. Assim, é possível ocorrer em uma cláusula a quantificação universal de quantificadores existenciais. A quantificação sobre quantificadores e a presenca de símbolos funcionais e predicados como variáveis distingue a programação lógica de ordem superior da programação lógica de primeira ordem. Os exemplos a seguir ilustram este fato.

EXEMPLO 2.18

Neste exemplo considera-se um conjunto de indivíduos relacionados por relações familiares como: "pai", "mãe", "esposa" etc. Escreve-se um programa lógico de ordem superior a partir do qual é possível determinar se dois indivíduos estão, ou não, relacionados por alguma relação familiar. Além disso, é possível também determinar o inverso, isto é, dados dois indivíduos, determinar se existe alguma relação familiar que os relaciona. Deve-se observar que a determinação de uma relação familiar que relaciona dois indivíduos somente é possível em programação lógica de ordem superior pois neste caso tem-se um símbolo predicado como variável.

Para simplificar a notação, no programa que se segue os símbolos para tipo associados às variáveis e constantes são omitidos. As constantes e as variáveis estão associadas a símbolos para tipo convenientes e as fórmulas são fórmulas bem formadas. Considere inicialmente a base de dados, que relaciona os indivíduos.

```
PAI BENEDITO SONIA
ESPOSA SONIA AGUINALDO
```

Cujo significado é o seguinte: **BENEDITO** é pai de **SONIA**, e **SONIA** é esposa de **AGUINALDO**. A seguir, considera-se o predicado

```
RELACAO-PRIMITIVA
```

que determina se um predicado é uma relação familiar primitiva. Isto é, se P e uma relação primitiva, entao

```
RELACAO-PRIMITIVA P
```

é uma proposição verdadeira. Os seguintes fatos indicam que **PAI** e **ESPOSA** são relações familiares primitivas.

```
RELACAO-PRIMITIVA  PAI
RELACAO-PRIMITIVA  ESPOSA
```

A partir do predicado **RELACAO-PRIMITIVA** define-se o predicado

```
RELACAO-FAMILIAR
```

Este predicado determina se um predicado é uma relação familiar. As cláusulas a seguir o definem. Observe que são utilizados os símbolos "(", ")", "[" e "]" no lugar de parênteses, para simplificar a leitura.

```
RELACAO-FAMILIAR  R  :- RELACAO-PRIMITIVA  R .

RELACAO-FAMILIAR ( λX λY ( ∃ Z (( S X Z ) Λ ( V Z Y ))) :-
                          :- RELACAO-PRIMITIVA  S ,
                             RELACAO-PRIMITIVA  V .
```

A primeira cláusula indica que se "R" é uma relação primitiva então é também uma relação familiar. Já a segunda cláusula diz que o predicado

$$\langle\ \lambda X\ \lambda Y\ [\ \exists\ Z\ \langle\langle\ S\ X\ Z\ \rangle\ \wedge\ \langle\ V\ Z\ Y\ \rangle\rangle$$

é uma relação familiar quando "S" e "V" são relações primitivas.

A partir do programa lógico apresentado, pode-se colocar questões como

:- PAI BENEDITO X

Neste caso, o programa se comporta como um programa lógico de primeira ordem, respondendo

X = SONIA

Considere agora a questão

(RELACAO-FAMILIAR R) ∧ (R BENEDITO AGUINALDO)

onde "**R**" é uma variável. Responder esta questão é determinar uma relação familiar "**R**", que relaciona os indivíduos **BENEDITO** e **AGUINALDO**. Observe inicialmente que no banco de fatos do programa não há nenhuma relação primitiva que relaciona tais indivíduos. A relação que se procura através deste objetivo é a relação familiar "sogro", pois **BENEDITO** é sogro de **AGUINALDO**. O programa lógico de ordem superior determina a resposta

R X Y ≡ λX λY [∃ Z ((PAI X Z) ∧ (ESPOSA Z Y))]

Observe que a questão indicada acima possui um símbolo predicado como variável. Deve-se enfatizar que isto so é possível em programação lógica de ordem superior. Em programação lógica de primeira ordem não é possível a presença de símbolos predicados ou funcionais como variáveis.

***

EXEMPLO 2.19

Neste exemplo apresenta-se um programa lógico de ordem superior, que representa um conhecimento sobre medidas de temperatura e pressão de um processo. Novamente, os símbolos para tipo associados às constantes e predicados do programa não são indicados, simplificando a notação. Inicia-se a apresentação do programa pela sua base de fatos.

TEMP C_1 20°

TEMP C_2 22°

TEMP C_3 22°

TEMP C_4 19°

PRES C_5 150

PRES C_6 70

O conhecimento representado por tais fatos indica que a temperatura no objeto C_1 é igual a 20°, em C_2 é igual a 22° e assim por diante. Tem-se também que a pressão em C_5 é igual a 150 e em C_6 é igual a 70. A seguir são definidas as cláusulas do programa, onde se considera a seguinte notação para listas:

(CONS X L) ≡ [X|L]

que é a mesma utilizada em prolog, [Turbo Prolog, 1986]. Considere inicialmente o predicado

TEM-PROPRIEDADE.

Dado um predicado P e duas listas L e K, então

TEM-PROPRIEDADE P L K

é verdadeiro quando K é uma lista formada com os elementos de L, que possuem a propriedade determinada pelo predicado P. Em outras palavras, se P é tal que

$$P\ X = \begin{cases} T \text{ se } X \text{ é um numero par.} \\ F \text{ se } X \text{ é um numero impar.} \end{cases}$$

então a proposição a seguir é verdadeira.

TEM-PROPRIEDADE P [2,3,4,5,6] [2,4,6]

As cláusulas a seguir definem este predicado.

```
TEM-PROPRIEDADE P NIL NIL.
TEM-PROPRIEDADE P [X|L] [X|K] :-
                    :- PX, TEM-PROPRIEDADE P L K.
TEM-PROPRIEDADE P [X|L] K :- TEM-PROPRIEDADE P L K.
```

Utilizando-se o predicado **TEM-PROPRIEDADE**, definido acima, são analisadas a seguir mais duas cláusulas.

(i) clausula C_1.

```
TEM-TEMPERATURA L K :-
        :- TEM-PROPRIEDADE (λ Z (Σα (λ X(TEMP Z X)))) L K
```

Esta cláusula define o predicado **TEM-TEMPERATURA**, que é satisfeito quando **K** é uma sublista de **L** e associados aos seus elementos há uma medida de temperatura. Isto é, se

$$K = \{\ X \in L \text{ tal que } \exists\ T,\ \text{TEMP } X\ T\ \}$$

Observe que o predicado

$$(\lambda\ Z\ (\Sigma_{\alpha}\ (\lambda\ X(\text{TEMP } Z\ X))))$$

que faz parte da primeira cláusula, também é escrito como,

$$(\lambda\ Z\ (\exists\ X(\text{TEMP } Z\ X))))$$

Considere que a seguinte fórmula objetivo seja apresentada ao sistema.

$$:-\ \text{TEM-TEMPERATURA } \{C_1,C_2,C_3,C_5,C_6\}\ X$$

Neste caso a resposta é

$$X = \{C_1,C_2,C_3\}$$

(ii) clausula C_2.

TEM-A-MESMA-TEMPERATURA A L K :-
:-TEM-PROPRIEDADE (λ Z (TEMP Z A)) L K

o predicado,

TEM-A-MESMA-TEMPERATURA A L K

é satisfeito quando **K** é uma sublista de **L** tal que,

L = { X ∈ K tal que a proposicao (TEMP X A) e verdadeira}

Isto é, a temperatura em **X** é **A**. Considere agora a fórmula objetivo:

$$:-\ \text{TEM-A-MESMA-TEMPERATURA } 22^{\circ}\ \{C_1,C_2,C_3,C_5,C_6\}\ X$$

cuja resposta é $X = \{C_2,C_3\}$.

Programação lógica de ordem superior permite a utilização de símbolos predicados como variáveis, o que não ocorre em programação lógica de primeira ordem. Por exemplo, suponha que se deseja determinar se existe um predicado que relaciona o objeto C_3 à temperatura 22°, isto é, existe **P** tal que a fórmula

$$P\ C_3\ 22^{\circ}$$

é satisfeita. Um valor possível para **P** é:

$P = \lambda X \lambda Y (TEMP\ X\ Y)$

Há ainda outras substituições possíveis para P tais como,

$P = \lambda X \lambda Y (X = C_8, Y = 22^\circ)$

$P = \lambda X \lambda Y (Y = 22^\circ)$

$P = \lambda X \lambda Y (TEMP\ C_1\ 20^\circ)$

$P = \lambda X \lambda Y T$

Existem várias substituições possíveis para o predicado P e a aplicação deve definir a substituição a ser adotada. Este problema não é considerado neste trabalho, sendo tema de futuras pesquisas na área.

***

2.6 CONCLUSAO.

Neste capítulo foram introduzidas as noções fundamentais do λ-cálculo tipado e de programação lógica de ordem superior visando a definição formal de uma programação funcional a ser apresentada no capítulo seguinte.

A utilização de símbolos para tipo associados às constantes e variáveis da linguagem do sistema determinam uma hierarquização dos objetos formados a partir de um conjunto D, como indicado na definição de "Relação tipada baseada em um conjunto D". Como será apresentado, no estudo da representação hierárquica de conhecimento, os símbolos para tipo também permitirão determinar hierarquias entre as fórmulas do λ-cálculo tipado. O estudo destes símbolos, juntamente com a descrição sintática das fórmulas do λ-cálculo tipado, também estabelecem condições necessárias para a derivação de conhecimento em sistemas que utilizam as fórmulas do λ-cálculo como elemento básico de representação. Finalmente, baseado nas hierarquias entre as fórmulas do λ-cálculo tipado, será possível definir a arquitetura de um sistema especialista no qual o conhecimento é representado utilizando-se fórmulas do sistema lógico analisado neste capítulo.

CAPITULO 3

λ-PROGRAMAS COM CRITERIO.

Neste capitulo propoe-se a conceituacao formal de programa com criterio. Este conceito e definido como um conjunto de formulas do λ-calculo tipado associado a um mecanismo de derivacao. Este mecanismo determina o sequenciamento da execucao das funcoes representadas pelas formulas do λ-calculo tipado. O programa com criterio e um programa funcional onde os simbolos para tipo associados aos elementos da linguagem sao utilizados explicitamente na representacao do conhecimento. A programacao introduzida e ilustrada pela apresentacao de λ-programas com criterio que simulam redes neurais e redes de Petri.

3.1 INTRODUCAO.

Neste capítulo é proposto o conceito formal de programa com critério e em seguida é analisada a derivação de conhecimento a partir destes programas. Um programa com critério é uma lista de fórmulas do λ-cálculo tipado associada a um mecanismo de derivação de conhecimento. Como cada fórmula do λ-cálculo tipado representa uma função, um λ-programa é um conjunto de funções, isto é, um programa funcional [Henderson, 1980]. Técnicas de programação recursiva [Burge, 1975] e linguagens de programação funcional como LISP [Winston, 1981], SCHEME [Friedman, 1987] e A* [Meira, 1988], amplamente utilizadas em inteligência artifical, também são definidas a partir dos conceitos fundamentais de λ-cálculo. Entretanto, elas não explicitam a utilização dos símbolos para tipos como elementos para a representação e derivação de conhecimento, no sentido que é proposto neste trabalho. Nas linguagens funcionais usuais, indicadas acima, um programa é uma função definida como a composição de outras funções. Por outro lado, um programa com critério é um conjunto de funções associadas a um critério de derivação. Este critério define como o conhecimento é derivado a partir do conjunto de funções. Enquanto em linguagens funcionais usuais, o mecanismo de derivação fica implícito na composição da função que define o programa, em um programa com critério, o mecanismo é explicitado pelo critério de derivação. O objetivo deste capítulo é estabelecer apenas os fundamentos formais de uma

linguagem funcional baseada no λ-cálculo tipado, não se considerando a sintaxe e detalhes de implementação da linguagem propriamente dita.

Inicialmente, define-se os conceitos de λ-programa, lista compatível, substituição resposta, aplicação e composição de substituições respostas, derivação em uma fórmula e derivação com substituição resposta.

Em seguida, na seção 3.3, é proposta uma representação gráfica para as fórmulas do λ-cálculo tipado.

Na seção 3.4 é proposto o conceito de derivação de conhecimento a partir de um λ-programa. Considera-se duas formas de derivação de conhecimento a partir de um programa com critério. Uma derivação "forward" e uma derivação "backward", ou por retrocesso. A derivação "forward" é analisada nesta seção. Da seção 3.4.1 à seção 3.4.8 são definidos, respectivamente, os conceitos de critério de derivação compatível com uma fórmula, critério de derivação global, λ-programa com critério, sequência de derivação com memória, sequência de derivação sem memória, derivação com substituição resposta e dedução por derivação.

Em seguida, na seção 3.5, os conceitos propostos nas seções anteriores são ilustrados. É proposto um programa com critério que simula uma rede neural.

Na seção 3.6 é proposta a derivação de conhecimento "backward", ou por retrocesso. Inicia-se definindo, nas seções 3.6.1 e 3.6.2, respectivamente, os conceitos de passo de derivação por retrocesso e sequência de derivação por retrocesso. Em seguida, a derivação de conhecimento "backward", ou por retrocesso, é ilustrada através de exemplos.

3.2 DEFINICOES BASICAS.

Apresenta-se, a seguir, um conjunto de definições básicas, utilizadas para introduzir o conceito de programa com critério. Estas definições são baseadas nos fundamentos do λ-cálculo tipado.

3.2.1. DEFINICAO. λ-Programa.

Um λ-programa é uma lista de fórmulas do λ-cálculo tipado, indicada por λ-*P*.

***

Um programa lógico de ordem superior é um λ-programa. Neste caso o ordenamento da lista é definido pela ordem em que as fórmulas aparecem no programa.

3.2.2 DEFINICAO. Lista compativel.

Seja

$$\Phi_\beta = \lambda X_1 \ldots \lambda X_n \ (A\ F_1 \ldots F_m)$$

uma fórmula do λ-cálculo tipado, isto é, $\Phi_\beta \in FORM_\beta$. Uma lista finita de fórmulas $\mathbb{E}$

$$\mathbb{E} = [A_1 \ldots A_s]$$

é compatível com Φ se

(i) $s \leq n$ e

(ii)

$$tipo[\ A_i\ ] = tipo[\ X_i\ ]$$

onde $1 \leq i \leq s$.

***

Dada uma fórmula sem variáveis de ligação

$$\Phi = (A\ F_1 \ldots F_m)$$

a única lista compatível com Φ é a lista vazia,

$$\mathbb{E} = NIL$$

3.2.3 DEFINICAO. Substituicao resposta.

Uma substituição resposta é um conjunto,

$$\theta = \{\langle X_1, F_1\rangle, \ldots, \langle X_n, F_n\rangle\}$$

tal que se $i \neq j$ então

$$X_i \in VAR,$$

$$F_i \in FORM,$$

$$X_i \neq F_i,$$

$$X_i \neq X_j$$

e

$$tipo[X_i] = tipo[F_i],\ 1 \leq i \leq n$$

Além disso, para todo i,j tal que $1 \leq i \leq n$, $1 \leq j \leq n$, X_j não ocorre livre em F_i.

***

3.2.4 DEFINICAO. Aplicacao e composicao de substituicoes respostas.

A aplicação da substituição resposta θ a uma fórmula ξ, indicada por $\xi\theta$, é definida a seguir

$$\xi\theta = (\lambda X_1 \ldots \lambda X_n \xi)\ F_1 \ldots F_n$$

A composição de duas substituições respostas

$$\theta_1 = (\langle x_1, F_1\rangle, \ldots, \langle x_n, F_n\rangle)$$

$$\theta_2 = (\langle y_1, G_1\rangle, \ldots, \langle y_m, G_m\rangle)$$

é definida por:

$$\theta = \theta_1\theta_2 = (\langle x_1, F_1\theta_2\rangle, \ldots, \langle x_n, F_n\theta_2\rangle, \langle y_1, G_1\rangle, \ldots, \langle y_m, G_m\rangle)$$

de onde se retiram os pares

$$\langle x_i, F_i\theta_2\rangle$$

tais que

$$x_i = F_i\theta_2$$

e os pares

$$\langle y_j, G_j\rangle$$

tais que

$$y_j \in (x_1, \ldots, x_n).$$

***

Observe que θ, definida como a composição das substituições θ_1 e θ_2, é uma substituição.

3.2.5 DEFINICAO. Derivacao em uma formula.

Sejam Φ uma fórmula e

$$\mathbb{E} = [A_1, \ldots, A_s]$$

uma lista de fórmulas compatível com Φ. Então a derivação em Φ conforme $\mathbb{E}$ é dada por Ψ, onde

$$\Psi = \Phi\ A_1 \ldots A_s$$

***

3.2.6 DEFINICAO. Derivacao com substituicao resposta.

Se existe uma substituição resposta

$$\theta = (\langle x_1, F_1\rangle, \ldots, \langle x_n, F_n\rangle),$$

tal que

$$\Psi = \zeta\theta$$

então diz-se que ζ é uma derivação em Φ com substituição resposta θ.

***

3.3 REPRESENTACAO GRAFICA DE UMA FORMULA.

A partir das definições da seção anterior é proposta uma representação gráfica para as fórmulas do λ-cálculo tipado. Considere inicialmente a definição.

3.3.1 DEFINICAO. Formula neutra.

Uma fórmula Φ,

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (A\ F_1 \ldots F_m)$$

é uma fórmula neutra se

$$n = \emptyset$$

Isto é, Φ não começa com o símbolo "λ", ou seja, Φ não possui variáveis de ligação.

***

Considere agora uma fórmula Φ,

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (A\ F_1 \ldots F_m)$$

e uma lista compatível com Φ, dada por,

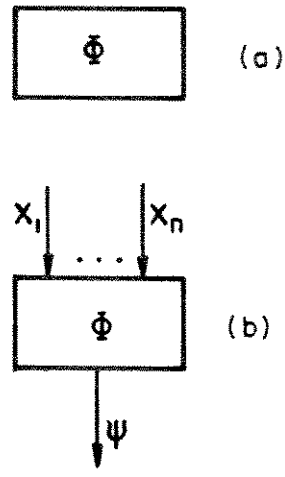
$$\mathbb{E} = [B_1, \ldots, B_s]$$

Propõe-se uma representação gráfica para Φ, como se segue,

(i) Se n=$\emptyset$, isto é, Φ é uma fórmula neutra, então Φ é representado conforme a figura 3.1(a).

(ii) Se n>$\emptyset$, então Φ é representado conforme a figura 3.1(b) onde Ψ é a derivação em Φ conforme $\mathbb{E}$.

***



Representacao grafica de uma formula.
FIGURA 3.1

3.4 DERIVACAO DE CONHECIMENTOS A PARTIR DE UM λ-PROGRAMA.

A derivação de conhecimento a partir de um λ-programa λ-*P* é obtida quando se associa um critério de derivação à lista de fórmulas de λ-*P*. Este critério representa um conhecimento que determina como tais funções são executadas. Ele determina a sequência de execução das funções, determinando os seus argumentos, ou entradas. O critério de derivação associado a um λ-programa determina uma lista compatível para cada função do λ-programa. Neste sentido, dado um programa com um critério de derivação associado a cada fórmula, é possível deduzir conhecimento a partir dele.

A seguir propõe-se a definição de critério de derivação e a derivação de conhecimento a partir de um λ-programa, seguindo um método "forward".

Um critério de derivação é uma função

$$\mathbb{D} : \mathfrak{L}\left[\bigcup_{\alpha \in \mathbb{T}} FORM_\alpha \right] \dashrightarrow \mathfrak{L}\left[\bigcup_{\alpha \in \mathbb{T}} FORM_\alpha \right]$$

onde $\mathfrak{L}[\,A\,]$ é o conjunto das listas constituídas pelos elementos de A.

3.4.1 DEFINICAO. Criterio de derivacao compativel com uma formula.

Sejam λ-P um programa, Φ uma fórmula de λ-P e $\mathbb{D}$ um critério de derivação. Então $\mathbb{D}$ é compatível com Φ em λ-P se e somente se

$$\mathbb{D}[\ \lambda\text{-}P\ ]$$

é compatível com Φ.

***

EXEMPLO 3.1.

Apresenta-se neste exemplo a definição de critérios de derivação compatíveis e incompatíveis com uma fórmula dada. Considere os dois casos a seguir.

(a) Seja $\Phi \in$ FORM onde

$$\Phi = \lambda X_1 \ldots \lambda X_n\ (AF_1 \ldots F_m)$$

e λ-P um λ-programa tal que $\Phi \in \lambda$-P. Considere $\mathbb{D}$, tal que

$$\mathbb{D}[\ \lambda\text{-}P\ ] = [X_1, \ldots, X_n],$$

$\mathbb{D}$ é um critério de derivação compatível com Φ em λ-P.

(b) Considere agora, $\Psi \in \lambda$-P onde

$$\Psi = \lambda Y_{int}\ (Y_{int} + 2_{int})$$

Seja o seguinte critério de derivação

$$\mathbb{D}[\ \lambda\text{-}P\ ] = [4_{int}]$$

Como

$$tipo[4_{int}] = tipo[Y_{int}],$$

então $\mathbb{D}$ é um critério compatível com Ψ em λ-P. Considere agora o seguinte critério de derivação

$$\mathbb{D}[\ \lambda\text{-}P\ ] = [string_{char}]$$

Como

$$tipo[string_{char}] \neq tipo[Y_{int}],$$

então $\mathbb{D}$ não é um critério compatível com Ψ em λ-P.

***

Dada uma fórmula $\Phi \in \lambda\text{-}P$ e um critério de derivação $\mathbb{D}$, então $\mathbb{D}$ é compatível com Φ em $\lambda\text{-}P$ se o resultado da aplicação de $\mathbb{D}$ em $\lambda\text{-}P$, $\mathbb{D}[\lambda\text{-}P]$, é uma lista de fórmulas compatível com Φ. Observe que na definição de tal critério, o λ-programa $\lambda\text{-}P$ é fixo e $\Phi \in \lambda\text{-}P$. A seguir, esta noção é ampliada considerando-se um $\lambda\text{-}P$ qualquer. Define-se critério de derivação global, que estabelece listas compatíveis para cada fórmula de um programa.

3.4.2 DEFINICAO. Criterio de derivacao global.

Seja $\lambda\text{-}P$ um programa, uma função $\mathbb{D}$

$$\mathbb{D} : \left[\bigcup_{\alpha \in \mathbb{T}} FORM_\alpha \right] \times \mathfrak{L}\left[\bigcup_{\alpha \in \mathbb{T}} FORM_\alpha \right] \dashrightarrow \mathfrak{L}\left[\bigcup_{\alpha \in \mathbb{T}} FORM_\alpha \right]$$

é um critério de derivação global se para toda fórmula Φ e todo λ-programa $\lambda\text{-}P$ onde $\Phi \in \lambda\text{-}P$, então

$$\mathbb{D}[\Phi, \lambda\text{-}P]$$

é uma lista de fórmulas compatível com Φ.

***

NOTACAO. No que se segue, o termo "critério de derivação global" é referido apenas como "critério de derivacao".

***

Dado um λ-programa $\lambda\text{-}P$, um critério de derivação determina listas de fórmulas compatíveis com cada fórmula Φ de $\lambda\text{-}P$. As fórmulas pertencentes a estas listas são utilizadas, como argumentos, na execução das fórmulas do λ-programa $\lambda\text{-}P$. Portanto, o critério $\mathbb{D}$ determina os argumentos de cada fórmula do λ-programa $\lambda\text{-}P$, possibilitando sua execução e a derivação de um novo conhecimento. Ele permite obter conhecimento a partir de um λ-programa. Os programas considerados neste trabalhos são listas de fórmulas do λ-cálculo tipado associados a critérios de derivação, que possibilitam a derivação de um novo conhecimento. Estes conceitos são definidos formalmente a seguir.

3.4.3 DEFINICAO. λ-Programa com criterio.

Sejam $\lambda\text{-}P$ um λ-programa e $\mathbb{D}$ um critério de derivação. Um programa com critério é definido pelo par ordenado

$$\langle \lambda\text{-}P, \mathbb{D} \rangle$$

***

Analisa-se a seguir a derivação de conhecimento a partir de um programa com critério, considerando-se dois aspectos distintos:

(a) Derivação de conhecimento com a memorização dos passos intermediários.

(b) Derivação de conhecimento sem a memorização dos passos intermediários.

No primeiro caso, o conhecimento intermediário obtido pelo sistema é memorizado. Tem-se a história de todo o conhecimento processado pelo sistema. Já no segundo caso, o conhecimento intermediário não é armazenado. Tem-se apenas a representação inicial e final do conhecimento. A derivação com memorização dos passos intermediários é definida a seguir, utilizando-se a seguinte notação.

NOTACAO. Dadas duas listas $[A_1,\ldots,A_N]$ e $\lambda\text{-}P$ então o resultado da concatenação destas listas é indicado por

$$[A_1,\ldots,A_N,\ |\ \lambda\text{-}P]$$

isto é,

$$[A_1,\ldots,A_N,\ |\ \lambda\text{-}P] = \text{CONS}\ [A_1,\ldots,A_N]\ \ \lambda\text{-}P$$

onde CONS é o predicado usualmente utilizado em LISP.

***

3.4.4 DEFINICAO. Sequencia de derivacao com memoria.

Seja $\langle\lambda\text{-}P,\mathbb{D}\rangle$ um programa com critério. Uma sequência de derivação **COM** memória a partir de $\langle\lambda\text{-}P,\mathbb{D}\rangle$ é dada por:

$$\langle\lambda\text{-}P_0,\mathbb{D}\rangle,\langle\lambda\text{-}P_1,\mathbb{D}\rangle,\ldots,\langle\lambda\text{-}P_n,\mathbb{D}\rangle,\ldots$$

onde

$$\langle\lambda\text{-}P_0,\mathbb{D}\rangle = \langle\lambda\text{-}P,\mathbb{D}\rangle$$

e se

$$\lambda\text{-}P_i = [\Phi_1,\ldots,\Phi_m],$$

então

$$\lambda\text{-}P_{i+1} = [\Psi_1,\ldots,\Psi_m,\Phi_1,\ldots,\Phi_m]$$

isto é,

$$\lambda\text{-}P_{i+1} = [\Psi_1,\ldots,\Psi_m\ |\ \lambda\text{-}P_i]$$

com Ψ_j uma derivação em Φ_j conforme

$$\mathbb{D}\left[\Phi_j,\ \lambda\text{-}P_i\right]$$

***

3.4.5 DEFINICAO. Sequencia de derivacao sem memoria.

Seja $\langle \lambda\text{-}P, \mathbb{D} \rangle$ um programa com critério. Uma seqüência de derivação SEM memória a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ é dada por:

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

onde

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle = \langle \lambda\text{-}P, \mathbb{D} \rangle$$

e se

$$\lambda\text{-}P_0 = [\Phi_1, \ldots, \Phi_m],$$

então

$$\lambda\text{-}P_{i+1} = [\Psi_1, \ldots, \Psi_m, \Phi_1, \ldots, \Phi_m]$$

isto é,

$$\lambda\text{-}P_{i+1} = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_0]$$

onde Ψ_j é uma derivação em Φ_j conforme

$$\mathbb{D}\left[\Phi_j, \lambda\text{-}P_i\right]$$

***

Observe que a derivação sem memória não considera os resultados intermediários pois,

$$\lambda\text{-}P_i = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_0]$$

O λ-programa $\lambda\text{-}P_i$ é obtido considerando apenas as fórmulas da lista inicial $\lambda\text{-}P_0$ e as últimas derivações $[\Psi_1, \ldots, \Psi_m]$. Na derivação com memória,

$$\lambda\text{-}P_{i+1} = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_i]$$

O λ-programa $\lambda\text{-}P_{i+1}$ é obtido a partir das últimas derivações $[\Psi_1, \ldots, \Psi_m]$ e do último λ-programa $\lambda\text{-}P_i$ da seqüência de derivação.

Nas duas definições anteriores tem-se que

$$\lambda\text{-}P_i = [\Psi_1, \ldots, \Psi_m, \Phi_1, \ldots, \Phi_m]$$

onde Ψ_j é uma derivação em Φ_j conforme

$$\mathbb{D}\left[\Phi_j, \lambda\text{-}P_i\right]$$

3.4.6 DEFINICAO. Sequencia de derivacao neutra.

Uma sequência de derivação

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

é neutra se para cada $\Phi_j \in \lambda\text{-}P_i$ tem-se que

$$\mathbb{D} \left[\Phi_j, \lambda\text{-}P_i \right]$$

é uma lista de fórmulas neutras.

***

3.4.7 DEFINICAO. Derivacao com substituicao resposta.

Se nas definições anteriores Ψ_j é uma derivação em Φ_j com resposta θ_i, então a derivação

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

é uma derivação com substituição resposta

$$\theta = \theta_0 \theta_1 \ldots \theta_n \ldots$$

***

3.4.8 DEFINICAO. Deducao por derivacao.

Sejam Φ uma fórmula e $\langle \lambda\text{-}P, \mathbb{D} \rangle$ um programa com critério. Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ se

$$\Phi \in \lambda\text{-}P_i$$

para algum "i", onde $\langle \lambda\text{-}P_i, \mathbb{D} \rangle$ pertence à sequência de derivação a seguir.

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

***

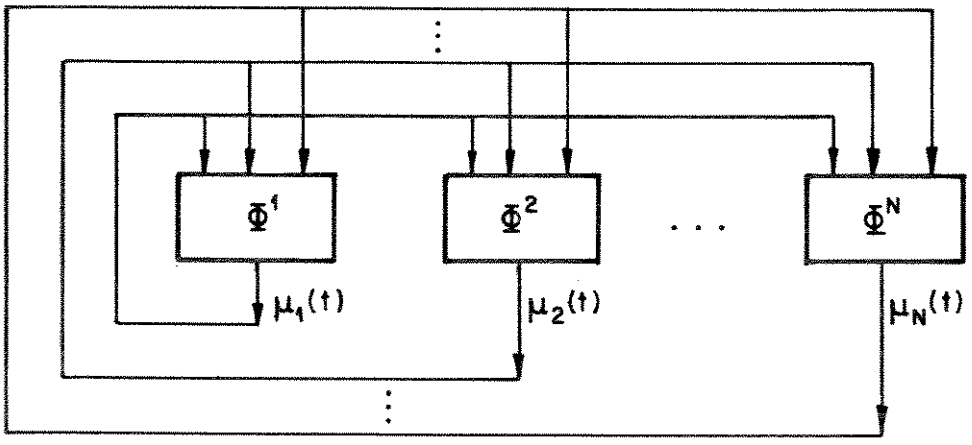
3.5 REPRESENTACAO DE REDES NEURAIS

Ilustra-se a seguir os conceitos definidos nas seções anteriores pela elaboração de um programa com critério que representa uma rede neural. A rede neural considerada foi

proposta por Hopfield em [Lippmann, 1987] e é analisada a seguir.

3.5.1 REDE NEURAL DE HOPFIELD.

Apresenta-se a seguir os conceitos básicos de redes neurais através de uma rede proposta por Hopfield [Lippmann, 1987]. Um neurônio é uma função que possui várias entradas ou argumentos e uma única saída. Esta saída é o resultado da aplicação da função aos seus argumentos. A rede neural é constituída pela ligação das saídas dos diferentes neurônios às suas entradas. Uma rede neural é uma rede em que cada nó é constituído por um neurônio, cuja saída é ligada a outros nós. Observe que a saída de um neurônio pode estar ligada a sua entrada, havendo uma realimentação. Apresenta-se na figura 3.2 a rede neural de Hopfield [Lippmann, 1987].



Rede neural de Hopfield.
FIGURA 3.2

Dado um conjunto de M vetores n-dimensionais

$$X^1 = (X_1^1, \ldots, X_N^1)$$

$$X^2 = (X_1^2, \ldots, X_N^2)$$

$$X^3 = (X_1^3, \ldots, X_N^3)$$

...

$$X^M = (X_1^M, \ldots, X_N^M)$$

e um outro vetor arbitrário também n-dimensional

$$Y = (Y_1, \ldots, Y_N)$$

a rede proposta por Hopfield identifica qual dos vetores X^i $1 \leq i \leq M$ mais se assemelha ao vetor Y. Isto é obtido fazendo

$$\mu_i(0) = Y_i$$

onde $\mu_i(0)$ é a saída do i-ésimo neurônio no ciclo zero. Já a saída do i-ésimo neurônio no ciclo t é dada por

$$\mu_i(t) = \Phi_i\ \mu_1(t-1) \ldots \mu_N(t-1)$$

onde Φ_i é a função do λ-cálculo tipado que representa o i-ésimo neurônio. No limite, quando a rede converge, o vetor

$$(\mu_i(t), \ldots, \mu_N(t))$$

se aproxima do vetor X^i para algum "i", isto é,

$$X^i = \lim_{t \to \infty} (\mu_i(t), \ldots, \mu_N(t))$$

Observe que esta rede é contituída por N neurônios, indicados por $\Phi_1, \ldots, \Phi_N$, e que são representados pelas funções

$$\Phi^j = \lambda X_1 \ldots \lambda X_N (f [t_{1j} * X_1 + \ldots + t_{Nj} * X_j])$$

onde $1 \leq j \leq N$ e a função f é dada por

$$f(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ -1 & \text{se } x < 0 \end{cases}$$

Os operadores "+" e "*" representam a adição e o produto de inteiros, respectivamente. Os pesos t_{ij} são números inteiros definidos a partir dos padrões básicos $X^1, \ldots, X^M$ a serem reconhecidos pela rede. t_{ij} é o peso da ligação entre a saída do i-ésimo neurônio e uma entrada do j-ésimo neurônio. Este peso é definido a seguir [Lippmann, 1987].

$$t_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s \, x_j^s & \text{se } i \neq j \\ 0 & \text{se } i = j \end{cases}$$

3.5.2 REPRESENTACAO DA REDE NEURAL DE HOPFIELD POR UM PROGRAMA COM CRITERIO.

Define-se a seguir um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$, que representa a rede neural indicada acima. Como foi visto nas seções anteriores, um programa com critério é definido por uma lista de fórmulas do λ-cálculo tipado associada a um critério de derivação. Na representação proposta neste trabalho, as fórmulas do λ-cálculo tipado são as funções que representam os neurônios da rede. Isto é, o λ-programa $\lambda\text{-}P$ é definido por,

$$\lambda\text{-}P = [\Phi_1, \ldots, \Phi_N]$$

onde Φ_i é a função que representa o i-ésimo neurônio.

A execução das fórmulas de $\lambda\text{-}P$ só é possível pela associação a $\lambda\text{-}P$ de um critério de derivação. No caso da rede neural de Hopfield este critério representa as ligações entre os neurônios da rede, o que possibilita a interação da rede. A definição formal do critério é dada a seguir,

$$\mathbb{D}\,[\Phi, [\Psi_1, \ldots, \Psi_p]] = \begin{cases} [\Psi_1, \ldots, \Psi_N] & \text{se } p \geq N \\ [\Psi_1, \ldots, \Psi_p, X_{p+1}, \ldots, X_N] & \text{se } p < N \\ \text{onde } tipo[X_i] = int, \; p+1 \leq i \leq N. \end{cases}$$

para toda fórmula Φ.

Isto é, se o λ-programa contém mais de N fórmulas, o critério seleciona as N primeiras. Caso contrário, o critério adiciona variáveis que estão associadas ao símbolo para tipo int. Observe ainda que este critério não depende de Φ e considera explicitamente a ordenação entre as fórmulas do λ-programa.

Considera-se a seguir as sequências de derivação com memória e sem memória. Nestas sequências, cada passo repesenta uma interação ou ciclo diferente na rede neural. O conhecimento gerado pela rede é representado nas sequências de derivação. Considere

$$Y = (A_1, \ldots, A_N)$$

o padrão que se deseja identificar. Considere também,

$$\lambda\text{-}P_1 = [A_1, \ldots, A_N, \mid \lambda\text{-}P]$$

Uma seqüência de derivação sem memória é dada por

$$\langle \lambda\text{-}P_1, \mathbb{D} \rangle, \langle \lambda\text{-}P_2, \mathbb{D} \rangle \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

onde

$$\lambda\text{-}P_i = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_1], \quad i \geq 1,$$

e Ψ_j é a derivação em Φ_j conforme $\mathbb{D}[\Phi_j, \lambda\text{-}P]$. Observe que nesta derivação os λ-programas $\lambda\text{-}P_i$ consideram apenas as fórmulas que representam a rede neural, o padrão inicial e as N saídas, dos N neurônios, a cada ciclo. Esta derivação não memoriza as saídas intermediárias dos neurônios. Quando a rede converge, tem-se que

$$X^i = \lim_{t \to \infty} (\Psi_1, \ldots, \Psi_N)$$

para algum "i", $1 \leq i \leq N$, onde "t" é o número de ciclos. Por outro lado, uma seqüência de derivação com memória é dada por

$$\lambda\text{-}P_{i+1} = [\Psi_1, \ldots, \Psi_N \mid \lambda\text{-}P_i], \quad i \geq 1,$$

na qual os resultados intermediários são memorizados. Se a rede converge, então

$$X^i = \lim_{t \to \infty} (\Psi_1, \ldots, \Psi_N)$$

para algum "i", $1 \leq i \leq N$. Observe que na seqüência de derivação com memória as saídas dos neurônios da rede são adicionadas aos λ-programas $\lambda\text{-}P_i$, a cada ciclo.

No apêndice A tem-se a ilustração da aplicação de programas com critério à representação de uma rede de Petri [Peterson, 1981].

3.6 DERIVACAO POR RETROCESSO.

A derivação de conhecimento por retrocesso é característica do método utilizado em programação lógica para estabelecer conseqüências lógicas [Lloyd, 1984]. Em programação lógica, dado um programa P e uma fórmula Φ, procura-se estabelecer se Φ é uma conseqüência lógica de P. Utiliza-se então a resolução, que é um método "backward" [Kowalski, 1979], para estabelecer a conseqüência lógica da fórmula Φ a partir de P. Neste trabalho propõe-se um método de derivação por retrocesso análogo à resolução utilizada em programação lógica de ordem superior. Na derivação de conhecimento por retrocesso, ou "backward", tem-se os passos

(i) dado um λ-programa λ-P e uma fórmula Φ, procura-se

(ii) determinar se existe algum critério $\mathbb{D}$ tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-P}, \mathbb{D} \rangle$.

Na análise que se segue, define-se inicialmente o passo de derivação por retrocesso. Em seguida é considerada a sequência derivação por retrocesso, que é uma composição de vários passos de derivação por retrocesso. Esta derivação é ilustrada através de um λ-programa equivalente ao programa lógico de ordem superior apresentado no exemplo 2.18. Nesta ilustração é construída uma sequência de derivação por retrocesso a partir de uma fórmula Φ dada. A definição de passo de derivação por retrocesso, dada a seguir, baseia-se na definição de P-derivação encontrada em [Nadathur, 1987].

3.6.1 DEFINICAO. Passo de derivacao por retrocesso.

Sejam λ-P um programa, θ_1 uma substituição resposta e OB_1 um conjunto de fórmulas, denominado conjunto objetivo. Um passo de derivação por retrocesso a partir de

$$\langle \lambda\text{-P}, \theta_1, OB_1 \rangle$$

é apresentado por

$$\langle \lambda\text{-P}, \theta_2, OB_2 \rangle$$

onde θ_2 e OB_2 são obtidos como descrito a seguir.

3.6 DERIVACAO POR RETROCESSO.

A derivação de conhecimento por retrocesso é característica do método utilizado em programação lógica para estabelecer conseqüências lógicas [Lloyd, 1984]. Em programação lógica, dado um programa P e uma fórmula Φ, procura-se estabelecer se Φ é uma conseqüência lógica de P. Utiliza-se então a resolução, que é um método "backward" [Kowalski, 1979], para estabelecer a conseqüência lógica da fórmula Φ a partir de P. Neste trabalho propõe-se um método de derivação por retrocesso análogo à resolução utilizada em programação lógica de ordem superior. Na derivação de conhecimento por retrocesso, ou "backward", tem-se os passos

(i) dado um λ-programa λ-P e uma fórmula Φ, procura-se

(ii) determinar se existe algum critério $\mathbb{D}$ tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

Na análise que se segue, define-se inicialmente o passo de derivação por retrocesso. Em seguida é considerada a sequência derivação por retrocesso, que é uma composição de vários passos de derivação por retrocesso. Esta derivação é ilustrada através de um λ-programa equivalente ao programa lógico de ordem superior apresentado no exemplo 2.18. Nesta ilustração é construída uma sequência de derivação por retrocesso a partir de uma fórmula Φ dada. A definição de passo de derivação por retrocesso, dada a seguir, baseia-se na definição de P-derivação encontrada em [Nadathur, 1987].

3.6.1 DEFINICAO. Passo de derivacao por retrocesso.

Sejam λ-P um programa, θ_1 uma substituição resposta e OB_1 um conjunto de fórmulas, denominado conjunto objetivo. Um passo de derivação por retrocesso a partir de

$$\langle \lambda\text{-}P,\ \theta_1, OB_1 \rangle$$

é apresentado por

$$\langle \lambda\text{-}P,\ \theta_2, OB_2 \rangle$$

onde θ_2 e OB_2 são obtidos como descrito a seguir.

(i) Reducao do objetivo.

(a) Se existe uma fórmula Φ e uma substituiçãao resposta θ tal que

$$\Phi \in OB_1,$$
$$\Psi \in \lambda\text{-}P$$
$$\Psi = \Phi\theta$$

entãao

$$OB_2 = (OB_1 - \Phi)\theta,$$

$$\theta_2 = \theta_1\theta.$$

(b) Se existe uma fórmula $\Phi \in OB_1$ e

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (\Psi_1 \wedge \Psi_2),$$

entãao

$$OB_2 = (OB_1 - \Phi) \cup (\lambda X_1 \ldots \lambda X_n \Psi_1 \ , \ \lambda X_1 \ldots \lambda X_n \Psi_2),$$

$$\theta_2 = \theta_1.$$

(c) Se existe uma fórmula $\Phi \in OB_1$ onde

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (\Psi_1 \vee \Psi_2),$$

entãao

$$OB_2 = (OB_1 - \Phi) \cup (\lambda X_1 \ldots \lambda X_n \Psi_1)$$

ou

$$OB_2 = (OB_1 - \Phi) \cup (\lambda X_1 \ldots \lambda X_n \Psi_2),$$

e

$$\theta_2 = \theta_1.$$

(d) Se existe uma fórmula $\Phi \in OB_1$ onde

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (\Sigma \ \lambda Y \ P),$$

entãao para alguma variável Z que nãao ocorre em θ_1, OB_1 e $\lambda\text{-}P$,

$$OB_2 = (OB_1 - \Phi) \cup \{\lambda X_1 \ldots \lambda X_n \ ((\lambda Y \ P)Z)\},$$

$$\theta_2 = \theta_1.$$

(ii) Passo de retrocesso.

Se existe $\Phi \in OB_1$ e $\Psi \in \lambda\text{-}P$ onde Φ possui uma derivação em Ψ conforme a lista compatível $[A_1,\ldots,A_e]$ com substituição resposta θ. Isto é, existe ζ tal que,

$$\zeta = \Psi A_1 \ldots A_e$$

e

$$\zeta = \Phi\theta$$

Então, OB_2 e a substituição são obtidas como se segue. Considere

$$OB = (OB_1 - \Phi)\theta$$

OB é obtido retirando a fórmula Φ de OB_1 e aplicando a substituição θ às fórmulas do conjunto resultante.

$$OB_2 = OB \cup (A_1,\ldots,A_e)$$

e

$$\theta_2 = \theta_1\theta$$

***

3.6.2 DEFINICAO. Sequencia de derivacao por retrocesso.

Sejam $\lambda\text{-}P$ um programa, θ_1 uma substituição e OB_1 um conjunto de fórmulas, denominado conjunto objetivo. Uma sequência de derivação por retrocesso a partir de

$$\langle \lambda\text{-}P,\ \theta_1, OB_1 \rangle$$

é uma sequência dada por:

$$\langle \lambda\text{-}P,\ \theta_1, OB_1 \rangle, \ldots, \langle \lambda\text{-}P,\ \theta_n, OB_n \rangle, \ldots$$

onde

$$\langle \lambda\text{-}P,\ \theta_{i+1}, OB_{i+1} \rangle$$

é um passo de derivação por retrocesso a partir de

$$\langle \lambda\text{-}P,\ \theta_i, OB_i \rangle$$

A sequência de derivação por retrocesso é uma sequência com sucesso se

$$OB_j = \emptyset$$

para algum j.

***

EXEMPLO 3.1

Considera-se neste exemplo a representação do programa lógico de ordem superior, indicado no exemplo 2.18, como um λ-programa λ-P. Este λ-programa é utilizado para ilustrar a derivação de conhecimento por retrocesso. É dado um conjunto objetivo

$$OB_1 = (\Phi)$$

e a partir de

$$\langle \lambda\text{-}P,\ \theta,\ OB_1 \rangle$$

onde

$$\theta = (\ )$$

é construída uma sequência de derivação por retrocesso com sucesso. A partir desta sequência de derivação por retrocesso, seguindo um raciocínio inverso, é possível definir um λ-programa λ-P_0 e um critério $\mathbb{D}$ tais que a fórmula Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

(i) DEFNICAO DE UM λ-PROGRAMA λ-P EQUIVALENTE AO PROGRAMA LOGICO DO EXEMPLO 2.18.

Na análise da derivação de conhecimento por retrocesso, é proposta uma nova lista de fórmulas para representar o mesmo conhecimento representado no programa lógico de ordem superior do exemplo 2.18. As modificações são necessárias pois em um programa lógico as fórmulas são proposições. Elas estão associadas ao símbolos para tipo "t". Por outro lado, em um programa com critério deve-se ter fórmulas que representem funções com um ou mais argumentos, e não apenas proposições.

CONVENCAO: No que se segue, os símbolos para tipo associados às variáveis e constantes que constituem as fórmulas são indicados somente quando há importância em fazê-lo.

***

Inicia-se a definição do λ-programa λ-P pelas constantes

BENEDITO, SONIA, AGUINALDO

que representam os indivíduos de uma família. Considere agora os predicados

$$\lambda X\ \lambda Y\ (\textbf{PAI}\ X\ Y),$$

$$\lambda X\ \lambda Y\ (\textbf{ESPOSA}\ X\ Y)$$

onde

$$tipo[\lambda X\ \lambda Y\ (\textbf{PAI}\ X\ Y)] = \alpha,$$

$$tipo[\lambda X\ \lambda Y\ (\textbf{ESPOSA}\ X\ Y)] = \alpha$$

que denotam os predicados **PAI** e **ESPOSA** na notação do λ-cálculo. As funções

$$\lambda Y\ (\textbf{RELACAO-PRIMITIVA}\ Y),$$

$$\lambda Y\ (\textbf{RELACAO-FAMILIAR}\ Y),$$

onde

$$tipo[\lambda Y\ (\textbf{RELACAO-PRIMITIVA}\ Y)] = \beta,$$

$$tipo[\lambda Y\ (\textbf{RELACAO-FAMILIAR}\ Y)] = \beta,$$

representam os predicados **RELACAO-PRIMITIVA** e **RELACAO-FAMILIAR**. Associados a tais predicados, tem-se as proposições

RELACAO-PRIMITIVA PAI

RELACAO-PRIMITIVA ESPOSA

Para representar as cláusulas do programa lógico, consideram-se as funções a seguir. Inicia-se por FC1, que representa a cláusula

$$C_1 : \textbf{RELACAO-FAMILIAR R :- RELACAO-PRIMITIVA R}$$

do programa lógico.

$$FC1 = \lambda Y_\beta\ (f\ Y_\beta)$$

onde a função f é definida a seguir.

(i) Se

$$Y = \textbf{RELACAO-PRIMITIVA R}$$

então

$$f\ Y = \textbf{RELACAO-FAMILIAR R}$$

(ii) Se

$$Y \neq \textbf{RELACAO-PRIMITIVA R}$$

então

$$f\ Y = T$$

onde "T" representa "verdadeiro". O resultado da aplicação de f sobre o seu argumento é "T", quando ele não satisfaz a condição

$$Y = \textbf{RELACAO-PRIMITIVA R}$$

Neste caso, o resultado de f não adiciona nenhum conhecimento ao sistema.

Fazendo

$$Y = \textbf{RELACAO-PRIMITIVA PAI}$$

então,

$$FC1\ Y = \textbf{RELACAO-FAMILIAR PAI}$$

Considere agora a função FC2 que representa a cláusula

$$C_2 : \textbf{RELACAO-FAMILIAR}\ (\lambda X\ \lambda Y\ (\ \sum \lambda V\ [(P\ X\ V) \wedge (Q\ V\ Y)])\ ) :-$$

$$\textbf{RELACAO-PRIMITIVA P},$$

$$\textbf{RELACAO-PRIMITIVA Q}.$$

do programa lógico.

$$FC2 = \lambda X_\beta\ \lambda Y_\beta\ (g\ X_\beta\ Y_\beta\ )$$

onde a função g é definida como se segue,

(i) Se

$$X = \textbf{RELACAO-PRIMITIVA P}$$

$$Y = \textbf{RELACAO-PRIMITIVA Q}$$

então

$$g\ X\ Y = \textbf{RELACAO-FAMILIAR}\ (\lambda X\ \lambda Y\ (\ \sum \lambda V\ [(P\ X\ V) \wedge (Q\ V\ Y)])$$

(ii) Se

$$X \neq \textbf{RELACAO-PRIMITIVA P}$$

ou

$$Y \neq \textbf{RELACAO-PRIMITIVA Q}$$

então

$$g\ X\ Y = T$$

onde "T" representa "true". De uma maneira analoga à definição da função f, o resultado da aplicação de g sobre os seus argumentos é "T", quando tais argumentos não satisfazem as condições

$$X = \textbf{RELACAO-PRIMITIVA P}$$

$$Y = \textbf{RELACAO-PRIMITIVA Q}$$

Fazendo

$$X = \textbf{RELACAO-PRIMITIVA PAI}$$

$$Y = \textbf{RELACAO-PRIMITIVA ESPOSA}$$

então,

$$\left[\begin{array}{l} FC2\ X\ Y = \\ \quad = \textbf{RELACAO-FAMILIAR}\ (\lambda X\ \lambda Y\ (\ \sum \lambda V\ [(\textbf{PAI}\ X\ V) \wedge (\textbf{ESPOSA}\ V\ Y)]) \end{array}\right.$$

O programa λ-P é definido a seguir,

$$\lambda\text{-}P = [\textbf{BENEDITO, SONIA, AGUINALDO,}$$

$$\textbf{PAI BENEDITO SONIA,}$$

$$\textbf{ESPOSA SONIA AGUINALDO,}$$

$$\textbf{RELACAO-PRIMITIVA PAI,}$$

$$FC1, FC2\ ]$$

(ii) UMA SEQUENCIA DE DERIVACAO POR RETROCESSO.

Apresenta-se a seguir uma sequência de derivação por retrocesso com sucesso a partir de

$$\langle \lambda\text{-}P,\ \theta,\ OB_1 \rangle$$

onde λ-P é o λ-programa definido na seção anterior, OB_1 é o conjunto objetivo

$$OB_1 = \{\ (\textbf{RELACAO-FAMILIAR}\ R) \wedge (R\ \textbf{BENEDITO AGUINALDO})\ \}$$

e θ é a substituição vazia

$$\theta_1 = \{\ \}$$

Constrói-se uma sequência

$$\langle \lambda\text{-}P, \theta_1, OB_1 \rangle,\ \langle \lambda\text{-}P, \theta_2, OB_2 \rangle, \ldots,\ \langle \lambda\text{-}P, \theta_8, OB_8 \rangle,$$

onde cada passo de derivação por retrocesso é indicado a seguir. Esta sequência é uma sequência de derivação por retrocesso com sucesso, pois

$$OB_8 = \{\ \}.$$

Nos passos de derivação por retrocesso as setas são rotuladas pelas regras da definição 3.7.1, que são utilizadas na dedução de OB_i e θ_i.

$\langle \lambda\text{-}P, \theta_1, OB_1 \rangle$ <-------> OB_1 = ((RELACAO-FAMILIAR R)∧ (R BENEDITO AGUINALDO)); θ_1 = ( )

$\langle \lambda\text{-}P, \theta_2, OB_2 \rangle$ <(i)(b)> OB_2 = ((RELACAO-FAMILIAR R), (R BENEDITO AGUINALDO)); θ_2 = ( )

$\langle \lambda\text{-}P, \theta_3, OB_3 \rangle$ <(ii)> OB_3 = ((RELACAO-PRIMITIVA P), (RELACAO-PRIMITIVA Q),

(λX λY (∑ λV [(P X V)∧(Q V Y)]) BENEDITO AGUINALDO)

θ_3 = {⟨R, λX λY (∑ λV [(P X V)∧(Q V Y)])⟩}

Utilizando a forma normal, tem-se que,

OB_3 = ((RELACAO-PRIMITIVA P), (RELACAO-PRIMITIVA Q), ∑ λV [(P BENEDITO V)∧(Q V AGUINALDO)] }

θ_3 = {⟨R, λX λY (∑ λV [(P X V)∧(Q V Y)])⟩}

$\langle \lambda\text{-}P, \theta_4, OB_4 \rangle$ <(i)(d)> OB_4 = ((RELACAO-PRIMITIVA P), (RELACAO-PRIMITIVA Q), (P BENEDITO Z)∧(Q Z AGUINALDO) }; θ_4 = θ_3

$\langle \lambda\text{-}P, \theta_5, OB_5 \rangle$ <(i)(b)> OB_5 = ((RELACAO-PRIMITIVA P), (RELACAO-PRIMITIVA Q), (P BENEDITO Z), (Q Z AGUINALDO) }; θ_5 = θ_4

$\langle \lambda\text{-}P, \theta_6, OB_6 \rangle \xrightarrow{(i)(a)}$

$OB_6 = (\langle$ **RELACAO-PRIMITIVA Q**$\rangle$,
$\langle$ **PAI BENEDITO Z**$\rangle$,
$\langle$ **Q Z AGUINALDO**$^2\rangle$ $)$

$\theta_6 = \theta_5\theta$,

$\theta = (\langle$ **P, PAI**$\rangle)$

$\langle \lambda\text{-}P, \theta_7, OB_7 \rangle \xrightarrow{(i)(a)}$

$OB_7 = (\langle$ **PAI BENEDITO Z**$\rangle$,
$\langle$ **ESPOSA Z AGUINALDO**$\rangle$ $)$

$\theta_7 = \theta_6\theta$,

$\theta = (\langle$ **Q, ESPOSA**$\rangle)$

$\langle \lambda\text{-}P, \theta_8, OB_8 \rangle \xrightarrow{(i)(a)}$

$OB_8 = (\ )$

$\theta_8 = \theta_7\theta$,

$\theta = (\langle$ **Z, SONIA**$\rangle)$

Observe que esta sequência de derivação por retrocesso é uma sequência com sucesso pois

$$OB_8 = (\ )$$

Tem-se também que θ_8 é a composição das seguintes substituições

$$(\langle \mathbf{R},\ \lambda X\ \lambda Y\ (\ \sum \lambda V\ [(P\ X\ V) \wedge (Q\ V\ Y)])\rangle)$$

$$(\langle \mathbf{P},\ \mathbf{PAI}\rangle)$$

$$(\langle \mathbf{Q},\ \mathbf{ESPOSA}\rangle)$$

$$(\langle \mathbf{Z},\ \mathbf{SONIA}\rangle)$$

logo

$\theta_8 = \{\langle$ **R**, $\lambda X\ \lambda Y\ (\ \sum \lambda V\ [($**PAI** $X\ V) \wedge ($**ESPOSA** $V\ Y)])\rangle\}$

$\langle$ **P, PAI**$\rangle$ $\langle$ **Q, ESPOSA**$\rangle$ $\langle$ **Z, SONIA**$\rangle$ $\}$

Nas seções a seguir é definido um critério de derivação $\mathbb{D}$ e um λ-programa $\lambda\text{-}P_o$ tal que a fórmula

Φ = (RELACAO-FAMILIAR R)$\wedge$(R BENEDITO AGUINALDO)

é uma dedução por derivação a partir do pograma com critério $\langle \lambda\text{-}P_o, \mathbb{D} \rangle$. Inicia-se pelo λ-programa $\lambda\text{-}P_o$.

***

Como é indicado no início desta seção, na derivação de conhecimento por retrocesso, ou "backward", dado um λ-programa $\lambda\text{-}P$ e uma fórmula Φ, procura-se determinar se existe algum critério $\mathbb{D}$ tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

A partir de uma sequência de derivação por retrocesso, como a indicada neste exemplo, é possível definir o critério $\mathbb{D}$ tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Isto é feito transformando a sequência de derivação por retrocesso (backward) em uma sequência de derivação (forward). Dada uma sequência de derivação por retrocesso a transformação desta sequência em uma sequência de derivação (forward) é obtida utilizando os seguintes passos:

(i) Defina um conjunto de funções que representam as operações de "redução do objetivo", conforme a definição de passo de derivação por retrocesso, definição 3.8.1.

(ii) Associe as funções definidas em (i) ao λ-programa original $\lambda\text{-}P$, obtendo um novo λ-progrma $\lambda\text{-}P_o$.

(ii) Construa a sequência de derivação (forward) seguindo o sentido inverso da sequência de derivação por retrocesso (backward). Esta sequência é considerada a partir do λ-progama $\lambda\text{-}P_o$.

(iii) Defina o critério de derivação $\mathbb{D}$ "ponto a ponto" de tal forma que se tenha uma sequência de derivação a partir de $\langle \lambda\text{-}P_o, \mathbb{D} \rangle$.

A transformaçao da sequência de derivação por retrocesso aprsentada no exemplo 3.1 em uma sequencia de derivação "forward" e a definição de um critério de derivação $\mathbb{D}$, que satisfaz as condições indicadas acima é apresentada no apêndice B.

3.7 CONCLUSAO.

Neste capítulo foram introduzidas as noções de λ-programa e de derivação a partir destes programas. Um λ-programa foi definido como uma lista de fórmulas do λ-cálculo tipado. Como cada fórmula do λ-cálculo tipado representa uma função, um λ-programa é um programa funcional [Henderson, 1980]. Foi proposto um mecanismo de derivação de conhecimento a partir de um λ-programa com critério. Dado um λ-programa λ-P e um critério de derivação $\mathbb{D}$ obteve-se um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$ a partir do qual foi possível derivar conhecimento. Foi mostrado que a derivação pode ser obtida de várias formas. Propôs-se a derivação de conhecimento "forward", com memória e sem memória e derivação "backward", definida por sequências de derivação por retrocesso. A representação de conhecimento a partir de λ-programas com critério, como foi proposta neste trabalho, foi realizada não só pelas fórmulas, como pelos critérios de derivação propriamente ditos. Isto é, representou-se conhecimento também nos critérios de derivação. Esta condição não ocorre em linguagens, como PROLOG, em que se tem critérios de derivação rigidos [Casanova, 1987]. No caso da linguagem PROLOG o critério é definido pelo mecanismo da resolução.

A presença explícita de símbolos para tipo associados aos elementos de um programa com critério permitem definir relações de complexidade entre conhecimento representado por fórmulas do λ-cálculo tipado e determinar condições necessárias para a derivação de conhecimento a partir de programas com critério.

CAPITULO 4

COMPLEXIDADE DAS FORMULAS DO λ-CALCULO TIPADO.

Neste capitulo, sao propostas as relacoes de complexidade entre as formulas do λ-calculo tipado. Estas relacoes sao definidas a partir da ordem dos simbolos para tipo associados as variaveis e constantes das formulas. Tem-se as relacoes de complexidade quanto ao nivel, quanto a descricao total, quanto a descricao parcial e quanto ao grau. Estas relacoes determinam as comparacoes das complexidades dos conhecimentos representados por formulas do λ-calculo tipado.

4.1 INTRODUCAO.

Propõe-se neste capítulo as relações de "complexidade" entre as fórmulas do λ-cálculo tipado. Esta relações fundamentam-se na ordem dos símbolos para tipo associados as constantes e variáveis que compõem as fórmulas do λ-cálculo tipado. Propõe-se árvores de descrições dos símbolos para tipo e das fórmulas do λ-cálculo tipado, que descrevem sintaticamente os símbolos para tipo e as fórmulas. A partir destas descrições sintáticas são definidas as relações de "complexidade" quanto a descrição, quanto ao nível e quanto ao grau.

Inicialmente, na seção 4.2, é analisado informalmente as relações de "complexidade" entre as fórmulas do λ-cálculo tipado.

Em seguida, na seção 4.3, são definidos formalmente as relações de complexidade quanto a descrição e ao nível. Na seção 4.3.1 é definida a árvore de descrição de um símbolo para tipo. Posteriormente, da seção 4.3.4 à seção 4.3.6 são definidos, respectivamente, os conceitos de árvore de descrição parcial de uma fórmula, árvore de descrição total de uma fórmula e ordens total e parcial de uma fórmula. Em seguida, os conceitos de ordem parcial, ordem total e ordem de símbolos para tipo são utilizados para relacionar formalmente a complexidade de fórmulas do λ-cálculo tipado. Define-se, na seção 4.3.7 as relações de complexidade, quanto à descrição e ao nível, entre as fórmulas do λ-cálculo tipado.

Na seção 4.4 é analisado informalmente o conceito de complexidade quanto ao grau. Inicia-se propondo uma classificação do conhecimento de grau zero, de grau um e de grau superior.

Finalmente, na seção 4.5 propõe-se formalmente as relações de complexidade quanto ao grau. Nas seções 4.5.1 e 4.5.2 são definidos, respectivamente, o grau de uma fórmula e as relações de complexidade quanto ao grau.

4.2 A COMPLEXIDADE.

Analisa-se a seguir, informalmente, as relações de "complexidade" do conhecimento representado pelas fórmulas do λ-cálculo tipado. A definição do termo "complexidade" é introduzida informalmente a partir de exemplos específicos. A partir destes exemplos são estabelecidas relações entre as "complexidades" das fórmulas que representam o conhecimento.

Considera-se inicialmente as relações de "complexidade" apenas entre o conhecimento representado por proposições formadas a partir de predicados. As relações de "complexidades" entre fórmulas que não são proposições seguem um raciocínio análogo ao descrito a seguir. Sejam então P_α e Q_β dois predicados tais que

$$\alpha = (\alpha_1 \succ \alpha_2 \succ \ldots \alpha_n \succ \iota),$$

$$\beta = (\beta_1 \succ \beta_2 \succ \ldots \beta_m \succ \iota).$$

Logo,

$$tipo[P_\alpha\ X_{\alpha_1}\ X_{\alpha_2} \ldots\ X_{\alpha_n}] = \iota$$

$$tipo[Q_\beta\ Y_{\beta_1}\ Y_{\beta_2} \ldots\ Y_{\beta_m}] = \iota$$

A ordem do símbolo para tipo associado as proposições

$$P_\alpha\ X_{\alpha_1}\ X_{\alpha_2} \ldots\ X_{\alpha_n}$$

$$Q_\beta\ Y_{\beta_1}\ Y_{\beta_2} \ldots\ Y_{\beta_m}$$

é zero. Apesar da ordem ser nula nos dois casos, os conhecimentos representados pelas proposições podem ser diferentes no que se refere a sua "complexidade". Considere, por exemplo, que

$$\alpha = (\iota \succ \iota),$$

$$\beta = ((\iota \succ \iota) \succ \iota)$$

e que

$$[(P_\alpha \ X_t) = T] \leftrightarrow [X_t = T]$$

$$[(Q_\beta \ Y_{(\iota \succ \iota)}) = T] \leftrightarrow [Y_{(\iota \succ \iota)} = P_\alpha]$$

onde "T" representa a constante "verdade". Observa-se que o resultado da aplicação do predicado P_α é obtido a partir de um conhecimento sobre uma fórmula X_t. Isto é, P_α possui um conhecimento que decide quando as fórmulas X_t são verdadeiras ou falsas. Tem-se um conhecimento sobre fórmulas de ordem zero. O predicado Q_β possui um conhecimento que decide quando as fórmulas $Y_{(\iota \succ \iota)}$ são iguais ao predicado P_α. Portanto, Q_β possui um conhecimento sobre fórmulas associadas a um símbolo para tipo cuja ordem é 1 e não 0, como no caso do predicado P_α. Neste sentido, os conhecimentos representado pelos predicados acima são de natureza diferente. Os conhecimentos representados por P_α são de ordem 0 e aqueles representados por Q_β são de ordem 1. Assim sendo, considera-se que o predicado Q_β representa um conhecimento mais "complexo", pois Q_β possui um conhecimento sobre P_α.

Sejam agora os predicados P_α e Q_β relacionados pela seguinte proposição,

$$P_\alpha \ (Q_\beta \ Y_{(\iota \succ \iota)})$$

Neste caso P_α é aplicado sobre o resultado da aplicação do predicado Q_β. Isto é, P_α possui um conhecimento sobre o resultado da aplicação do predicado Q_β. Este fato não significa que P_α possui um conhecimento sobre Q_β. Pois caso ocorresse este conhecimento, P_α teria um conhecimento sobre Q_β, que por sua vez possui conhecimento sobre P_α (recursividade!). Como

$$ordem[\alpha]=1$$

e

$$ordem[\beta]=2$$

não se pode ter uma fórmula cujo símbolo para tipo associado é de ordem 1 com conhecimento sobre uma fórmula cujo símbolo para tipo é de ordem 2. Como é ressaltado no capítulo 1, ao se considerar a teoria dos tipos é esta recursividade que se deseja evitar, pois caso contrário seria possível representar conhecimento sobre si próprio.

"Responder"

$$P_{\alpha}\ (Q_{\beta}\ Y_{(t\succ t)})$$

é diferente de "responder" uma questão como

$$P_{\alpha}\ X_{t}.$$

No primeiro caso, P_{α} considera o resultado de um predicado mais "complexo", o que não ocorre no segundo caso. Neste sentido,

$$P_{\alpha}\ (Q_{\beta}\ Y_{(t\succ t)})$$

representa um conhecimento mais complexo que

$$P_{\alpha}\ X_{t}.$$

Considera-se a seguir as relações de "complexidade" entre cláusulas, que também são proposições do λ-cálculo tipado. São consideradas as relações de "complexidade" entre diferentes cláusulas, formadas a partir dos predicados P_{α} e Q_{β} consideradas anteriormente. Uma análise da "complexidade" dos conhecimento representado por cláusulas, inicia-se pela análise do predicado ":-". Observe que este predicado faz parte da cabeca da forma normal de uma cláusula. Quando se tem uma cláusula como,

$$P :- Q$$

está sendo utilizado a forma infixa do predicado ":-". A outra maneira de representa-la é:

$$((:-Q)\ P),$$

onde o tipo do predicado ":-" é:

$$tipo[:-] = (t \succ t \succ t)$$

Portanto, ":-" considera inicialmente uma fórmula associada ao símbolo para tipo "t", em seguida outra fórmula associada ao símbolo para tipo "t" e fornece como resultado uma fórmula associada ao símbolo para tipo "t". Uma cláusula como

$$((:-Q)\ P)$$

considera inicialmente o resultado da avaliação da proposição Q, em seguida o resultado da avaliação de P, fornecendo como resultado uma proposição. Neste sentido, o conhecimento representado em uma cláusula é apenas um conhecimento sobre o resultado da avaliação de fórmulas associadas ao símbolo para tipo "t". Como foi enfatizado anteriormente as fórmulas associadas ao símbolo para tipo "t" podem ser o resultado da manipulação de conhecimento mais "complexo" ou mais "simples".

Sejam as duas cláusulas a seguir, onde P_α e Q_β são os predicados indicados nas proposições anteriores. Observe que o predicado "=" representa a igualdade usual e está sendo utilizado na sua forma infixa.

$$\mathbf{W=BOM} :- P_\alpha\ X_t$$

$$\mathbf{Z=OTIMO} :- Q_\beta Y_{(t\rightarrow t)}$$

A primeira cláusula indica que a variável "W" é igual a "BOM" se o resultado do predicado

$$P_\alpha\ X_t$$

é verdadeiro e a segunda diz que "Z" é igual a "OTIMO", quando

$$Q_\beta\ Y_{(t\rightarrow t)}$$

é verdadeiro.

Como foi descrito anteriormente, considera-se que a "complexidade" do conhecimento representado pelo predicado Q_β é maior que a do predicado P_α. Neste sentido, tem-se também que a "complexidade" do conhecimento representado na segunda cláusula acima é maior que aquele representado na primeira. Seja agora seguinte cláusula:

$$\mathbf{W=EXCELENTE} :- Q_\beta\ Y_{(t\rightarrow t)},$$

$$Y_{(t\rightarrow t)}\ X_t$$

que é escrita na forma normal como,

$$((:-(\wedge\ (Q_\beta\ Y_{(t\rightarrow t)})\ Y_{(t\rightarrow t)}\ X_t)\ (\mathbf{W=EXCELENTE}))$$

Observe que neste caso, o primeiro argumento do predicado ":-" é resultado do predicado "$\wedge$" aplicado aos seus dois argumentos. Isto é, ":-" considera o resultado de "$\wedge$", que por sua vez considera o resultado das fórmulas

$$Q_\beta\ Y_{(t\rightarrow t)}$$

e

$$Y_{(t\rightarrow t)}\ X_t.$$

Neste sentido, considera-se que esta cláusula é ainda mais "complexa" que as duas cláusulas indicadas anteriormente, pois ela representa resultados, que são resultados da avaliação de predicados sobre seus argumentos.

4.3 RELACOES DE COMPLEXIDADE QUANTO A DESCRICAO E AO NIVEL.

O sentido do termo "complexidade", indicado de uma maneira informal na seção anterior, considera a forma sintática das fórmulas e os símbolos para tipo associados as variáveis e constantes utilizadas na representação do conhecimento. A seguir, inicia-se a formalização destes conceitos, considerando-se as árvores de descrição das fórmulas e dos símbolos para tipo. Informalmente, uma árvore de descrição "descreve" sintaticamente a complexidade de um símbolo para tipo ou de uma fórmula.

4.3.1 DEFINICAO. Arvore de descricao de um simbolo para tipo.

Dado um símbolo para tipo α, a sua árvore de descrição, indicada por arv[α], é dada por

(i) Se $\alpha \in \mathbb{F}$, então a árvore de descrição de α é constituída apenas por um vértice, rotulado por arv[α].

(ii) Se

$$\alpha = \langle c\ \alpha_1 \alpha_2 \ldots \alpha_n \rangle$$

$$\langle c, n \rangle \in \mathbb{C},$$

$$\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{T},$$

então a árvore de descrição de α é a árvore indicada na figura 4.1(a). Tem-se a raiz rotulada por arv[α] e ligada a cada um dos vértices "filhos" tem-se as sub-árvores

$$arv[\alpha_1], arv[\alpha_2], \ldots, arv[\alpha_n].$$

(iii) Se

$$\alpha = \langle \beta \succ \tau \rangle,$$

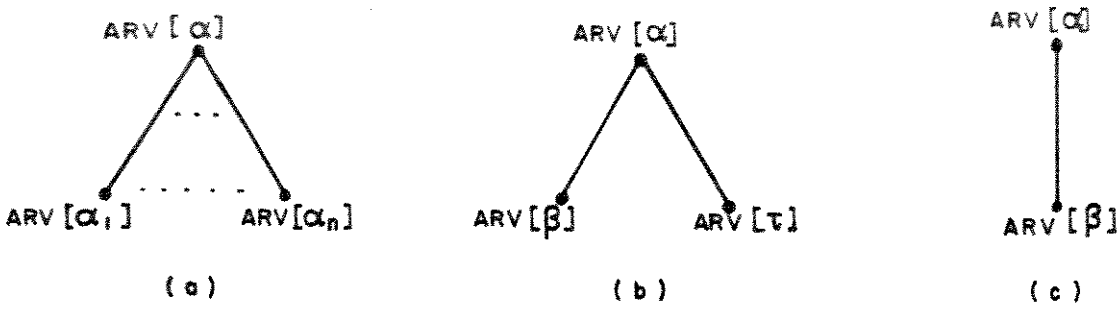
então a árvore de descrição de α é aquela indicada na figura 4.1(b). Tem-se uma raiz rotulada por arv[α] conectada a dois vértices "filhos", aos quais estão ligadas as sub-árvores arv[β] e arv[τ].

(iv) Se

$$\alpha = \langle \beta \rangle,$$

então a árvore de descrição de α é aquela indicada na figura 4.1(c). Neste caso, tem-se a raiz rotulada com arv[α] e apenas um vértice "filho", ao qual está ligada a sub-árvore arv[β].

***



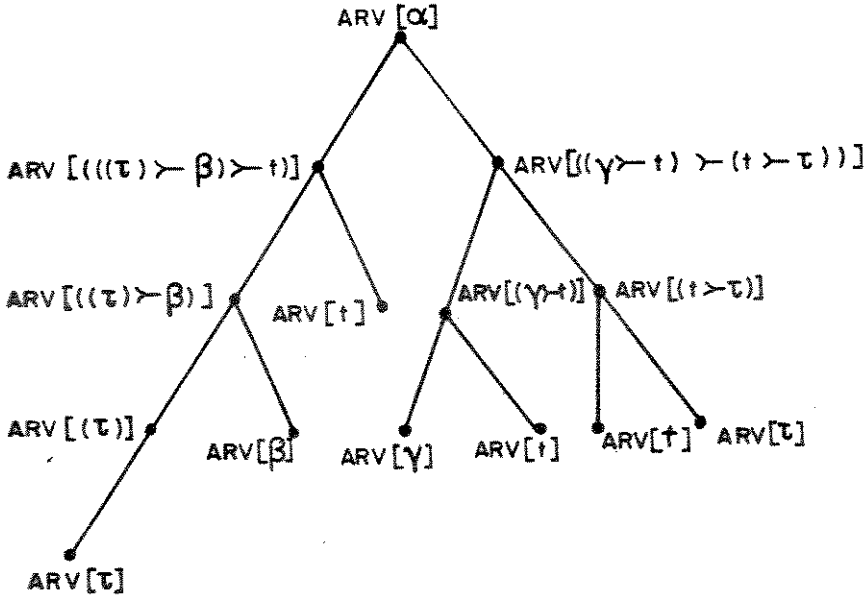
Arvore de descricao de um simbolo para tipo.
FIGURA 4.1

NOTACAO. E utilizado neste trabalho a notação arv[α] como referência à árvore de descrição do símbolo para tipo α.

EXEMPLO 4.1.

Considere o seguinte símbolo para tipo, onde são utilizados os símbolos "[" e "]" no lugar de parênteses para facilitar a leitura. A sua árvore de descrição arv[α] é indicada na figura 4.2.

$$\alpha = [[[\tau] \succ \beta] \succ \iota] \succ [[\gamma \succ \iota] \succ [\iota \succ \tau]]$$



Arvore de descricao do simbolo para tipo α.
FIGURA 4.2

***

Apresenta-se a seguir um conjunto de termos e a definição de profundidade de uma árvore, análogos àqueles encontrados em teoria do grafos [Bolobás, 1979].

4.3.2 DEFINICAO. Profundidade de uma arvore [Bolobás, 1979].

Considere uma árvore T e as seguintes definições,

(i) A distância de uma folha f_i de T a sua raiz r é igual ao menor número de arestas que ligam a folha à raiz. Esta distância é indicada por $dist[f_i,r]$. Se o menor caminho que liga uma folha f_i à raiz r (figura 4.3) é, por exemplo, a sequência

$$(f_i \; a_1 \; v_1 \; a_2 \; v_2 \; a_3 \ldots a_n \; r)$$

onde

$$a_1, a_2, \ldots, a_n,$$

e

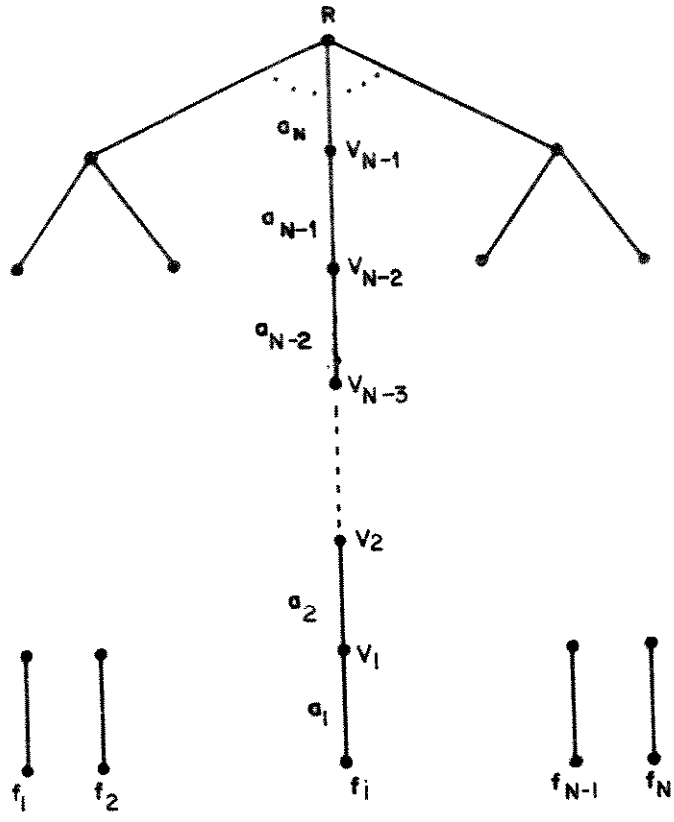
$$v_1, v_2, \ldots, v_{n-1}$$

são respectivamente as arestas e vértices que ligam f_i a r, então

$$dist[f_i,r] = n$$

(ii) Sejam $f_1 \; f_2 \ldots f_n$ as folhas de T, então a profundidade de T, indicada por $prof[T]$, é definida por:

$$prof[T] = \max_{1 \le i \le n} \{dist[f_i,r]\}$$



Profundidade de uma arvore.
FIGURA 4.3

***

4.3.3 PROPOSICAO.

Dado um símbolo para tipo α, a profundidade de sua árvore de descrição é igual a sua ordem. Isto é,

$$prof[\ arv[\alpha]\ ] = ordem[\alpha]$$

Demonstracao.

A demonstração é feita por indução na ordem de α.

(i) Se $ordem[\alpha]=0$, então $\alpha \in \mathbb{F}$, logo o resultado é claramente verdadeiro.

(ii) Considere que

$$prof[\ arv[\alpha]\ ] = ordem[\alpha]$$

no caso em que $ordem[\alpha] \leq k$.

Supondo $ordem[\alpha]=k+1$, $k>0$, então α possui uma das formas a seguir:

(a) $\alpha=(c\ \alpha_1\alpha_2\ldots\alpha_s)$

(b) $\alpha=(\beta)$

(c) $\alpha=(\beta \succ \gamma)$

Considere o caso (a). Logo existe j tal que

$$k = ordem[\alpha_j] = max\ (ordem[\alpha_i],\ 1\leq i\leq s).$$

Assim, por hipótese de indução,

$$prof[\ arv[\alpha_j]\ ] = k.$$

e pelo item (ii) da definição 4.3.1, tem-se que

$$prof[\ arv[\alpha]\ ] = prof[\ arv[\alpha_j]\ ] + 1$$

Logo, a profundidade da árvore de descrição de α é igual a $k+1$, isto é,

$$\begin{aligned} prof[\ arv[\alpha]\ ] &= k+1 \\ &= ordem[\alpha] \end{aligned}$$

A demonstração dos outros casos é análoga.

***

A partir da definição de árvore de descrição de um símbolo para tipo é definido a seguir as árvores de descrição parcial e total de uma fórmula. Estes conceitos são utilizados na formalização das relações de complexidade entre as fórmulas do λ-cálculo tipado.

4.3.4 DEFINICAO. Arvore de descricao parcial de uma formula.

Dada uma fórmula Φ_α, a sua árvore de descrição parcial, denotada por $arv[\Phi_\alpha]^P$, é definida por

(i) Se

$$\Phi_\alpha \in CONS_\alpha \cup VAR_\alpha,$$

então $arv[\Phi_\alpha]^P$ é apenas o nó raiz rotulado por $arv[\Phi_\alpha]^P$.

(ii) Se

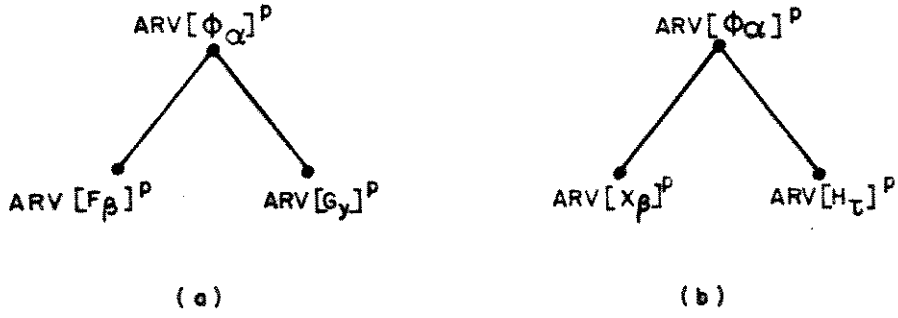
$$\Phi_\alpha = F_\beta G_\gamma,$$

então a árvore de descrição parcial de Φ_α é aquela indicada na figura 4.4(a). Nesta figura a raiz é rotulada por $arv[\Phi_\alpha]^P$ e os dois vértices "filhos" são rotulados por $arv[F_\beta]^P$ e $arv[G_\gamma]^P$.

(iii) Se

$$\Phi_\alpha = \lambda\, x_\beta\, H_\tau,$$

então a árvore de descrição parcial é aquela indicada na figura 4.4(b). Neste caso a raiz é rotulada por $arv[\Phi_\alpha]^P$ e dois vértices "filhos" são rotulados por $arv[x_\beta]^P$ e $arv[H_\tau]^P$.



Arvore de descricao parcial de uma formula.
FIGURA 4.4.

4.3.5 DEFINICAO. Arvore de descricao total de uma formula.

Dada uma fórmula Φ_α, a sua árvore de descrição total, denotada por $arv[\Phi_\alpha]^t$, é definida por

(i) Se

$$\Phi_\alpha \in CONS_\alpha \cup VAR_\alpha,$$

então

$$arv[\Phi_\alpha]^t = arv[\alpha].$$

(ii) Se

$$\Phi_\alpha = F_\beta G_\gamma,$$

então a árvore de descrição total de Φ_α é a raiz rotulada por $arv[\Phi_\alpha]^t$, ligada aos dois vértices "filhos" rotulados por $arv[F_\beta]^t$ e $arv[G_\gamma]^t$.

(iii) Se

$$\Phi_\alpha = \lambda\, x_\beta\, H_\tau,$$

então a árvore de descrição total é a raiz rotulada por $arv[\Phi_\alpha]^t$, ligada aos dois vértices "filhos" rotulados por $arv[x_\beta]^t$ e $arv[H_\tau]^t$.

***

4.3.6 DEFINICAO. Ordem de uma formula.

(i) A ordem parcial de uma fórmula Φ_α, indicada por $ordem[\Phi_\alpha]^p$, é dada por:

$$ordem[\Phi_\alpha]^p = prof[\ arv[\Phi_\alpha]^p\ ].$$

(ii) A ordem total de uma fórmula Φ_α, indicada por $ordem[\Phi_\alpha]^t$, é dada por:

$$ordem[\Phi_\alpha]^t = prof[\ arv[\Phi_\alpha]^t\ ].$$

***

OBSERVACAO: Quando não houver necessidade em distinguir as ordens parcial e total de uma fórmula, os indices "p" e "t" serão omitidos.

EXEMPLO 4.2.

Considera-se a seguir as árvores de descrição e o cálculo das ordens parciais e totais das proposições

$$P_\alpha$$

$$P_\alpha\ x_t$$

$$Q_\beta\ Y_{(t>t)}$$

$$P_\alpha\ Q_\beta\ Y_{(t>t)}$$

que foram analisadas na seção anterior. Tem-se que P_α e Q_β são predicados onde,

$$\alpha=(t \succ t),$$

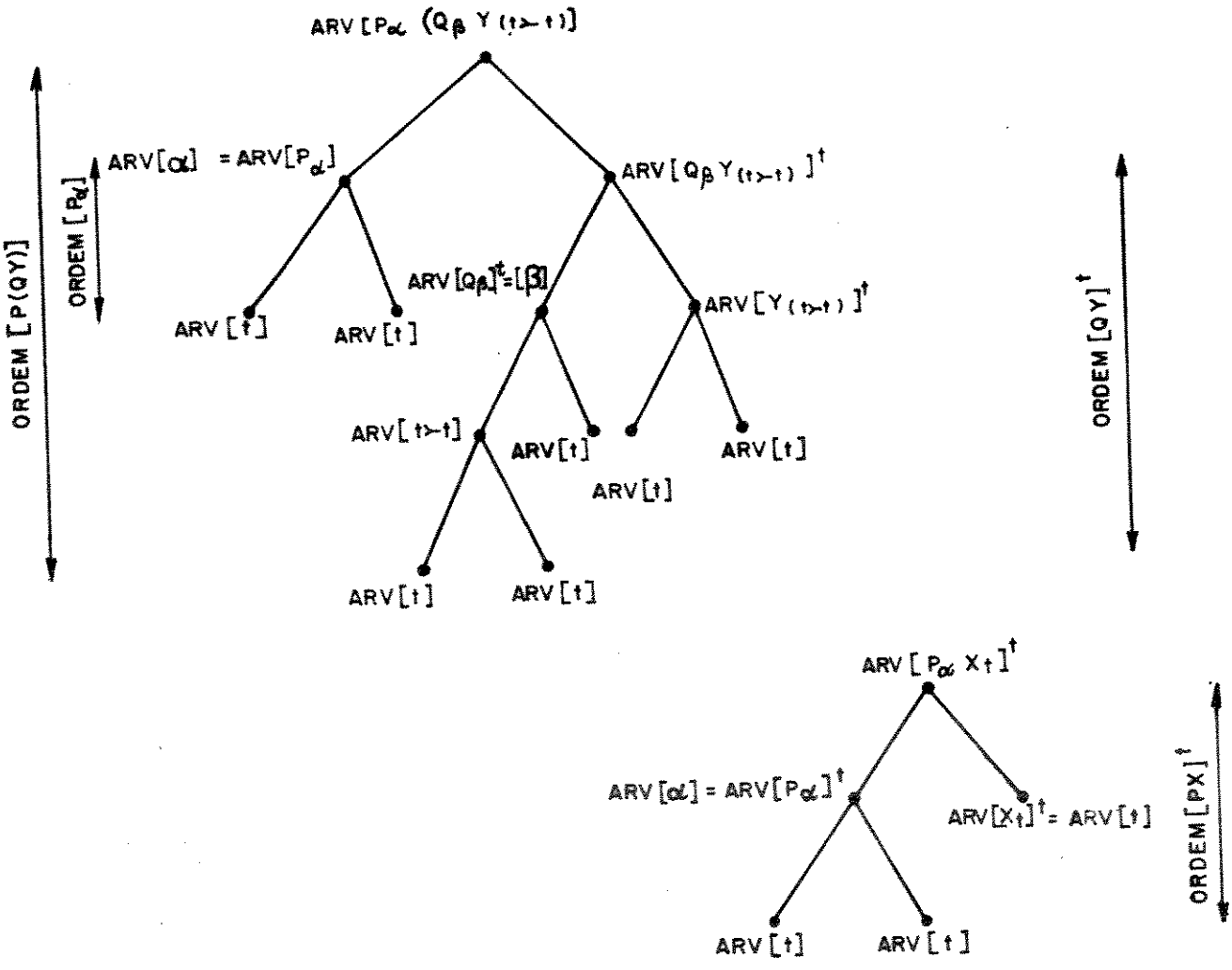
$$\beta=(t \succ t \succ t)$$

As árvores de descrição total e parcial destas fórmulas são indicadas na figura 4.5. Observe que a profundidade das árvores aumenta à medida em que se tem mais conhecimento representado, ou uma maior "complexidade", conforme foi analisado na seção anterior. Verificando-se

$$ordem[P_\alpha] < ordem[P_\alpha{}^{X_t}] < ordem[Q_\beta{}^{Y_{(t>t)}}] < ordem[P_\alpha{}^{(Q_\beta{}^{Y_{(t>t)}})}]$$

Os indices "p" e "t" são omitidos pois as relações acima são validas tanto para as ordens parciais como para as totais.

***



Arvores de descricao total e parcial.

FIGURA 4.5

Define-se a seguir as relações de complexidade, quanto a descrição e ao nível, entre as fórmulas do λ-cálculo tipado. As relações dividem-se em dois grupos: relações de complexidade entre fórmulas associadas ao mesmo símbolo para tipo e relações de complexidade entre fórmulas associadas a símbolos para tipo diferentes.

4.3.7 DEFINICAO. Relacoes de complexidade, quanto a descricao e ao nivel, entre formulas do λ-calculo tipado.

Dadas as fórmulas Φ_α e Ψ_β tem-se as seguintes relações de complexidade entre elas.

(i) caso 1. $\alpha \neq \beta$

Diz-se que Φ_α e Ψ_β são equivalentes quanto ao nível, o que é indicado por

$$\Phi_\alpha =^n \Psi_\beta,$$

se

$$\text{ordem}[\ \alpha\ ] = \text{ordem}[\ \beta\ ]$$

Φ_α é mais complexa que Ψ_β quanto ao nível, ou equivalentemente, Φ_α está em um nível hierárquico superior a Ψ_β, o que é indicado por

$$\Phi_\alpha >^n \Psi_\beta$$

se

$$\text{ordem}[\ \alpha\ ] > \text{ordem}[\ \beta\ ]$$

Os símbolos $=^n$ e $>^n$ representam as relações de complexidade quanto ao nível. No caso em que

$$\text{ordem}[\alpha] = k$$

utiliza-se a notação

$$\alpha =^n k$$

(ii) caso 2. ordem[α] = ordem[β].

(a) Φ_α é mais complexa que Ψ_β quanto a descrição parcial, o que é indicado por

$$\Phi_\alpha >^{dp} \Psi_\beta$$

se

$$\text{ordem}[\ \Phi_\alpha]^p > \text{ordem}[\ \Psi_\beta\ ]^p$$

Neste caso, utiliza-se também a notação

$$\Phi_\alpha =^{dp} \Psi_\beta + k$$

onde

$$k = \text{ordem}[\ \Phi_\alpha]^p - \text{ordem}[\ \Psi_\beta\ ]^p$$

Quando k=0, então Φ_α e Ψ_β são equivalentes quanto a descrição parcial, isto é,

$$\Phi_\alpha \equiv^{dp} \Psi_\beta$$

No caso em que

$$ordem[\Phi_\alpha]^p = k$$

utiliza-se a notação

$$\Phi_\alpha \equiv^{dp} k$$

(b) Φ_α é mais complexa que Ψ_β quanto a descrição total, o que é indicado por

$$\Phi_\alpha >^{dt} \Psi_\beta$$

se

$$ordem[\Phi_\alpha]^t > ordem[\Psi_\beta]^t$$

Neste caso, utiliza-se também a notação

$$\Phi_\alpha \equiv^{dp} \Psi_\beta + k$$

onde

$$k = ordem[\Phi_\alpha]^p - ordem[\Psi_\beta]^p$$

Quando k=0, então Φ_α e Ψ_β são equivalentes quanto a descrição total, isto é,

$$\Phi_\alpha \equiv^{dt} \Psi_\beta$$

No caso em que

$$ordem[\Phi_\alpha]^t = k$$

utiliza-se a notação

$$\Phi_\alpha \equiv^{dt} k$$

***

EXEMPLO 4.3.

Considera-se neste exemplo as relações de complexidade entre as fórmulas,

$$\Phi_\alpha = \lambda X_\tau \, (A_{\langle\langle t\rangle t\rangle \gamma} \, B_{\langle t\rangle t})_\gamma$$

e

$$\Psi_\alpha = \lambda X_\tau \, (C_{\langle t\rangle \gamma} \, D_t)_\gamma$$

onde

$$\tau, \gamma \in \mathbb{F}.$$

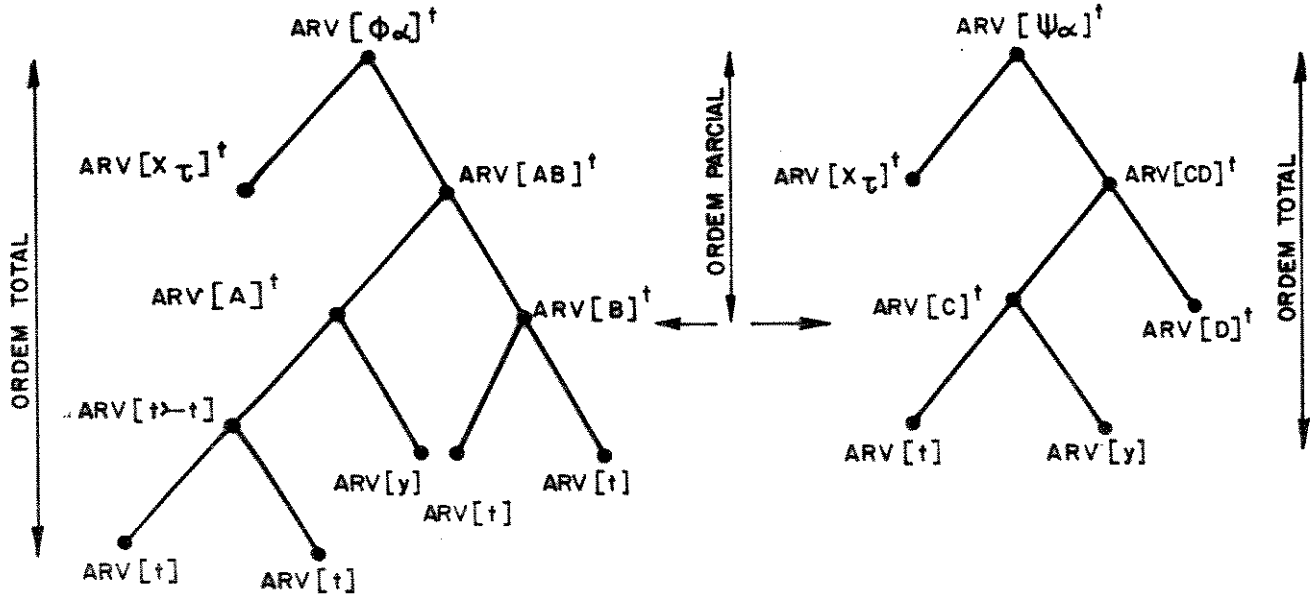
Tais relações são obtidas a partir das árvores de descrição total e parcial e dos símbolos para tipo associados as fórmulas propriamente ditas. As árvores de descrição são indicadas na figura 4.6. A partir delas, tem-se que:

$$ordem[\Phi_\alpha]^t = 4,$$

$$ordem[\Phi_\alpha]^p = 2,$$

$$ordem[\Psi_\alpha]^t = 3,$$

$$ordem[\Psi_\alpha]^p = 2,$$



Arvores de descricao.

FIGURA 4.6

Observe que as fórmulas são diferentes, mas são equivalentes quanto ao nível pois estão associadas ao mesmo símbolo para tipo. Elas também são equivalentes quanto descrição parcial pois

$$ordem[\Phi_\alpha]^p = ordem[\Psi_\alpha]^p = 2.$$

A equivalência quanto à descrição parcial deve-se ao fato de que as fórmulas possuem uma descrição sintática equivalente, (à menos de renomeação das constantes e variáveis). Quando não se considera os símbolos para tipo associados às variáveis e constantes, tem-se que

$$\Phi = \lambda X\ (AB)$$

$$\Psi = \lambda X\ (CD)$$

A descrição total considera os símbolos para tipos e neste caso Φ_α é mais complexa que Ψ_β pois

$$ordem[\Phi_\alpha]^t > ordem[\Psi_\alpha]^t$$

Considere agora o caso em que

$$\gamma = ((\iota \succ \iota) \succ \iota),$$

isto é,

$$ordem[\ \gamma\ ] = 2$$

e

$$ordem[\ \alpha\ ] = ordem[\ (\tau \succ \gamma)\ ] = 3.$$

Logo,

$$ordem[\Phi_\alpha]^t = 5,$$

$$ordem[\Phi_\alpha]^p = 2,$$

$$ordem[\Psi_\alpha]^t = 5,$$

$$ordem[\Psi_\alpha]^p = 2,$$

$$ordem[\alpha] = 3$$

Assim, Φ e Ψ são fórmulas diferentes mas equivalentes quanto ao nível e quanto às descrições parcial e total. Isto ocorre porque com o aumento da ordem do símbolo para tipo γ, este passa a ter um peso decisivo na descrição total das fórmulas, eliminando a influência dos outros símbolos para tipo.

***

4.4 COMPLEXIDADE QUANTO AO GRAU.

A seguir analisa-se a relação de complexidade quanto ao grau, entre as fórmulas do λ-cálculo tipado. Esta relação de complexidade é proposta a partir do conceito de complexidade quanto ao nível.

A representação do conhecimento é introduzida informalmente a partir do seguinte paradigma.

> **Tem-se um conjunto de robots que transportam cargas entre diversos pontos de uma fabrica. As pistas utilizadas pelos robots sao retilineas, mas podem conter aclives e declives. Alem disso, ha diversos tipos de robots, de mercadorias e de locais para armazenamento.**

Considerando então um conjunto de robots, que transportam cargas entre diferentes lugares, é proposta a seguir uma representação hierárquica do conhecimento deste sistema. Nesta representação, o conhecimento é diferenciado conforme o grau das fórmulas que os representam. Inicia-se a análise a partir de exemplos informais e em seguida é considerada a definição formal de grau de uma fórmula.

4.4.1 CONHECIMENTO DE GRAU 0.

Na proposta considerada, o conhecimento de mais baixo grau hierárquico, ou menos complexo, é o conhecimento que representa as medidas. Tem-se, por exemplo, a temperatura em uma carga, seu peso, sua dimensão, enfim todas as medidas referentes às cargas. Há também as medidas sobre as condições físicas das pistas como a sua declividade, as condições do piso etc. Finalmente, há as medidas sobre cada robot, como sua velocidade máxima, seu rendimento, etc.

Portanto, o conhecimento de mais baixo grau hierárquico são as medidas, que são representadas por constantes e variáveis

$$A_\alpha \in \text{CONST},$$

$$x_\beta \in \text{VAR}$$

ou por fórmulas

$$H_\gamma \in \text{FORM}$$

onde

$$\alpha, \beta, \gamma \in \mathbb{F}.$$

Isto é, o conhecimento de grau zero é representado por fórmulas associadas a símbolos para tipo pertencentes ao conjunto fundamental. O conhecimento de grau zero coincide com o conhecimento de nível zero.

4.4.2 CONHECIMENTO DE GRAU UM.

O primeiro grau de complexidade, acima do conhecimento de grau zero, é caracterizado pelo conhecimento sobre o conhecimento de nível zero. Este conhecimento é o conhecimento de grau um. Um exemplo deste tipo de conhecimento é analisado a seguir.

Considere a proposição,

$$P_{(temp > t)}\ T_{temp}$$

que é verdadeira quando a variável "T_{temp}" é uma medida de temperatura superior a 20 graus. Observe que para a função

$$\lambda X_{temp}\ (P_{(temp > t)}\ X_{temp})$$

determinar se a medida de temperatura é, ou não, superior a 20 graus, deve-se ter um conhecimento que possibilite reconhecer e classificar medidas que representam temperatura, isto é, fórmulas associadas ao símbolo para tipo "temp". Propõe-se que o conhecimento representado pela função indicada possui um grau hierárquico igual a um, pois esta função classifica as medidas e deve ter um grau hierárquico superior ao do conhecimento que representa medidas.

Considere também a proposição,

$$\textbf{TEMPERATURA-CARGA}\ C_{st}\ T_{temp}$$

que é verdadeira quando T_{temp} é a temperatura na carga C_{st}. C_{st} é uma fórmula associada ao símbolo para tipo "st" (string). Observe que a função,

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp})$$

representa o seguinte conhecimento:

(i) identifica se a variável X_{st} representa uma carga,

(ii) identifica se X_{temp} representa uma medida de temperatura e

(iii) diz se a medida de temperatura representada por X_{temp} é a temperatura da carga representada por X_{st}.

Portanto, esta função não só reconhece e classifica conhecimentos de grau zero, como também os relaciona. Propõe-se que o conhecimento representado pela função indicada é de grau 1. Entretanto, observe que

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp}) =^{dt} 6$$

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp}) =^{dp} 4$$

Por outro lado, tem-se que,

$$\lambda X_{temp}\ (P_{(temp \succ t)}\ X_{temp}) =^{dt} 3$$

e

$$\lambda X_{temp}\ (P_{(temp \succ t)}\ X_{temp}) =^{dp} 2$$

Observe também que,

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp}) =^{n} 2$$

e

$$\lambda X_{temp}\ (P_{(temp \succ t)}\ X_{temp}) =^{n} 1$$

Assim, funções que representam conhecimento de grau um, que identifica e relaciona conhecimento de nível zero, possuem diferentes complexidades quanto a descrição e quanto ao nível. Portanto, estas funções representam diferentes conhecimentos sobre fórmulas de um mesmo nível. A função

$$\lambda X_{temp}\ (P_{(temp \succ t)}\ X_{temp})$$

representa apenas o conhecimento que identifica e classifica um outro conhecimento de nível zero. Já a função

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp}),$$

além da identificar e classificar conhecimentos de nível zero, também os relacionam. Neste sentido, as funções representam conhecimentos de mesmo grau, mas com complexidades quanto ao nível e quanto a descrição total e parcial diferentes.

Considere uma outra proposição que relaciona um número maior de diferentes conhecimentos de nível zero. Seja a proposição,

$$\textbf{DECLIVIDADE}\ Lx_{loc}\ Ly_{loc}\ X_{ang}$$

onde

$$loc,\ ang \in \mathbb{F}$$

O símbolo "loc" esta associado às variáveis e constantes que identificam lugares nas pistas utilizadas pelos robots e "ang" esta associado às medidas de inclinação destas pistas. Esta fórmula é verdadeira quando a declividade da pista entre os locais Lx_{loc} e Ly_{loc} é dado pela medida de angulo X_{ang}. Neste

caso, a função

$$\lambda X_{loc}\ \lambda Y_{loc}\ \lambda Z_{ang}\ (\textbf{DECLIVIDADE}\ X_{loc}\ Y_{loc}\ Z_{ang})$$

também representa conhecimento sobre fórmulas de nível zero, a saber sobre medidas de angulo e medidas que identificam locais. Neste sentido, considera-se que esta função representa um conhecimento de grau um. Entretanto, ela é formada a partir de um predicado que possui três argumentos o que a diferencia das funções apresentadas anteriormente. Naqueles casos tem-se funções com um argumento e com dois argumentos. Observa-se também que,

$$\lambda X_{loc}\ \lambda Y_{loc}\ \lambda Z_{ang}\ (\textbf{DECLIVIDADE}\ X_{loc}\ Y_{loc}\ Z_{ang} =^{dt} 9$$

$$\lambda X_{loc}\ \lambda Y_{loc}\ \lambda Z_{ang}\ (\textbf{DECLIVIDADE}\ X_{loc}\ Y_{loc}\ Z_{ang} =^{dp} 6$$

$$\lambda X_{loc}\ \lambda Y_{loc}\ \lambda Z_{ang}(\textbf{DECLIVIDADE}\ X_{loc}Y_{loc}Z_{ang} =^{dp} 3$$

Novamente, tem-se uma função que representa conhecimento de grau 1, mas cujo nível e ordens total e parcial são diferentes das funções anteriores. Neste caso, o nível e as ordens são superiores aos casos anteriores. Isto se deve ao fato de que a função indicada acima, além de identificar e classificar o conhecimeto representado por tres fórmulas de grau zero, ela também os relaciona. Esta relação pode ser uma relação dois a dois, quando um argumento é mantido constante, ou uma relação entre os tres argumentos. Observe que o conhecimento necessário para relacionar tres fórmulas entre si é mais "complexo" que aquele utilizado para relacionar duas fórmulas. Neste sentido, a complexidade da função

$$\lambda X_{loc}\ \lambda Y_{loc}\ \lambda Z_{ang}\ (\textbf{DECLIVIDADE}\ X_{loc}\ Y_{loc}\ Z_{ang})$$

quanto ao nível e quanto as descrições total e parcial é maior que a complexidade das funções

$$\lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMPERATURA-CARGA}\ X_{st}\ X_{temp})$$

$$\lambda X_{temp}\ (P_{(temp)>t)}\ X_{temp}),$$

consideradas anteriormente.

As funções formadas a partir da aplicação de predicados sobre fórmulas de nível zero podem possuir diferentes complexidades quanto ao nível e quanto à descrição. O incremento desta complexidade é obtido com o aumento do número de argumentos dos predicados considerados.

Seja, por exemplo,

$$A_1, A_2, \ldots, A_n$$

onde

$$tipo[A_i] = char, \ \forall\ i,$$

e

$$char \in \mathbb{F}$$

isto é,

$$A_i =^n 0 \quad \forall \quad i.$$

Seja também o conjunto de predicados

$$P_1, P_2, \ldots, P_n$$

onde

$$tipo[P_1] = (char \succ t)$$

$$tipo[P_2] = (char \succ (char \succ t))$$

$$tipo[P_3] = (char\ (char \succ (char \succ t)))$$

$$\ldots$$

$$tipo[P_n] = (char\ (char \succ \ \ldots\ (char \succ t))\ldots)$$

Considere agora as funções,

$$\lambda X_1\ (P_1 X_1)$$

$$\lambda X_1 \lambda X_2\ (P_2 X_1 X_2)$$

$$\ldots\ldots$$

$$\lambda X_1 \ldots \lambda X_n\ (P_n X_1 \ldots X_n)$$

Logo,

$$\lambda X_1 \ldots \lambda X_i\ (P_i X_1 \ldots X_i) =^{dt} 3i$$

$$\lambda X_1 \ldots \lambda X_i\ (P_i X_1 \ldots X_i) =^{dp} 2i,$$

para $1 \leq i \leq n$. Portanto, há conhecimento sobre o conhecimento de nível zero representado em qualquer complexidade quanto ao nível e às descrições total e parcial.

O conhecimento de grau um caracteriza-se como o conhecimento sobre o conhecimento de nível zero. Este é um conhecimento sobre a identificação e relação de medidas do sistema. Tais relações são representadas por funções que possuem a forma geral:

$$\lambda X_1 \ldots \lambda X_n\ (P\ X_1 \ldots X_n)$$

onde P é um predicado e X_i é uma variável tal que

$$tipo[X_i] \in \mathbb{F}$$

e que representa uma medida. De uma maneira mais geral, o conhecimento de grau um é representado por funções

$$\Phi_{\alpha} = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m)$$

onde

$$tipo[X_i] = \beta_i,$$

$$\alpha = \langle \beta_1 \succ \ldots \beta_n \succ \gamma \rangle$$

e

$$\beta_i, \gamma \in \mathbb{F}.$$

Isto é, tem-se uma função cujos argumentos são fórmulas de nível zero. Além disso, como $\gamma \in \mathbb{F}$, então o resultado da aplicação de Φ_{α} sobre os seus argumentos é também uma fórmula de nível zero.

4.4.3. CONHECIMENTO DE GRAU SUPERIOR.

O conhecimento de grau superior é um conhecimento de grau acima de um. Ele versa sobre o conhecimento de nível superior. Isto é, tem-se o conhecimento sobre o conhecimento que relaciona as medidas em diferentes equipamentos. Tem-se conhecimento sobre funções cujo nível é maior que um. Um exemplo deste tipo de conhecimento é considerado a seguir.

Considere inicialmente dois predicados, indicados por

$$\Psi_1 = \lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMP-CARGA-1}\ X_{st}\ X_{temp})$$

e

$$\Psi_2 = \lambda X_{st}\ \lambda X_{temp}\ (\textbf{TEMP-CARGA-2}\ X_{st}\ X_{temp}),$$

que determinam se a temperatura na carga X_{st} é dada por X_{temp}. O predicado Ψ_1 considera medidas de temperatura com precisão até à primeira casa após a vírgula e o predicado Ψ_2 com precisão até à segunda casa. Considere então a função

$$\Phi_{\beta} = \lambda Y_{\alpha}\ \lambda Y_{loc}\ X_{st}\ (\textbf{DET-FUNC}\ Y_{\alpha}\ Y_{loc}\ X_{st})$$

onde

$$\alpha = tipo[\Psi_1] = tipo[\Psi_2]$$

e a função **DET-FUNC** é definida como se segue:

(i)

$$\textbf{DET-FUNC}\ Y_{\alpha}\ Y_{loc}\ X_{st} = \Psi_2$$

se

$$Y_{\alpha} = \Psi_1$$

$$Y_{loc} = L_1$$

$$X_{st} = C_1$$

(ii)

$$\textbf{DET-FUNC}\ Y_{\alpha}\ Y_{loc}\ X_{st} = Y_{\alpha}$$

nos outros casos.

Observe que a função **DET-FUNC** transforma o predicado Ψ_1 para Ψ_2 no caso em que C_1 é a carga considerada pelo sistema, a qual está localizada no local L_1. Nos outros casos em que uma das condições

$$Y_{\alpha} = \Psi_1$$

$$Y_{loc} = L_1$$

$$X_{st} = C_1$$

não é satisfeita, então a função **DET-FUNC** é a identidade na primeira coordenada.

A função Φ_{β}, definida acima, representa um conhecimento sobre os predicados Ψ_1 e Ψ_2. Ela determina qual desses predicados deve ser considerado a partir dos valôres das variáveis Y_{loc} e X_{st}. Como

$$\beta = (\alpha \succ loc \succ st \succ \omega)$$

e

$$\alpha = (st \succ temp \succ t)$$

onde

$$ordem[\alpha] = 2$$

$$ordem[loc] = ordem[st] = 0$$

logo, os argurmentos da função Φ_{β} e os resultados de suas execuções são fórmulas cujo nível máximo é 2. Neste sentido, Φ_{β} representa um conhecimento sobre fórmulas de nível máximo igual a 2. Propõe-se que o conhecimento representado por uma função como Φ_{β} seja um conhecimento de grau superior. Especificamente, propõe-se que Φ_{β} representa um conhecimento de grau 3.

Assim, o conhecimento de grau tres é um conhecimento sobre o conhecimento representado por fórmulas cujo nível máximo é dois. É um conhecimento sobre as funçõe cujo nível é dois. Analogamente, o conhecimento de grau quatro é um conhecimento sobre os conhecimento representados por fórmulas cujo nível máximo é tres. Extendendo este raciocínio, um conhecimento de grau K representam conhecimento sobre fórmulas cujo nível máximo é K - 1.

Finalmente, observe que uma fórmula

$$\xi_{\gamma} = \lambda X_1 \ldots \lambda X_n \ (A \ F_1 \ldots F_m)$$

que representa um conhecimento de grau K esta associada a um símbolos para tipo de qualquer ordem. Isto é,

$$ordem[\Phi_{\alpha}]^t, \ ordem[\Phi_{\alpha}]^p$$

são quaisquer.

4.5 RELACAO DE COMPLEXIDADE QUANTO AO GRAU.

Define-se a seguir, formalmente, a relação de complexidade quanto ao grau. Esta relação considera a complexidade quanto ao grau introduzida informalmente na seção anterior.

4.5.1 DEFINICAO. Grau de uma formula.

Dada uma fórmula Φ, tal que,

$$\Phi_{\beta} = \lambda X_{\alpha_1} \ldots \lambda X_{\alpha_n} \ (A \ F_1 \ldots F_m)$$

$$\beta = \langle \alpha_1 \succ \ldots \succ \alpha_n \succ \gamma \rangle$$

então o grau de Φ_{β}, indicado por $grau[\Phi_{\beta}]$, é dado por,

$$grau[\Phi_{\beta}] = p + 1$$

onde

$$p = max(\ ordem[\gamma], \ ordem[\alpha_i], \ 1 \leq i \leq n \ )$$

No caso em que n=0, isto é, Φ_{β} é uma fórmula neutra, então

$$grau[\Phi_{\beta}] = ordem[\beta]$$

Observe que este é um caso trivial em que a complexidade quanto ao grau coincide com a complexidade quanto ao nível.

***

4.5.2 DEFINICAO. Relacao de complexidade quanto ao grau.

Dada duas fórmulas Φ e Ψ, se

$$grau[\Phi] > grau[\Psi]$$

então Φ é mais complexa que Ψ quanto ao grau de complexidade, o que é indicado por

$$\Phi >^{g} \Psi.$$

Se

$$grau[\Phi] = grau[\Psi] + k$$

indica-se por

$$\Phi =^{g} \Psi + K.$$

No caso em que k=0, então Φ e Ψ são equivalentes quanto ao grau, isto é,

$$\Phi =^{g} \Psi.$$

***

Observe que se uma fórmula Φ é mais complexa que Ψ quanto ao grau, nada se pode dizer sobre as relações de complexidade entre Φ e Ψ quanto ao nível e quanto à descrição. Uma relação entre o grau de uma fórmula Φ_{β} e a ordem de β é estabelecida pela proposição a seguir.

4.5.3 PROPOSICAO.

Dada uma fórmula Φ_{β},

$$\Phi_{\beta} = \lambda X_{\alpha_1} \ldots \lambda X_{\alpha_n} \ (A \ F_1 \ldots F_m)$$

n > 0, então,

$$grau[\Phi_{\beta}] \leq ordem[\beta]$$

DEMONSTRACAO.

A demonstração é imediata. Se

$$\gamma = tipo[\ (A \ F_1 \ldots F_m) \ ]$$

então,

$$\beta = (\ \alpha_1 \succ \ldots \succ \alpha_n \succ \gamma \ )$$

logo,

$$grau[\Phi_{\beta}] = max(\ ordem[\gamma], \ ordem[\alpha_i], \ 1 \leq i \leq n \ ) + 1$$
$$\leq ordem[\beta]$$

***

4.6 CONCLUSAO.

Neste capítulo definiu-se relações de complexidade que traduzem sintaticamente o conceito de "complexidade" de um conhecimento. Elas foram definidas considerando os seguintes elementos:

(i) as ordens dos símbolos para tipo associados às fórmulas do λ-cálculo tipado,

(ii) a estrutura de formação das fórmulas. Esta estrutura foi considerada pelas definições de árvores de descrição total e parcial de uma fórmula.

As tres relações de complexidade definidas neste capítulo traduzem sintaticamente relações de complexidades entre diferentes conhecimentos. Elas ordenam as fórmulas do λ-cálculo tipado, identificando vários conhecimentos representados em um sistema. Considera-se o conhecimento mais "elementar" como, por exemplo, as medidas do sistema, como o conhecimento de grau zero. A partir deste conhecimento, tem-se o conhecimento sobre as medidas, ou seja o meta-conheciemnto [Buchanan, 1984] (grau 1), o conhecimento sobre o conhecimento das medidas, ou seja o meta-meta-conhecimento (grau 2) e assim por diante. As relações de complexidade quanto ao nível e quanto à descrição total e parcial descrevem a estrutura sintática das fórmulas utilizadas na representação do conhecimento.

Estas relações de complexidade são utilizadas, no próximo capítulo, para se estabelecer um conjunto de condições necessárias à dedução de uma fórmula a partir de um programa com critério. Estas relações de "complexidade" estabelecem uma hierarquia entre as fórmulas do λ-cálculo tipado, que será utilizada no estudo da representação hierárquica de conhecimento, no capítulo 6.

CAPITULO 5

CONDICOES NECESSARIAS PARA DEDUCAO DE CONHECIMENTO A PARTIR DE λ-PROGRAMAS.

Neste capitulo sao analisadas as condicoes necessarias para que uma formula Φ tenha uma deducao por derivacao a partir de um programa com criterio $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Nesta analise utiliza-se as relacoes de complexidade entre as formulas do λ-calculo tipado, introduzidas no capitulo anterior. As condicoes necessarias sao descritos em termos destas relacoes de complexidade.

5.1 INTRODUCAO.

Neste capítulo são analisadas as condições necessárias para a dedução de conhecimento a partir de λ-programas. Dada uma fórmula Φ e um λ-programa $\lambda\text{-}P$ são determinadas condições necessárias, que relacionam Φ e $\lambda\text{-}P$, para a existência de uma sequência de derivação por retrocesso com sucesso a partir de

$$\langle \lambda\text{-}P, (\ ), (\Phi) \rangle$$

Isto é, são determinadas condições necessárias para que possa existir um critério $\mathbb{D}$, tal que Φ seja uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Para estabelecer estas condições são utilizadas as relações de complexidade, propostas no capítulo anterior, e as noções de substituição e de λ-conversão entre fórmulas do λ-calculo tipado, introduzidas no capítulo 2.

Inicialmente, na seção 5.2, é demonstrado um conjunto de resultados básicos, que são utilizados posteriormente na demonstração dos teoremas que estabelecem as condições necessárias propriamente ditas.

Na seção 5.3 demonstra-se um conjunto de teoremas que estabelecem as condições necessárias citadas acima.

Finalmente, na seção 5.4 analisa-se as condições necessárias demonstradas na seção 5.3.

5.2 RESULTADOS BASICOS.

A seguir é demonstrado um conjunto de resultados a partir das noções de relações de complexidade entre fórmulas do λ-calculo tipado, de substituição e de operações de λ-conversão. Estes resultados são utilizados na proxima seção, na demonstração de teoremas que estabelecem as condições necessárias para que uma fórmula Φ seja uma dedução por derivação a partir de um programa com critério. Antes de iniciar a análise considere a observação a seguir.

OBSERVACAO: Dadas as fórmulas H_α, F_β e a variável X_α, ao se escrever

$$S^X_H F_\beta$$

está implícito que H_α é livre para X_α em F_β.

***

Define-se a seguir o comprimento de uma fórmula. Esta definição baseia-se no conceito correntemente utilizado em lógica matemática [Shoenfield, 1967].

5.2.1 DEFINICAO. Comprimento de uma formula. [Shoenfield, 1967]

O comprimento de uma fórmula Φ, indicado por $comp[\Phi]$, é definido recursivamente a seguir.

(i) Se $\Phi \in VAR \cup CONST$, então

$$comp[\Phi] = 1.$$

(ii) Se $\Phi = (AB)$, então

$$comp[\Phi] = comp[A] + comp[B]$$

(iii) Se $\Phi = \lambda X\ H$, então

$$comp[\Phi] = 1 + comp[H]$$

***

LEMA 5.1

Seja A_α uma fórmula, então

$$ordem[A_\alpha]^t \geq ordem[\alpha]$$

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de A_α.

(i) Se $comp[A_\alpha] = 1$, então $A_\alpha \in VAR_\alpha \cup CONST_\alpha$. Por definição,

$$ordem[A_\alpha]^t = prof[\ arv[\alpha]\ ]$$

e pela proposição 4.3.3,

$$prof[\ arv[\alpha]\ ] = ordem[\alpha]$$

logo,

$$ordem[A_\alpha]^t = ordem[\alpha]$$

Se $comp[A_\alpha] = n$, então pode ocorrer o seguinte:

$$A_\alpha = B_\tau C_\gamma$$

ou

$$A_\alpha = \lambda X_\beta H_\sigma$$

Estes casos são analisados a seguir.

(ii) Se $A_\alpha = B_\tau C_\gamma$, $\tau = \langle \gamma \succ \omega$, então

$$comp[B_\tau] < n$$

$$comp[C_\gamma] < n$$

e por hipótese de indução

$$ordem[B_\tau]^t \geq ordem[\tau]$$

$$ordem[C_\gamma]^t \geq ordem[\gamma]$$

Por definição,

$$ordem[A_\alpha]^t = 1 + max\ (\ ordem[B_\tau]^t,\ ordem[C_\gamma]^t\ )$$

logo,

$$ordem[A_\alpha]^t > ordem[B_\tau]^t$$

Como por hipótese de indução,

$$ordem[B_\tau]^t \geq ordem[\tau]$$

e como $\tau = \langle \gamma \succ \omega$,

$$ordem[\tau] > ordem[\alpha]$$

portanto,

$$ordem[A_\alpha]^t \geq ordem[\alpha]$$

(iii) Se $A_\alpha = \lambda X_\beta H_\sigma$, $\alpha = \langle \beta \succ \sigma$, então

$$comp[H_\sigma] < n$$

e

$$ordem[H_\sigma]^t \geq ordem[\sigma]$$

Por definição,

$$ordem[A_\alpha]^t = 1 + max (ordem[X_\beta]^t, ordem[H_\sigma]^t)$$

Com a hipótese de indução, tem-se

$$ordem[A_\alpha]^t \geq 1 + max (ordem[\beta], ordem[\sigma])$$

Como $\alpha = \langle \beta \succ \sigma$, então

$$ordem[\alpha] = 1 + max (ordem[\beta], ordem[\sigma])$$

Portanto,

$$ordem[A_\alpha]^t \geq ordem[\alpha]$$

COD.

***

O lema 5.1 não é válido quando se considera ordem parcial. O exemplo a seguir ilustra este fato.

EXEMPLO 5.1

Seja $A_{(\iota \succ \iota)} \in$ CONST, então,

$$ordem[A_{(\iota \succ \iota)}]^P = \emptyset$$

e

$$ordem[(\iota \succ \iota)] = 1$$

logo,

$$ordem[A_{(\iota \succ \iota)}]^P = < ordem[(\iota \succ \iota)]$$

***

LEMA 5.2

Considere F_β uma fórmula, X_α uma variável e H_α uma variável ou constante, então

$$ordem[F] \leq ordem[S^X_H F]$$

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de F_β.

(i) Se $F_\beta \in VAR_\beta \cup CONST_\beta$, tem-se dois casos:

(i.a) $F_\beta = X_\alpha$, isto é, $\beta = \alpha$, logo

$$S_H^X F = H_\alpha$$

Como $F_\beta \in VAR$, $H_\alpha \in VAR \cup CONST$, então, considerando ordem total,

$$ordem[F_\beta]^t = ordem[\beta]$$

$$ordem[H_\alpha]^t = ordem[\alpha]$$

Como $\beta = \alpha$, logo

$$ordem[F_\beta]^t = ordem[H_\alpha]^t$$

Sabendo que $S_H^X F = H_\beta$, então

$$ordem[F_\beta]^t = ordem[\, S_H^X F \,]^t$$

Considerando ordens parciais, tem-se

$$ordem[F_\beta]^p = \emptyset = ordem[H_\alpha]^p$$

logo

$$ordem[F_\beta] = ordem[S_H^X F]$$

(i.b) Se $F \neq X$, então,

$$S_H^X F = F_\beta$$

logo

$$ordem[F_\beta] = ordem[S_H^X F]$$

(ii) Se $F_\beta = A_\tau B_\gamma$, $\tau = \langle \gamma \succ \beta$, por definição

$$ordem[F_\beta] = 1 + \max (ordem[A_\tau], ordem[B_\gamma])$$

e

$$S_H^X F = (S_H^X A)(S_H^X B)$$

logo,

$$ordem[S_H^X F] = 1 + \max (ordem[S_H^X A], ordem[S_H^X B])$$

Utilizando a hipótese de indução

$$ordem[S_H^X F] \geq 1 + \max (ordem[A_\tau], ordem[B_\gamma])$$

Como $F_\beta = A_\tau B_\gamma$, tem-se que

$$ordem[F_\beta] = 1 + max\ (\ ordem[A_\tau],\ ordem[B_\gamma]\ )$$

logo,

$$ordem[S_H^X F] \geq ordem[F_\beta]$$

(iii) Se $F_\beta = \lambda Y_\sigma\ C_\nu$, $\beta = (\sigma \succ \nu)$, tem-se dois casos:

(iii.a) $Y_\sigma \neq X_\alpha$, então,

$$S_H^X F = \lambda Y\ (S_H^X C)$$

Como

$$ordem[F_\beta] = 1 + max\ (\ ordem[Y_\sigma],\ ordem[C_\nu]\ )$$

e

$$ordem[S_H^X F] = 1 + max(\ ordem[Y_\sigma],\ ordem[S_H^X C]\ )$$

utilizando a hipótese de indução,

$$ordem[S_H^X F] \geq 1 + max\ (\ ordem[Y_\sigma],\ ordem[C_\nu]\ )$$

logo,

$$ordem[S_H^X F] \geq ordem[F_\beta]$$

(iii.b) $Y_\sigma = X_\alpha$, então,

$$S_H^X F = F_\beta$$

logo,

$$ordem[S_H^X F] = ordem[F_\beta]$$

CQD

***

LEMA 5.3.

Sejam as fórmulas F_β e

$$H_\alpha = (A_\tau B_\gamma),$$

onde $\tau = (\gamma \succ \omega)$, e a variável X_α, então

$$ordem[F_\beta] \leq ordem[S_H^X F]$$

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de F_β.

(i) Se $F_\beta \in VAR_\beta \cup CONST_\beta$, tem-se dois casos:

(i.a) $F_\beta = X_\alpha$, isto é, $\beta = \alpha$, logo

$$S_H^X F = H_\alpha$$

Por definição,

$$ordem[H_\alpha] = 1 + max (ordem[A_\tau], ordem[B_\gamma])$$

logo

$$ordem[H_\alpha] > ordem[A_\tau]$$

Considerando a ordem total e o Lema 5.1,

$$ordem[A_\tau]^t \geq ordem[\tau]$$

logo,

$$ordem[H_\alpha]^t > ordem[\tau]$$

Como $\tau = \langle \gamma \succ \alpha \rangle$, então

$$ordem[\tau] = 1 + max (ordem[\gamma], ordem[\alpha])$$

logo

$$ordem[\tau] > ordem[\alpha]$$

Tem-se também que,

$$ordem[F_\beta]^t = ordem[\beta]$$

Como $\alpha = \beta$ tem-se

$$ordem[F_\beta]^t = ordem[\alpha]$$

Portanto,

$$ordem[F_\beta]^t < ordem[H_\alpha]^t$$

Como $S_H^X F = H_\alpha$, então

$$ordem[F_\beta]^t \leq ordem[S_H^X F]^t$$

Considerando agora ordem parcial, como $F_\beta \in VAR \cup CONST$ tem-se

$$ordem[F_\beta]^P = \emptyset.$$

logo

$$ordem[F_\beta]^P \leq ordem[S_H^X F]^P$$

Conclui-se então que para a ordem total e parcial

$$ordem[F_\beta] \leq ordem[S_H^X F]$$

(i.b) Se $F \neq X$, então,

$$S_H^X F = F_\beta$$

logo,

$$\text{ordem}[F_\beta] = \text{ordem}[S_H^X F]$$

(ii) Se $F_\beta = A_\tau B_\gamma$, $\tau = (\gamma \succ \beta)$, então,

$$\text{ordem}[F_\beta] = 1 + \max (\text{ordem}[A_\tau], \text{ordem}[B_\gamma])$$

Por definição,

$$S_H^X F = (S_H^X A)(S_H^X B)$$

logo,

$$\text{ordem}[S_H^X F] = 1 + \max (\text{ordem}[S_H^X A], \text{ordem}[S_H^X B])$$

Por hipótese de indução,

$$\text{ordem}[S_H^X F] \geq 1 + \max (\text{ordem}[A_\tau], \text{ordem}[B_\gamma])$$

logo

$$\text{ordem}[S_H^X F] \geq \text{ordem}[F_\beta]$$

(iii) Se

$$F_\beta = \lambda Y_\sigma \, C_\nu,$$

onde $\beta = (\sigma \succ \nu)$, tem-se dois casos:

(iii.a) $Y_\sigma \neq X_\alpha$, então,

$$S_H^X F = \lambda Y \, (S_H^X C)$$

Por definição,

$$\text{ordem}[F_\beta] = 1 + \max (\text{ordem}[Y_\sigma], \text{ordem}[C_\nu])$$

e

$$\text{ordem}[S_H^X F] = 1 + \max (\text{ordem}[Y_\sigma], \text{ordem}[S_H^X C])$$

Logo, por hipótese de indução,

$$\text{ordem}[S_H^X F] \geq 1 + \max (\text{ordem}[Y_\sigma], \text{ordem}[C_\nu])$$

portanto,

$$\text{ordem}[S_H^X F] \geq \text{ordem}[F_\beta]$$

(iii.b) $Y_\sigma = X_\alpha$, então,

$$S_H^X F = F_\beta$$

logo,

$$\text{ordem}[S_H^X F] = \text{ordem}[F_\beta]$$

CQD

***

LEMA 5.4.

Sejam as fórmulas F_β e

$$H_\alpha = \lambda Y_\tau \; C_\sigma,$$

onde $\alpha = \langle \tau \succ \sigma$ e a variável X_α, então,

$$\text{ordem}[F] \leq \text{ordem}[S^X_H \; F]$$

DEMONSTRACAO.

A demonstração é análoga à demonstração do lema anterior.

***

5.2.2 PROPOSICAO.

Sejam as fórmulas H_α, F_β e a variável X_α, então

$$\text{ordem}[F_\beta] \leq \text{ordem}[S^X_H \; F_\beta]$$

$$\text{ordem}[H_\alpha] \leq \text{ordem}[S^X_H \; F_\beta]$$

DEMONSTRACAO.

A demonstração de

$$\text{ordem}[F_\beta] \leq \text{ordem}[S^X_H \; F_\beta]$$

é feita utilizando os lemas 5.1, 5.2, 5.3 e 5.4. A demonstração de

$$\text{ordem}[H_\alpha] \leq \text{ordem}[S^X_H \; F_\beta]$$

é realizada utilizando resultados análogos aos dos lemas anteriores.

CQD.

***

5.2.3 COROLARIO.

Sejam as fórmulas F, H e a substituição θ tais que

$$H = F\theta$$

então

$$F \leq^{dt} H$$

$$F \leq^{dp} H$$

DEMONSTRACAO.

A demonstração decorre imediatamente da proposição anterior e da definição de substituição.

***

5.2.4 PROPOSICAO.

Sejam as fórmulas

$$H_\alpha = \lambda X_1 \ldots \lambda X_n \ \Phi$$

$$F_\alpha = (\lambda X_{n+1} \lambda X_1 \ldots \lambda X_n \ X_{n+1}) \ \Phi$$

onde H_α é uma β-redução de F_α, isto é, Φ é livre para X_{n+1} em

$$\lambda X_1 \ldots \lambda X_n \ X_{n+1}$$

Então,

$$ordem[F_\alpha] \leq 2 + ordem[H_\alpha]$$

DEMONSTRACAO.

A demonstração é feita considerando-se apenas a ordem total. No caso da ordem parcial tem-se um raciocínio análogo.

Considere inicialmente que n=0. Neste caso tem-se que

$$H_\alpha = \Phi$$

e

$$F_\alpha = (\lambda X_{n+1} \ X_{n+1}) \ \Phi$$

Como F_α é uma fórmula bem formada,

$$tipo[\Phi] = tipo[X_{n+1}]$$

Pelo lema 5.1,

$$ordem[H_\alpha]^t \geq ordem[\alpha]$$

logo,

$$ordem[H_\alpha]^t \geq ordem[\ tipo[X_{n+1}]\ ]$$

isto é,

$$ordem[H_\alpha]^t \geq ordem[X_{n+1}]^t$$

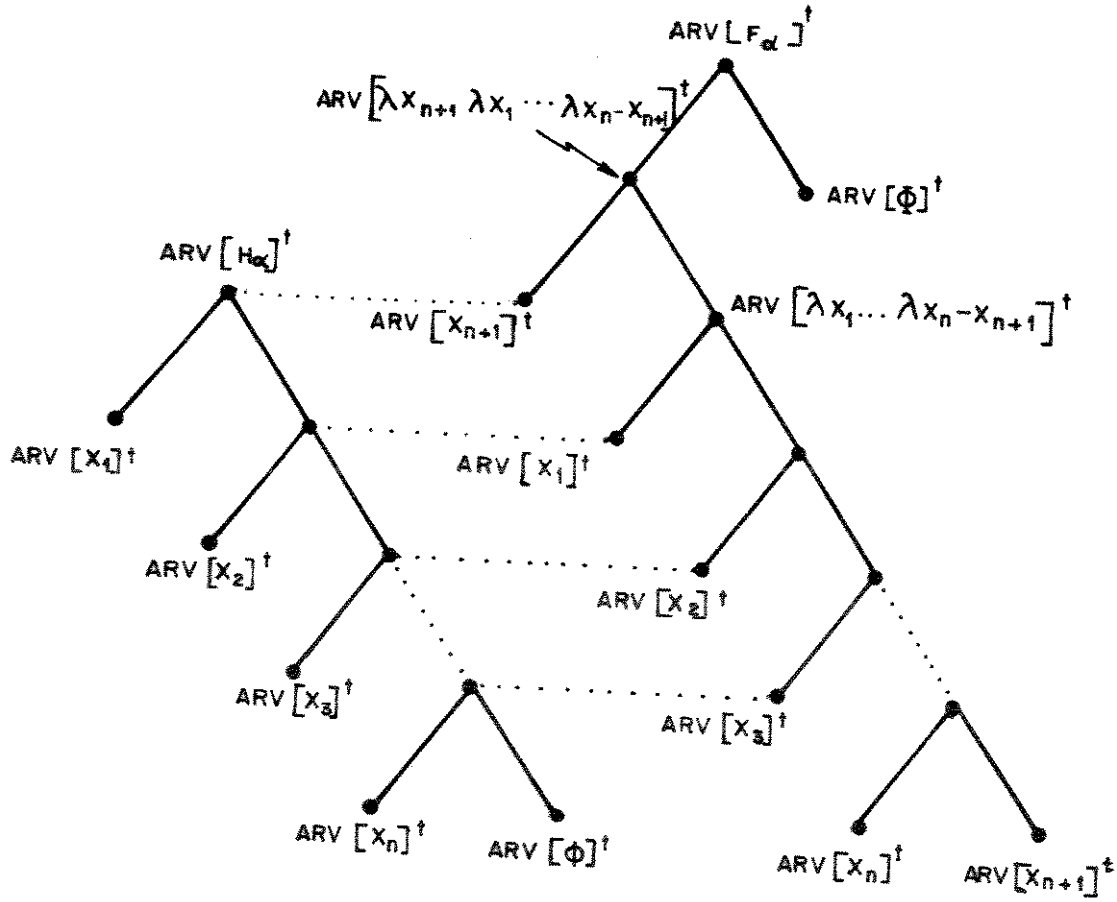
Como

$$F_\alpha = (\lambda X_{n+1}\ X_{n+1})\ \Phi$$

então

$$ordem[F_\alpha]^t \leq 2 + ordem[H_\alpha]^t$$

No caso em que n>0, considere a figura 5.1, que apresenta as árvores de descrição das fórmulas F_α e H_α.



Arvores de descricao das formulas F e H.
FIGURA 5.1

Considere

$$K = ordem[\Phi]^t$$

A demonstração divide-se em dois casos.

(i) Inicialmente, supõe-se que para algum i, $1 \leq i \leq n$,

$$ordem[H_\alpha]^t = i + ordem[X_i]^t$$

isto é,

$$i + ordem[X_i]^t \geq n + K$$

Logo, como $i \leq n$,

$$ordem[X_i]^t \geq K$$

Como $X_{n+1} \in VAR$, então por definição de árvore de descrição total,

$$ordem[X_{n+1}]^t = ordem[\ tipo[X_{n+1}]\ ]$$

Como

$$tipo[X_{n+1}] = tipo[\Phi]$$

então,

$$ordem[X_{n+1}]^t = ordem[\ tipo[\Phi]\ ]$$

Pelo lema 5.1,

$$ordem[\ tipo[\Phi]\ ] \leq ordem[\Phi]^t$$

logo,

$$ordem[X_{n+1}]^t \leq ordem[\Phi]^t$$

isto é,

$$ordem[X_{n+1}]^t \leq K$$

Assim, a partir da árvore de descrição de F_α e das equações

$$ordem[X_i]^t \geq K$$

e

$$ordem[X_{n+1}]^t \leq K$$

tem-se que

$$ordem[F_\alpha]^t = 2 + (i + ordem[X_i]^t)$$

isto é,

$$ordem[F_\alpha]^t = 2 + ordem[H_\alpha]^t$$

(ii) Neste caso, supõe-se que a ordem total de H_α é dada por,

$$ordem[H_\alpha]^t = n + K$$

isto é,

$$ordem[H_\alpha]^t = n + ordem[\Phi]^t.$$

Há várias possibilidades para o cálculo do valor de $ordem[F_\alpha]^t$.

(ii.a) A ordem total de F_α é dada por,

$$ordem[F_\alpha]^t = 1 + ordem[\ ]^t$$

isto é,

$$ordem[F_\alpha]^t = 1 + K$$

logo

$$ordem[F_\alpha]^t \leq n + K$$

Como

$$ordem[H_\alpha]^t = n + K$$

então

$$ordem[F_\alpha]^t \leq ordem[H_\alpha]^t$$

(ii.b) A ordem total de F_α é dada por,

$$ordem[F_\alpha]^t = 2 + n + ordem[X_{n+1}]^t$$

logo,

$$ordem[F_\alpha]^t \leq 2 + n + K$$

Como

$$ordem[H_\alpha] = n + K$$

então

$$ordem[F_\alpha]^t \leq 2 + ordem[H_\alpha]$$

(ii.c) A ordem total de F_α é dada por

$$ordem[F_\alpha]^t = j + ordem[X_j]^t + 2$$

para algum j, $1 \leq j \leq n$. Como,

$$ordem[H_\alpha]^t = n + K$$

então

$$j + ordem[X_j]^t \leq n + K$$

Assim,

$$ordem[F_\alpha]^t \leq n + K + 2$$

logo,

$$ordem[F_\alpha]^t \leq 2 + ordem[H_\alpha]^t$$

CQD.

***

5.2.5 PROPOSICAO.

Sejam as fórmulas

$$F_\alpha = (\lambda X_{n+1}\lambda X_1 \ldots \lambda X_n\ \Psi)\ \Phi$$

$$H_\alpha = \lambda X_1 \ldots \lambda X_n\ (S_\Phi^{X_{n+1}}\ \Psi)$$

onde H_α é uma β-redução de F_α isto é, Φ é livre para X_{n+1} em

$$\lambda X_1 \ldots \lambda X_n \Psi,$$

então,

$$\text{ordem}[F_\alpha] \leq 2 + \text{ordem}[H_\alpha]$$

DEMONSTRACAO.

Conforme a proposição 5.2.2,

$$\text{ordem}[\Psi] \leq \text{ordem}[S_\Phi^{X_{n+1}}\Psi]$$

e

$$\text{ordem}[\Phi] \leq \text{ordem}[S_\Phi^{X_{n+1}}\Psi]$$

Utilizando estes resultados, a demonstração é análoga a da proposição anterior.

***

5.2.6 COROLARIO.

Sejam as fórmulas, F, H tais que H é obtida a partir de F por uma β-redução, isto é,

$$F \blacktriangleright_\beta H$$

Então,

$$\text{ordem}[F] \leq \text{ordem}[H] + 2$$

DEMONSTRACAO.

Imediata a partir da proposição 5.2.5, da definição de ordem de uma fórmula e de β-redução.

***

5.2.7 PROPOSICAO.

Sejam as fórmulas F, H tais que H é obtida a partir de F pela aplicação de K β-reduções. Então,

$$ordem[F] \leq ordem[H] + 2K$$

DEMONSTRACAO.

A demonstração é feita por indução no número de β-reduções utilizadas para obter H a partir de F.

***

5.2.8 PROPOSICAO.

Sejam as fórmulas F, H onde H é obtida a partir de F por uma η-redução, isto é,

$$F \blacktriangleright_{\eta} H$$

Então,

$$ordem[F] \geq ordem[H] + 2$$

DEMONSTRACAO.

Como é possível aplicar uma η-redução em F, então podem ocorrer dois casos:

(i) F = $\lambda X(HX)$ tal que X não ocorre livre em H. A partir da árvore de descrição apresentada na figura 5.2, verifica-se o seguinte:

(i.a) Se

$$ordem[H] \geq ordem[X]$$

então

$$ordem[F] = ordem[H] + 2$$

(i.b) Se

$$ordem[X] > ordem[H]$$

então

$$ordem[F] = ordem[X] + 2$$

logo,

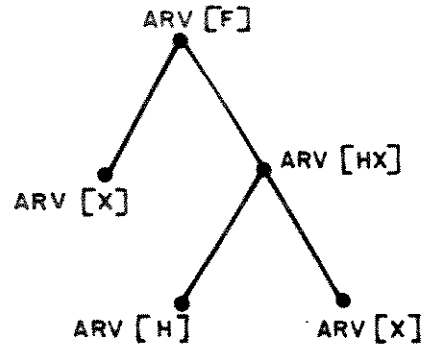
$$ordem[F] > ordem[H] + 2$$

(ii) F possui uma subfórmula do tipo $\lambda X(GX)$ tal que X não ocorre livre em G. Quando se aplica uma η-redução em F, no lugar da subfórmula $\lambda X(GX)$ tem-se a subfórmula G. Utilizando o resultado do item (i) tem-se

$$ordem[\ \lambda X(GX)\ ] \geq ordem[\ G\ ] + 2$$

Logo, a profundidade da árvore de descrição de H diminui no mínimo 2 unidades. Portanto,

$$ordem[F] \geq ordem[H] + 2$$



Arvore de descricao da formula λx(gx)

FIGURA 5.2

***

5.2.9 PROPOSICAO.

Sejam as fórmulas F, H onde H é obtida a partir de F pela aplicação de P η-reduções. Então,

$$ordem[F] \geq ordem[H] + 2P$$

DEMONSTRACAO.

A demonstração é feita por indução no número de η-reduções utilizadas para obter H a partir de F.

***

Demonstra-se a seguir o Lema 5.5, cujo resultado é utilizado na demonstração da proposição 5.2.10.

LEMA 5.5

Sejam as fórmulas P, H, G e as variáveis Y e X. Considere que Y não ocorre livre em H e que

$$PY = S^{X}_{H}G$$

então,

$$G = QY$$

para alguma fórmula Q.

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de G.

(i) Se G $\in$ VAR $\cup$ CONST tem-se dois casos:

(i.a) G = X, então

$$S^X_H G = H$$

logo

$$H = PY$$

o que é absurdo pois Y não ocorre livre em H. Logo não se pode ter G = X.

(i.b) G $\neq$ X, então

$$S^X_H G = G$$

logo

$$G = PY$$

(ii) G = AB, logo

$$S^X_H G = (S^X_H A)\ (S^X_H B)$$

e

$$PY = (S^X_H A)\ (S^X_H B)$$

Então

$$P = S^X_H A$$

e

$$Y = S^X_H B$$

Como Y não ocorre livre em H e

$$Y = S^X_H B$$

então B = Y. Logo,

$$G = AY$$

(iii) G = λW F, tem-se dois casos:

(iii.a) W = X, logo

$$S^X_H G = G$$

e

$$G = PY$$

(iii.b) W $\neq$ X, logo

$$S^X_H G = \lambda W\ S^X_H F$$

e

$$PY = \lambda W\ S^X_H F$$

o que é impossível pois PY é a justaposição de duas fórmulas (do tipo AB) o que não ocorre com

$$\lambda W\ S^X_H F$$

Logo não se pode ter W ≠ X.

CQD.

***

5.2.10 PROPOSICAO.

Sejam as fórmulas F_β, H_α e a variável X_α onde F_β e H_α não possuem η-redex. Então, $S^X_H F$ não possui η-redex.

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de F.

(i) Se F ∈ VAR ∪ CONST tem-se dois casos.

(i.a) X = F, logo $S^X_H F = H$ ou

(i.b) X ≠ F, logo $S^X_H F = F$.

Nos dois casos, tem-se então que $S^X_H F$ não possui η-redex.

(ii) Se F = AB, por definição,

$$S^X_H F = (S^X_H A)(S^X_H B)$$

Utilizando a hipótese de indução, $S^X_H A$ e $S^X_H B$ não possuem η-redex. Suponha por contradição que $S^X_H F$ possui η-redex, logo $S^X_H F$ possui uma subfórmula Ψ do tipo

$$\Psi = (\lambda X\ (PX)).$$

onde X não ocorre livre em P. Como $S^X_H A$ e $S^X_H B$ não possuem η-redex, então Ψ não é uma subfórmula de $S^X_H A$ e nem de $S^X_H B$. Logo,

$$S^X_H F = \Psi,$$

e

$$F = \lambda X(PX)$$

o que é um absurdo pois, por hipótese, F = AB.

(ii) Se

$$F = \lambda Y\ G$$

tem-se os seguintes casos:

(ii.a) $Y = X$, então

$$S^X_H F = F,$$

logo $S^X_H F$ não possui η-redex.

(ii.b) $Y \neq X$, então

$$S^X_H F = \lambda Y\ (S^X_H G).$$

Por hipótese de indução, $S^X_H G$ não possui η-redex. Deve-se então demonstrar que

$$S^X_H F = \lambda Y\ (S^X_H G)$$

não possui η-redex.

Suponha por contradição que

$$\lambda Y\ (S^X_H G)$$

possui η-redex. Como $S^X_H G$ não possui η-redex então

$$\lambda Y\ S^X_H G = \lambda Y\ (PY)$$

onde Y não ocorre livre em P, isto é,

$$S^X_H G = PY$$

Como H é livre para X em F e

$$S^X_H F = \lambda Y\ (S^X_H G)$$

então a variável Y não ocorre livre em H pois Y não ocorre livre em

$$\lambda Y\ (S^X_H G).$$

Tem-se então que,

$$PY = S^X_H G$$

Y não ocorre livre em H

Logo, utilizando o lema 5.5

$$G = QY$$

isto é,

$$S^X_H G = S^X_H QY$$

Como

$$PY = S^X_H G$$

então,

$$PY = S^X_H QY$$

e

$$P = S^X_H Q$$

Como Y não ocorre livre em P e $Y \neq X$, então Y não ocorre livre em Q.

Por hipótese

$$F = \lambda Y\ G$$

logo,

$$F = \lambda Y\ (QY)$$

onde Y não ocorre livre em Q. Isto é, F é um η-redex, o que é uma contradição.

CQD.

***

Tem-se portanto que se F, H não possuem η-redex, então $S^X_H F$ não possui η-redex. Entretanto, pode ocorrer que $S^X_H F$ possua β-redex. Além disso, se Q é tal que

$$S^X_H F \blacktriangleright_\beta Q$$

então Q pode conter algum η-redex. Veja o exemplo a seguir.

EXEMPLO 5.2

Considere as fórmulas a seguir,

$$F = \lambda Y\ (F(ZY))$$

$$H = \lambda X\ X$$

tal que Y não ocorre livre em F e H é livre para Y em F. Observe que estas fórmulas não possuem η-redex. Considere então,

$$S^Z_H F = \lambda Y\ (F((\lambda X\ X)Y))$$

que não possui η-redex, mas posui β-redex. Isto é,

$$S^Z_H F \blacktriangleright_\beta \lambda Y\ (FY)$$

Mas como Y não ocorre livre em F, então $\lambda Y\ (FY)$ é um η-redex. Isto é, $S^Z_H F$ não possui η-redex, mas após uma β-reduçãO aparece um η-redex.

***

A próxima proposição estabelece condições para que dadas duas fórmulas F, H e uma variável X, então $S^X_H F$ já esteja na forma normal, isto é, não possua nem β-redex e nem η-redex.

5.2.11 PROPOSICAO.

Sejam as fórmulas F_β, H_α e a variável X_α tais que

(i) F, H estão na forma normal,

(ii) H é uma fórmula neutra.

Então $S^X_H F$ está na forma normal.

DEMONSTRACAO.

Como F, H são fórmula normais, elas não possuem η-redex. Logo, pela proposição 5.2.10,

$$S^X_H F$$

não possui η-redex. Falta demonstrar que

$$S^X_H F$$

não possui β-redex. Esta demonstração é feita por indução no comprimento de F.

(i) Se F $\in$ VAR $\cup$ CONST então o resultado é óbvio.

(ii) Se F = AB, tem-se tres casos:

(ii.a) A $\in$ CONST $\cup$ VAR. Por definição,

$$S^X_H F = (S^X_H A)\ (S^X_H B)$$

Como A $\in$ VAR $\cup$ CONST tem-se dois casos.

(ii.a.1) X = A, então

$$S^X_H A = H$$

Como H é uma fórmula neutra, então H é da forma,

$$H = D\ C_1 \ldots C_n$$

onde D $\in$ VAR $\cup$ CONST. Portanto,

$$S^X_H A = D\ C_1 \ldots C_n,$$

logo

$$S^X_H F = (D\ C_1 \ldots C_n)\ (S^X_H B)$$

Como H é livre para X em F, então H é livre para X em B. Utilizando a indução,

$$S^X_H B$$

não possui β-redex. Como D $\in$ VAR $\cup$ CONST então

$$S^X_H F$$

não possui β-redex.

(ii.a.2) X $\neq$ A, então

$$S^X_H A = A$$

logo

$$S^X_H F = A\ (S^X_H B)$$

Como

$$S^X_H B$$

não possui β-redex, então

$$S^X_H F$$

também não possui β-redex pois

$$A \in \text{VAR} \cup \text{CONST}.$$

(ii.b) A = QS, logo,

$$F = QSB$$

e

$$S^X_H F = (S^X_H Q)\ (S^X_H S)\ (S^X_H B)$$

Pela hipótese de indução, as fórmulas

$$(S^X_H Q),\ (S^X_H S),\ (S^X_H B)$$

não possuem β-redex. Utilizando o resultado apresentado no item anterior conclui-se que

$$S^X_H F$$

não possui β-redex.

(ii.c) A = λY C, logo,

$$F = (\lambda Y\ C)B$$

então F é um β-redex, o que é absurdo pois F é uma fórmula normal.

(iii) Se F = λY R, tem-se novamente dois casos.

(iii.a) $Y = X$, então

$$S^X_H F = F,$$

logo $S^X_H F$ não possui β-redex.

(iii.b) $Y \neq X$, então

$$S^X_H F = \lambda Y\ (S^X_H R).$$

Como H é livre para X em F, então H é livre para X em R. Por hipótese de indução, $S^X_H R$ não possui β-redex. Logo $S^X_H F$ não possui β-redex.

CQD

***

5.2.12 PROPOSICAO.

Sejam as fórmulas,

$$F_\beta = \lambda X_\alpha\ \lambda Y_1 \ldots \lambda Y_n (A\ F_1 \ldots F_m)$$

$$H_\alpha = B\ C_1 \ldots C_s$$

onde

(i) H_α é uma fórmula neutra,

(ii) F, H são fórmulas normais e

(iii) H_α é livre para X_α em $\lambda Y_1 \ldots \lambda Y_n (A\ F_1 \ldots F_m)$.

Então a forma normal de FH pode ser obtida por uma única β-reduçãO. Isto é, a forma normal de FH é uma fórmula G tal que

$$FH \blacktriangleright_\beta G$$

DEMONSTRACAO.

Seja

$$P = \lambda Y_1 \ldots \lambda Y_n (A\ F_1 \ldots F_m)$$

então,

$$F = \lambda X_\alpha\ P$$

Tem-se que P, H são fórmulas normais, H é uma fórmula neutra e H é livre para X em P, logo, pela proposiçãO 5.2.11, $S^X_H P$ é uma fórmula normal. Finalmente,

$$FH = (\lambda X_\alpha\ P)H \blacktriangleright_\beta S^X_H P$$

CQD

***

5.2.13 COROLARIO.

Sejam as fórmulas,

$$F_\beta = \lambda X_\alpha \lambda Y_1 \ldots \lambda Y_n (A\ F_1 \ldots F_m)$$

$$H_\alpha = B\ C_1 \ldots C_s$$

onde

(i) H_α é uma fórmula neutra,

(ii) F_β, H_α são fórmulas normais e

(iii) H_α é livre para X_α em $\lambda Y_1 \ldots \lambda Y_n (A\ F_1 \ldots F_m)$.

Se P é a forma normal de FH, então

$$\text{ordem}[P] \leq \text{ordem}[FH] + 2$$

DEMONSTRACAO.

Pela proposição 5.2.12, a forma normal de FH obtida por apenas uma única β-redução. Isto é,

$$FH \blacktriangleright_\beta G$$

Logo, pelo corolário 5.2.6,

$$\text{ordem}[G] \leq \text{ordem}[FH] + 2$$

Como G e P são fórmulas normais associadas a uma mesma fórmula, elas se distinguem apenas pela renomeação de variáveis (teorema da unicidade da forma normal). Logo,

$$\text{ordem}[P] = \text{ordem}[G]$$

e então,

$$\text{ordem}[P] \leq \text{ordem}[FH] + 2$$

CQD.

***

5.2.14 COROLARIO.

Sejam as fórmulas,

$$F = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m)$$

$$H^i = B^i\ C^i_1 \ldots C^i_s,\ 1 \leq i \leq n$$

onde para $1 \leq i \leq n$,

(i) H^i é uma fórmula neutra,

(ii) $tipo[H^i] = tipo[X_i]$,

(iii) F, H_i são fórmulas normais

(iv) H_i é livre para X_i em $\lambda X_{i+1} \ldots \lambda X_n (A\ F_1 \ldots F_m)$.

Se P é a forma normal de $FH_1 \ldots H_n$, então

$$ordem[P] \leq ordem[FH_1 \ldots H_n] + 2n$$

DEMONSTRACAO.

É imediata a partir do corolário 5.2.7.

***

5.2.15 COROLARIO.

Sejam as fórmulas,

$$F = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m)$$

$$H^i = B^i\ C^i_1 \ldots C^i_s,\ 1 \leq i \leq n$$

onde para $1 \leq i \leq n$,

(i) H^i é uma fórmula neutra,

(ii) $tipo[H^i] = tipo[X_i]$,

(iii) F, H_i são fórmulas normais

(iv) H_i é livre para X_i em $\lambda X_{i+1} \ldots \lambda X_n (A\ F_1 \ldots F_m)$,

(v) $ordem[H_i] \leq ordem[F]$.

Se P é a forma normal de $FH_1 \ldots H_n$, então

$$ordem[P] \leq ordem[F] + 3n$$

DEMONSTRACAO.

Tem-se que,

$$ordem[F\ H_1 \ldots H_n] = 1 + max(ordem[F\ H_1 \ldots H_{n-1}],\ ordem[H_n])$$

$$= 1 + max((1 + max(ordem[F\ H_1 \ldots H_{n-2}],\ ordem[H_{n-1}]),\ ordem[H_n])$$

Seguindo o desenvolvimento desta igualdade e considerando que

$$ordem[H_i] \leq ordem[F]\ \forall\ i,$$

então

$$ordem[F\ H_1 \ldots H_n] = ordem[F] + n$$

Pelo corolário 5.2.14,

$$ordem[P] \leq ordem[F\ H_1 \ldots H_n] + 2n$$

logo,

$$ordem[P] \leq ordem[F] + 3n$$

CQD.

***

5.3 CONDICOES NECESSARIAS.

A seguir, os resultados básicos introduzidos na seção anterior são utilizados para estabelecer as condições necessárias para a dedução de conhecimento a partir de um λ-programa. Apresenta-se inicialmente um teorema que determina as condições necessárias para que uma fórmula Φ seja uma derivação em uma outra fórmula Ψ. Em seguida, os resultados estabelecidos por este teorema são utilizados na análise da dedução de conhecimento a partir de um λ-programa.

5.3.1 TEOREMA.

Sejam as fórmulas,

$$\Phi_\alpha = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m)$$

$$\Psi_\beta = \lambda Y_1 \ldots \lambda Y_r (B\ H_1 \ldots H_s)$$

onde Φ é a forma normal de uma derivação em Ψ conforme a lista compatível

$$\mathbb{E} = [C_1 \ldots C_p], \quad 0 < p \leq n,$$

tal que C_i é uma fórmula neutra $1 \leq i \leq p$, então

(i)

$$\Phi_\alpha <^n \Psi_\beta$$

(ii)

$$\Phi_\alpha \leq^g \Psi_\beta$$

(iii)

$$\Phi_\alpha \leq^{dt} (\Psi_\beta\ C_1 \ldots C_p) + 2p$$

$$\Phi_\alpha \leq^{dp} (\Psi_\beta\ C_1 \ldots C_p) + 2p$$

(iv) Se $C_i \leq^{dt} \Psi_\beta$, $1 \leq i \leq p$, então

$$\Phi_\alpha \leq^{dt} \Psi_\beta + 3p.$$

e se $C_i \leq^{dp} \Psi_\beta$, $1 \leq i \leq p$, então

$$\Phi_\alpha \leq^{dp} \Psi_\beta + 3p.$$

(v) Se $B \in CONST$, então

$$A = B.$$

(vi) Se $B \in CONST$, então $r > n$.

DEMONSTRACAO.

(i) Φ_α é a forma normal de $\Psi_\beta\ C_1 \ldots C_p$, logo

$$\beta = \langle \beta_1 \succ \ldots \beta_p \succ \omega$$

onde

$$\beta_i = \text{tipo}[C_i],\ 1 \leq i \leq p$$

Como $p > 0$, então

$$\text{ordem}[\alpha] < \text{ordem}[\beta],$$

isto é,

$$\Phi_\alpha <^n \Psi_\beta.$$

(ii) Como $\Phi_\alpha = \Psi_\beta\ C_1 \ldots C_p$, o resultado segue da definição de grau.

(iii) É equivalente ao corolário 5.2.14.

(iv) É equivalente ao corolário 5.2.15.

(v) Tem-se que,

$$\Psi_\beta\ C_1 = (\lambda Y_1 \ldots \lambda Y_r (B\ H_1 \ldots H_s))\ C_1 =$$

$$= \lambda Y_2 \ldots \lambda Y_r\ (B\ (S_{C_1}^{Y_1} H_1) \ldots (S_{C_1}^{Y_1} H_s))$$

Observe que a constante B permanece inalterada. Como Φ_α é a forma normal de

$$(\Psi_\beta\ C_1 \ldots C_p)$$

então

$$A = B.$$

(vi) Demonstra-se utilizando um raciocínio análogo a (v).

CQD

***

O teorema 5.3.1 estabelece condições necessárias para que uma fórmula Φ possa ser obtida a partir da derivação de uma outra fórmula Ψ. Estas condições são estabelecidas a partir das relações de complexidade das fórmulas Φ e Ψ e dos argumentos $(C_1, \ldots, C_p)$ utilizados na derivação de Φ. As condições necessárias estabelecidas nos itens (iii) e (iv) do teorema 5.3.1 são válidas apenas para derivações onde os argumentos considerados pela função são fórmulas neutras. Quando os argumentos são funções genéricas, isto é, os argumentos $(C_1, \ldots, C_p)$ não são necessariamente fórmulas neutras, a análise é feita como é mostrado no apêndice C.

A seguir são consideradas as condições necessárias para a dedução de uma fórmula Φ a partir de um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Sao analisadas as condições necessárias sobre Φ para que exista uma sequência de derivação

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_k, \mathbb{D} \rangle$$

onde

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle = \langle \lambda\text{-}P, \mathbb{D} \rangle$$

$$\Phi \in \lambda\text{-}P_k$$

Nesta análise considera-se uma sequência de derivação particular definida a seguir.

5.3.2 DEFINICAO. Sequencia de derivacao restrita.

Uma sequência de derivação

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

é restrita quando para cada $\Phi_j \in \lambda\text{-}P_i$ se

$$C \in \mathbb{D}\left[\Phi_j, \lambda\text{-}P_i\right]$$

então

$$C \leq^{dt} \Phi_j$$

$$C \leq^{dp} \Phi_j$$

***

LEMA 5.6

Dada uma seqüência de derivação neutra e restrita

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_1, \mathbb{D}\rangle, \ldots, \langle \lambda\text{-}P_k, \mathbb{D}\rangle,$$

e uma fórmula $\Phi \in \lambda\text{-}P_i$, $0 \leq i \leq k$, onde

$$\Phi = \lambda X_1 \ldots \lambda X_n \ (A\ F_1 \ldots F_s)$$

Então,

$$\Phi + 3n \leq^{dl} \Psi^l + 3.Ml$$

$$\Phi + 3n \leq^{dp} \Psi^p + 3.Mp$$

onde Ψ^l, Ψ^p, Ml e Mp são tais que,

$$\Psi^l, \Psi^p \in \lambda\text{-}P_0,$$

$$\Psi^l = \lambda X_1 \ldots \lambda X_{Ml} \ (J\ L_1 \ldots L_b)$$

$$\Psi^p = \lambda Y_1 \ldots \lambda Y_{Mp} \ (P\ Q_1 \ldots Q_c)$$

$$ordem[\Psi^l] + 3.Ml = \max_{F,u} (\ ordem[F]^l + 3u$$

onde

$$F \in \lambda\text{-}P_0,$$

$$F = \lambda X_1 \ldots \lambda X_u \ (B\ C_1 \ldots C_p) \ )$$

$$ordem[\Psi^p] + 3.Mp = \max_{F,u} (\ ordem[F]^p + 3u$$

onde

$$F \in \lambda\text{-}P_0,$$

$$F = \lambda X_1 \ldots \lambda X_u \ (B\ C_1 \ldots C_p) \ )$$

DEMONSTRACAO.

A demonstração é feita considerando-se apenas a ordem total. O outro caso, ordem parcial, pode ser demonstrado de uma maneira análoga. A demonstração divide-se em dois casos.

(i) Se $\Phi \in \lambda\text{-}P_0$ o resultado é óbvio pela definição da fórmula Ψ^l e de M.

(ii) Se $\Phi \in \lambda\text{-}P_i$, $i>0$, então existe uma substituição θ e uma fórmula Ω tais que

$$\Omega = \Phi\theta$$

onde Ω é uma derivação em ξ,

$$\xi \in \lambda\text{-}P_{i-1}$$

$$\xi = \lambda Y_1 \ldots \lambda Y_r \ (C\ D_1 \ldots D_s)$$

isto é, existem fórmulas neutras

$$B_1, \ldots, B_p$$

tais que

$$\Omega = \xi\ B_1 \ldots B_p,\ p>0$$

Tem-se duas possibilidades para ξ.

(ii.a) $\xi \in \lambda\text{-}P_0$, logo, por definição,

$$\xi + 3r \leq^{dt} \Psi^t + 3.Mt$$

Utilizando o teorema 5.3.1

$$\Omega \leq^{dt} \xi + 3p$$

Como

$$\Omega = \Phi\theta$$

então

$$\Omega = \lambda X_1 \ldots \lambda X_n \ (\ (A\theta)\ (F_1\theta) \ldots (F_s\theta))$$

Por outro lado,

$$\Omega = \xi\ B_1 \ldots B_p$$

logo,

$$n + p = r$$

Da desigualdade

$$\Omega \leq^{dt} \xi + 3p$$

obtem-se,

$$\Omega + 3n \leq^{dt} \xi + 3r$$

Portanto,

$$\Omega + 3n \leq^{dt} \Psi^t + 3.Mt$$

Como $\Omega = \Phi\theta$, Pelo corolário 5.2.3

$$\Phi \leq^{dt} \Omega$$

logo,

$$\Phi + 3n \leq^{dt} \Psi^t + 3Mt$$

(ii.b) $\xi \notin \lambda\text{-}P_0$. Por hipótese tem-se que

a seqüência de derivação é neutra,

$$\xi \in \lambda\text{-}P_{i-1}$$

ξ não é uma fórmula neutra,

logo, existem fórmulas neutras

$$H_1, \ldots, H_v,$$

uma fórmula

$$\Gamma \in \lambda\text{-}P_0,$$

$$\Gamma = \lambda Z_1 \ldots \lambda Z_q \ (E\ G_1 \ldots G_l)$$

e uma substituição θ tal que

$$G = \Gamma\ H_1 \ldots H_v$$

$$G = \xi\theta$$

Utilizando o teorema 5.3.1

$$G \leq^{dt} \Gamma + 3w$$

Como

$$G = \Gamma\ H_1 \ldots H_v$$

então

$$r + w = q$$

Da desigualdade,

$$G \leq^{dt} \Gamma + 3w$$

obtem-se

$$G + 3r \leq^{dt} \Gamma + 3q$$

Como

$$G = \xi\theta$$

então pelo corolário 5.2.3 e utilizando um raciocínio análogo ao do item anterior, tem-se que,

$$\Omega + 3n \leq^{dt} \xi + 3r$$

e pela definição de Ψ^t e Mt,

$$\Phi + 3n \leq^{dt} \Psi^t + 3.Mt$$

CQD.

***

A seguir são analisadas as condições necessárias para a dedução por derivação de uma fórmula Φ a partir de um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

5.3.3 TEOREMA.

Considere uma fórmula Φ e um programa com critério

$$\langle \lambda\text{-}P, \mathbb{D} \rangle .$$

Se Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ então.

(i)

$$\Phi \leq^{n} k$$

onde

$$k = \max (\text{ordem}[\beta] \text{ tal que } F_{\beta} \in \lambda\text{-}P)$$

(ii)

$$\Phi \leq^{g} q$$

onde

$$q = \max (\text{grau}[F] \text{ tal que } F \in \lambda\text{-}P)$$

(iii) Se a derivação considerada na dedução de Φ é uma derivação neutra e restrita, então

$$\Phi \leq^{dt} \Psi^{t} + 3.Mt$$

$$\Phi \leq^{dp} \Psi^{p} + 3.Mp$$

onde Ψ^{t}, Ψ^{p}, Mt e Mp são tais que

$$\Psi^{t}, \Psi^{p} \in \lambda\text{-}P,$$

$$\Psi^{t} = \lambda X_{1} \ldots \lambda X_{Mt} \ (J \ L_{1} \ldots L_{b})$$

$$\Psi^{t} = \lambda Y_{1} \ldots \lambda Y_{Mp} \ (P \ Q_{1} \ldots Q_{c})$$

$$\text{ordem}[\Psi^{t}) + 3.Mt = \max_{F,u} (\text{ordem}[F]^{t} + 3u$$

onde

$$F \in \lambda\text{-}P,$$

$$F = \lambda X_{1} \ldots \lambda X_{u} \ (B \ C_{1} \ldots C_{p}) \ )$$

$$ordem(\Psi^P) + 3.Mp = \max_{F,u} (ordem(F)^P + 3u$$

onde

$$F \in \lambda\text{-}P,$$

$$F = \lambda X_1 \ldots \lambda X_u \ (B\ C_1 \ldots C_p)\ )$$

DEMONSTRACAO.

Como Φ é uma dedução por derivação a partir de

$$\langle \lambda\text{-}P,\ \mathbb{D} \rangle,$$

então existe uma sequência de derivação

$$\langle \lambda\text{-}P_0,\ \mathbb{D} \rangle,\ \langle \lambda\text{-}P_1,\ \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_k,\ \mathbb{D} \rangle$$

onde

$$\langle \lambda\text{-}P_0,\ \mathbb{D} \rangle = \langle \lambda\text{-}P,\ \mathbb{D} \rangle$$

$$\Phi \in \lambda\text{-}P_k$$

Além disso, Φ é obtida como se segue: existe uma substituição θ e uma fórmula ξ tais que

$$\xi = \Phi\theta$$

e

$$\xi = (\lambda X_1 \ldots \lambda X_p \ (A\ F_1 \ldots F_s))\ B_1 \ldots B_p$$

onde

$$\lambda X_1 \ldots \lambda X_p \ (A\ F_1 \ldots F_s) \in \lambda\text{-}P_{k-1}$$

Observa-se que quando

$$\theta = ( )$$

a fórmula Φ é obtida diretamente por uma derivação a partir de uma fórmula de $\lambda\text{-}P_{k-1}$. Isto é,

$$\Phi = (\lambda X_1 \ldots \lambda X_p \ (A\ F_1 \ldots F_s))\ B_1 \ldots B_p$$

onde

$$\lambda X_1 \ldots \lambda X_p \ (A\ F_1 \ldots F_s) \in \lambda\text{-}P_{k-1}$$

Considera-se a seguir a demonstração de cada item do teorema.

(i) A demonstração é feita por indução no comprimento K da sequência de derivação

$$\langle \lambda\text{-}P_0,\ \mathbb{D} \rangle,\ \langle \lambda\text{-}P_1,\ \mathbb{D} \rangle, \ldots, \langle \lambda\text{-}P_k,\ \mathbb{D} \rangle$$

(i.a) Se K = 1 então $\Phi \in \lambda\text{-}P$ e o resultado é imediato.

(i.b) Suponha o resultado válido para $K \leq N$. Isto é, se existe uma sequência de derivação

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_1, \mathbb{D}\rangle, \ldots, \langle \lambda\text{-}P_k, \mathbb{D}\rangle$$

tal que,

$$H \in \lambda\text{-}P_k$$

onde $k \leq N$. Então

$$H \leq^n k$$

Suponha agora que Φ é deduzida a partir de $\langle \lambda\text{-}P, \mathbb{D}\rangle$ por uma sequência de derivação com comprimento igual a $N + 1$. Isto é, existe uma sequência

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_0, \mathbb{D}\rangle, \ldots, \langle \lambda\text{-}P_N, \mathbb{D}\rangle$$

onde

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle = \langle \lambda\text{-}P, \mathbb{D}\rangle$$

$$\Phi \in \lambda\text{-}P_N$$

Pela definição de derivação, existe uma substituição θ e uma fórmula ξ tais que,

$$\xi = \Phi\theta$$

onde

$$\xi = (\lambda X_1 \ldots \lambda X_p\ A)\ B_1 \ldots B_p$$

e

$$\lambda X_1 \ldots \lambda X_p\ A \in \lambda\text{-}P_{N-1}$$

Logo, ξ é uma derivação em

$$\lambda X_1 \ldots \lambda X_p\ A$$

Se $p>0$, então pelo teorema 5.3.1

$$\xi <^n \lambda X_1 \ldots \lambda X_p\ A$$

Quando $p=0$, então

$$\xi = A$$

Portanto, para $p \geq 0$,

$$\xi \leq^n \lambda X_1 \ldots \lambda X_p\ A$$

Utilizando a hipótese de indução,

$$\lambda X_1 \ldots \lambda X_p\ A \leq^n k$$

$$k = \max(\ \text{ordem}[\beta] \text{ tal que } F_\beta \in \lambda\text{-}P\ )$$

Logo,

$$\xi <^n k$$

Por outro lado, como $\Phi = \xi\theta$, onde θ é uma substituição, então

$$\Phi =^{n} \xi$$

logo,

$$\Phi <^{n} k$$

(ii) Esta demonstração é análoga a do item anterior. Deve-se observar apenas que se

$$\xi = \Phi\theta$$

onde θ é uma substituição, então

$$\xi =^{g} \Phi$$

(iii) A demonstração neste caso é feita considerando-se apenas a ordem total. Quando se considera a ordem parcial a uma demonstração análoga.Suponha que Φ é deduzida a partir de $\langle \lambda\text{-}P, \mathbb{D}\rangle$ pela sequência de derivação

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_1, \mathbb{D}\rangle, \ldots, \langle \lambda\text{-}P_k, \mathbb{D}\rangle$$

onde

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle = \langle \lambda\text{-}P, \mathbb{D}\rangle$$

$$\Phi \in \lambda\text{-}P_k$$

Por definição de sequência de derivação, existem uma substituição θ e uma fórmula ξ tais que,

$$\xi = \Phi\theta$$

onde

$$\xi = (\lambda X_1 \ldots \lambda X_p\ A\ )\ B_1 \ldots B_p$$

e

$$\lambda X_1 \ldots \lambda X_p\ A \in \lambda\text{-}P_{k-1}$$

Logo, ξ é uma derivação em

$$\lambda X_1 \ldots \lambda X_p\ A$$

Como a sequência de derivação é neutra e restrita, as fórmulas

$$B_1 \ldots B_p$$

sao fórmulas neutras e

$$B_i \leq^{dt} \lambda X_1 \ldots \lambda X_p\ A$$

Se $p>0$, então pelo teorema 5.3.1,

$$\xi \leq^{dt} (\lambda X_1 \ldots \lambda X_p\ A) + 3p$$

No caso em que $p=0$, as desigualdades são obvias.

Como a sequência de derivação é neutra e restrita, pelo lema 5.6,

$$(\lambda X_1 \ldots \lambda X_p\ A) + 3p \leq^{dt} \Psi^t + 3.M_t$$

logo,

$$\xi \leq^{dt} \Psi^t + 3.M_t$$

Como

$$\xi = \Phi\theta$$

pelo corolário 5.2.3.

$$\Phi \leq^{dt} \xi$$

logo,

$$\Phi \leq^{dt} \Psi^t + 3.M_t$$

COD.

***

5.4 APLICACAO DAS CONDICOES NECESSARIAS.

Uma aplicação do λ-calculo tipado considerada neste trabalho refere-se à utilização das condições necessárias, demonstradas na seção anterior, na derivação de conhecimento a partir de programas com critério. Esta derivação de conhecimento pode ocorrer de duas formas distintas: a derivação "forward" e a derivação "backward", ou por retrocesso. Considera-se a seguir a relação entre as condições necessárias e as diferentes sequências de derivação.

5.4.1 AS CONDICOES NECESSARIAS E AS SEQUENCIAS DE DERIVACAO.

Dados um λ-programa λ-P e uma fórmula Φ, uma sequência de derivação por retrocesso

$$\langle \lambda\text{-}P,\ \theta_1, OB_1 \rangle,\ \langle \lambda\text{-}P,\ \theta_2, OB_2 \rangle, \ldots, \langle \lambda\text{-}P,\ \theta_n, OB_n \rangle$$

é determinada definindo cada passo de derivação por retrocesso. O conjunto OB_{i+1} e a substituição θ_{i+1} são obtidos a partir de OB_i e θ_i conforme a definição de passo de derivação por retrocesso, definição 3.6.1. O passo de derivação por retrocesso subdivide-se em dois itens: a redução do objetivo e o passo de retrocesso. Neste último, se existem fórmulas

$$\Phi \in OB_i \text{ e } \Gamma \in \lambda\text{-}P$$

onde Φ possui uma derivação em Ψ com resposta θ, conforme a lista compatível

$$[A_1, \ldots, A_s]$$

Isto é, existe ξ tal que,

$$\xi = \Psi A_1 \ldots A_n$$

e

$$\xi = \Phi\theta$$

Então,

$$OB_{i+1} = OB \cup (A_1, \ldots, A_n)$$

onde

$$OB = (OB_i - \Phi)\theta$$

e

$$\theta_{i+1} = \theta_i\theta$$

Para determinar o conjunto OB_{i+1} e a substituição θ_{i+1} a partir de OB_i e θ_i é necessário responder se existem as fórmulas Φ e Γ tais que

$$\Phi \in OB_i \text{ e } \Gamma \in \lambda\text{-}P$$

onde Φ é uma derivação em Γ. Dada uma fórmula $\Phi \in OB_i$ para que exista uma tal fórmula $\Gamma \in \lambda\text{-}P$ é necessário que

$$\Phi \leq^{dt} \Psi^t + 3.Mt$$

$$\Phi \leq^{dp} \Psi^p + 3.Mp$$

onde Ψ^t, Ψ^p, Mt e Mp são definidos no teorema 5.3.3. Este fato determina restrições sobre as fórmulas de OB_i para que a sequência de derivação por retrocesso seja construída.

Considerando um raciocínio análogo àquele desenvolvido no final do capítulo 3, dados uma fórmula Φ e um λ-programa $\lambda\text{-}P$, se existe uma derivação por retrocesso com sucesso a partir de

$$\langle \lambda\text{-}P, \theta_1, OB_1 \rangle$$

onde

$$OB_1 = (\Phi),$$

$$\theta_1 = ( ),$$

então existe um critério $\mathbb{D}$ tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Inversamente, é imediato que se Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$, então existe uma sequência de derivação por retrocesso com sucesso a partir de

$$\langle \lambda\text{-}P, \theta_1, OB_1 \rangle$$

Portanto, para responder se é possível, ou não, determinar um critério $\mathbb{D}$ tal que a fórmula Φ seja uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ deve-se determinar se é possível, ou não, a construção de uma sequência de derivação por retrocesso a partir de

$$\langle \lambda\text{-}P, \theta_1, OB_1 \rangle$$

Se é possível a construção de uma sequência de derivação neutra e restrita, então é necessário que

$$\Phi \leq^{dt} \Psi^t + 3.Mt$$

$$\Phi \leq^{dp} \Psi^p + 3.Mp$$

onde Ψ^t, Ψ^p, Mt e Mp são definidos no teorema 5.3.3.

Como a sequência de derivação apresentada na seção 3.5 é neutra e restrita e as fórmulas deduzidas na sequência de derivação são constantes, é imediato verificar que as condições necessárias, demonstradas na seção anterior, são satisfeitas.

Observa-se também que as sequências de derivação apresentadas nos apendices A e B não são neutras, logo as condições necessárias demonstradas na seção anterior não se aplica.

As condições necessárias estabelecem restrições sobre a complexidade de uma fórmula Φ para que ela seja uma derivação por dedução a partir de um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Tem-se também o inverso. Isto é, são estabelecidas restrições sobre o λ-programa $\lambda\text{-}P$ para que exista uma dedução por derivação da fórmula Φ a partir do programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. As restrições sobre $\lambda\text{-}P$ são tais que existam Ψ^t, Ψ^p, Mt e Mp como definidos no teorema 5.3.3 tais que

$$\Phi \leq^{dt} \Psi^t + 3.Mt$$

$$\Phi \leq^{dp} \Psi^p + 3.Mp$$

5.5 CONCLUSAO.

Neste capítulo foram demonstradas as condições necessárias para que uma fórmula Φ seja uma dedução por derivação a partir de um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Quando a fórmula Φ possui complexidades quanto ao grau, quanto ao nível e quanto à descrição total e parcial menores que as respectivas complexidades das fórmulas pertencentes a $\lambda\text{-}P$, então esta fórmula pode ser uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Estas condições determinam restrições sobre a fórmula Φ para que seja possível a construção de uma sequência de derivação neutra restrita a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

No próximo capítulo, é considerada a representação hierárquica de conhecimento, utilizando os programas com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

CAPITULO 6

REPRESENTACAO HIERARQUICA DE CONHECIMENTO.

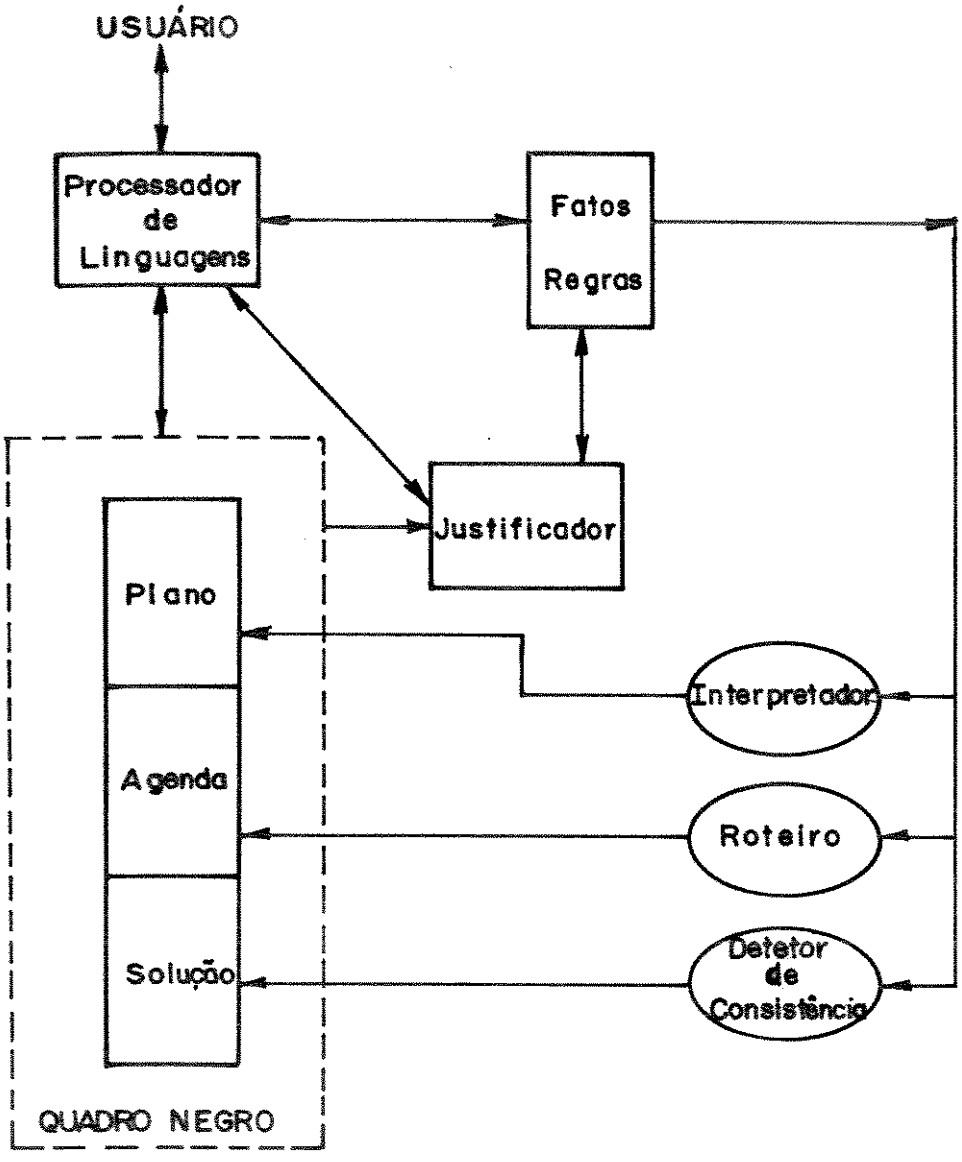
Neste capitulo e analisada a representacao hierarquica de conhecimento utilizando programas com criterio $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Nesta representacao considera-se uma hierarquizacao das formulas do λ-programa $\lambda\text{-}P$. Esta hierarquia e definida a partir das relacoes de complexidade entre as formulas do λ-calculo tipado. Tem-se tambem uma ilustracao da representacao hierarquica de conhecimentos com a definicao de uma arquitetura de um sistema onde os conhecimentos sao representados por formulas do λ-calculo tipado.

6.1 INTRODUCAO.

Neste capítulo é analisada a representação hierárquica de conhecimento a partir de um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. A hierarquia da representação do conhecimento fundamenta-se em uma hierarquia das fórmulas do λ-programa $\lambda\text{-}P$. É introduzida uma hierarquia que utiliza noções fundamentais do λ-cálculo tipado e as relações de complexidade propostas no capítulo 4. O critério $\mathbb{D}$ e o sequenciamento da utilização do conhecimento representado pelas fórmulas do λ-programa $\lambda\text{-}P$ são definidos a partir da hierarquia das fórmulas de $\lambda\text{-}P$. São considerados vários sequenciamentos na utilização do conhecimento: (i) do menos complexo para os mais complexo (modo ascendente), (ii) do mais complexo para os menos complexo (modo descendente), etc.

Na literatura existem diversas propostas de esquemas básicos ou arquiteturas para a representação de conhecimento, [Hayes-Roth, 1983], [Buchanan, 1984], [Stefik, 1981], [Taylor, 1984], [Laird, 1987], [Erman, 1980], [Forsyth, 1983], [Brachman, 1985], [Hayes-Roth, 1984], [Hayes-Roth, 1985(a)][Hayes-Roth, 1985(b)]. Cada uma delas enfatiza determinados aspectos da representação do conhecimento envolvido. Elas são bastantes gerais, pouco profundas e não detalham o funcionamento de seus componentes. Entretanto, são úteis pois explicitam partes de um problema mais geral a ser analisado. Em [Hayes-Roth, 1983] é proposta uma arquitetura genérica de um sistema para representação de conhecimento, figura 6.1. Conforme é citado pelo próprio autor, este esquema

é bastante completo e não existe sistema que apresenta todas as caracteristicas indicadas nele. Este o esquema é constituído por:



Arquitetura para representacao de conhecimentos genericos.
FIGURA 6.1

(i) Um processador de linguagem que possibilita a comunicação do usuário com o sistema.

(ii) Um justificador que responde perguntas feitas pelo usuário justificando seus resultados.

(iii) Um quadro negro constituído por um planejador onde estão armazenados os planos utilizados na resolução de um problema. Tem-se também uma agenda onde está armazenado um conjunto de regras factíveis e que pode ser executado a qualquer momento. Finalmente, há um elemento onde estão armazenadas todas as

soluções encontradas pelo sistema e suas interelações.

(iv) Uma base de conhecimento onde são armazenadas regras e fatos pertinentes ao domínio de conhecimento do sistema.

(v) Um interpretador que escolhe regras entre as factíveis, baseado no conhecimento representado pela base de regras.

(vi) Um roteiro que determina o gerenciamento da agenda.

(vii) Um teste de consistência das soluções obtidas pelo sistema.

A arquitetura para representação hierárquica de conhecimento, proposta neste trabalho, é denominada arquitetura EEO e foi inicialmente analisada em [Souza, 1988]. A arquitetura EEO pode ser considerada como um sistema para representar determinadas características propostas por Hayes-Roth. Neste trabalho não se considera a análise dos elementos da interface do sistema com o usuário, como o processador de linguagens e o justificador, indicados nos itens (i) e (ii). Na arquitetura EEO, o conhecimento é representado por uma lista de fórmulas do λ-cálculo tipado, λ-programa, associada a um critério de derivação. Isto é, o conhecimento é representado por um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. As fórmulas de λ-P representam o conhecimento armazenado no quadro negro e na base de conhecimento do sistema, conforme são indicados no itens (iii) e (iv), respectivamente. O interpretador, o roteiro é o elemento que determina o teste de consistência, indicados nos itens (v), (vi) e (vii), respectivamente, são representados no criterio $\mathbb{D}$.

Como os elementos básicos de um programa com critério são fórmulas do λ-cálculo tipado, a representação hierárquica de conhecimento por esta arquitetura é facilitada, como será apresentado neste capítulo.

Inicialmente, na seção 6.2 é proposta a hierarquização de um conjunto de fórmulas do λ-cálculo tipado. Esta hierarquização é definida a partir das relações de complexidade das fórmulas do λ-cálculo tipado e determina uma hierarquização de λ-programas. Nas seções 6.2.1 e 6.2.2 propõe-se o conceito de hierarquia em uma lista de fórmulas e hierarquia completa, respectivamente. A partir destes conceitos, propõe-se, na seção 6.2.4, o algoritmo da hierarquia completa. Este algoritmo determina, a partir de um conjunto qualquer de fórmulas, um conjunto hieraraquizado de fórmulas, representado por uma lista. O conjunto é ordenado a partir das fórmulas menos complexa, para as mais complexa, conforme as relações de complexidade definidas no capítulo 4. Em seguida, na seção 6.2.5, demonstra-se a correção do algoritmo.

Na seção 6.3 os elementos fundamentais da arquitetura EEO são descritos.

Na seção 6.4 é proposto o λ-programa da arquitetura EEO. Da seção 6.4.2 à seção 6.4.4 são definidas as fórmulas que representam os passos elementares da arquitetura.

Na seção 6.5 é proposto o critério de derivação. Este critério é definido a partir de conceitos básicos definidos nas seções 6.5.1 e 6.5.2.

Na seção 6.6 propõe-se um sequenciamento das funções representadas no λ-programa λ-*P*. Na seção 6.6.1 considera-se o sequenciamento dos passos elementares e na seção 6.6.2 o sequenciamento da execução de funções de diferentes graus.

Na seção 6.7 propõe-se o controle do sequenciamento dos passos elementares. São definidas, na seção 6.7.1, as funções controladoras.

Na seção 6.8 é proposto um algoritmo que controla os modos de operação do sistema.

6.2 HIERARQUIZACAO DE UM CONJUNTO DE FORMULAS.

Como é proposto no capítulo 4, as fórmulas do λ-cálculo tipado, que representam conhecimento em um sistema, estão associadas a diferentes complexidades. Tem-se uma representação por fórmulas de diferentes complexidades quanto ao grau, quanto ao nível e quanto à descrição total ou parcial. A seguir, as relaçóes de complexidade são utilizadas para estabelecer uma hierarquia em um conjunto de fórmulas do λ-cálculo tipado.

Inicialmente, observe que fórmulas de um mesmo grau podem estar associadas a símbolos para tipos cujas ordens são diferentes. Isto é, dadas duas fórmulas Φ_α, Ψ_β pode ocorrer que

$$\Phi_\alpha =^g \Psi_\beta$$

e

$$\mathbf{ordem}[\alpha] \neq \mathbf{ordem}[\beta]$$

isto é,

$$\Phi_\alpha \neq^n \Psi_\beta$$

Um conhecimento de grau k pode ser representado por uma fórmula Ψ_α em diversos níveis de complexidade. Neste caso a única restrição que se tem, conforme a proposição 4.5.3,. é que

$$\Psi >^n k$$

Dada uma lista de fórmulas

$$[\Psi_1, \ldots, \Psi_n],$$

todas representando um conhecimento de mesmo grau, isto é,

$$\Psi_i =^g \Psi_j, \quad 1 \leq i, j \leq n$$

tem-se uma hierarquia, ou ordenação entre elas, estabelecida pela relação de ordem

$$"<^n".$$

Isto é, a lista de fórmulas acima pode ser ordenada de tal forma que,

$$\Psi_1 \leq^n \ldots \leq^n \Psi_n$$

Por outro lado, fórmulas equivalentes quanto ao grau e ao nível podem ter diferentes complexidades quanto à descrição total. Isto é, pode se ter duas fórmulas Ψ e Φ tais que

$$\Psi =^g \Phi$$

$$\Psi =^n \Phi$$

e

$$\Psi \neq^{dt} \Phi$$

Dada uma lista

$$[\xi_1, \ldots, \xi_p]$$

de fórmulas tais que,

$$\xi_i \ ^g\!\!= \xi_j,$$

$$\xi_i \ ^n\!\!= \xi_j,$$

onde $1 \leq i, j \leq p$, ela pode ser ordenada pela relação de ordem

$$"<^{dt}"$$

determinando uma hierarquia, ou ordenação da lista tal que

$$\xi_1 \leq^{dt} \ldots \leq^{dt} \xi_p$$

Define-se a seguir vários tipos de hierarquias, ou ordenação entre as fórmulas do λ-cálculo tipado. Inicialmente, considera-se as hierarquias definidas por uma única relação de complexidade. Em seguida, considera-se uma hierarquia definida por várias relações de complexidade.

6.2.1 DEFINICAO. Hierarquia em uma lista de formulas.

Seja $\mathbb{B} = [\Phi_1,\ldots,\Phi_n]$ uma lista de fórmulas.

(i) Se

$$\Phi_1 \leq^g \ldots \leq^g \Phi_n$$

diz-se que $\mathbb{B}$ está ordenado quanto ao grau, ou que $\mathbb{B}$ é uma hierarquia quanto ao grau.

(ii) Se

$$\Phi_1 \leq^n \ldots \leq^n \Phi_n$$

diz-se que $\mathbb{B}$ está ordenado quanto ao nível, ou que $\mathbb{B}$ é uma hierarquia quanto ao nível.

(iii) Se

$$\Phi_1 \leq^{dt} \ldots \leq^{dt} \Phi_n$$

diz-se que $\mathbb{B}$ está ordenado quanto à descrição total, ou que $\mathbb{B}$ é uma hierarquia quanto à descrição total.

***

6.2.2 DEFINICAO. Hierarquia completa.

Uma lista de fórmulas

$$\mathbb{B} = [\Phi_1,\ldots,\Phi_n]$$

é uma hierarquia completa se dado duas fórmulas quaisquer

$$\Phi_i, \Phi_j \in \mathbb{B},\ i < j,$$

então uma das seguintes condições são válidas.

(a) $\Phi_i <^g \Phi_j$,

(b) Se

$$\Phi_i =^g \Phi_j,$$

então

$$\Phi_i <^n \Phi_j$$

e a lista de fórmulas

$$[\Phi_i, \Phi_{i+1}, \ldots, \Phi_j]$$

é uma hierarquia quanto ao nível. Isto é,

$$\Phi_i \leq^n \ldots \leq^n \Phi_j$$

(c) Se

$$\Phi_i =^g \Phi_j,$$

e

$$\Phi_i =^n \Phi_j,$$

então

$$\Phi_i <^{dt} \Phi_j$$

e a lista de fórmulas

$$[\Phi_i, \Phi_{i+1}, \ldots, \Phi_j]$$

é uma hierarquia quanto a descrição total. Isto é,

$$\Phi_i \leq^{dt} \ldots \leq^{dt} \Phi_j$$

(d) Se

$$\Phi_i =^g \Phi_j,$$

$$\Phi_i =^n \Phi_j,$$

e

$$\Phi_i =^{dt} \Phi_j,$$

então a lista de fórmulas

$$(\Phi_i, \Phi_{i+1}, \ldots, \Phi_j)$$

esta ordenado em uma ordem qualquer.

***

Neste trabalho, propõe-se que a hierarquização do conhecimento representado em um sistema seja feita conforme a hierarquia completa definida acima. Inicialmente as fórmulas são ordenadas quanto ao grau, considerando-se como conhecimento básico o conhecimento representado por fórmulas de grau zero. O conhecimento sobre o conhecimento básico, denominado de meta-conhecimento [Buchanan, 1984], é representado por fórmulas de grau 1. O conhecimento sobre o meta-conhecimento, denominado de meta-meta-conhecimento, é representado por fórmulas de grau 2 e assim por diante.

Considere, por exemplo, um colégio onde se tem alunos, professores, inspetores e um diretor. Se os alunos são representados por um conhecimento básico, isto é, por fórmulas de grau zero, então os professores, que possuem conhecimento sobre os alunos, são representados por um meta-conhecimento, ou seja, por fórmulas de grau 1. Os inspetores, que possuem conhecimento sobre os professores, são representado por um meta-meta-conhecimento (grau 2). Finalmente, o diretor, que possui conhecimento sobre os inspetores, é representado por um meta-meta-meta-conhecimento (grau 3).

Além de ordenar os conhecimentos do sistema quanto ao grau, a hierarquia completa também considera a ordenação quanto ao nível. Isto significa que os professores e inspetores são diferenciados entre si. Considere, simplificadamente, que um professor seja representado por uma função

$$\Psi_\gamma = \lambda X_{\alpha_1} \ldots \lambda X_{\alpha_n} H_\beta$$

onde $X_{\alpha_1} \ldots X_{\alpha_n}$ são variáveis que representam os seus alunos. Como os alunos são representados por um conhecimento básico, então

$$\alpha_i \in \mathbb{F}, \ 1 \leq i \leq n$$

Considerando,

$$\mathbf{ordem}[\beta] = k$$

então

$$\mathbf{ordem}[\gamma] = n + k$$

Assim, quanto maior o número de alunos do professor, maior a ordem do símbolo para tipo associado à função que o representa. Se dois professores P_1 e P_2 são representados, respectivamente, pelas funções

$$\Psi_\gamma = \lambda X_{\alpha_1} \ldots \lambda X_{\alpha_n} H_\beta$$

e

$$\Phi_\eta = \lambda X_{\sigma_1} \ldots \lambda X_{\sigma_m} G_{\beta'}$$

onde $n > m$, então

$$\Psi_\gamma >^n \Phi_\eta$$

Isto é, o conhecimento que representa P_1 é mais complexo, quanto ao nível, que o conhecimento que representa P_2. Tem-se uma representação sintática do seguinte fato: um professor deve ter além de um conhecimento sobre cada um de seus alunos, um conhecimento que o possibilite compará-los. Assim, quanto maior o número de alunos, maior deve ser o conhecimento do professor, possibilitando-o comparar seus alunos. O conhecimento que representa os professores é ordenado, quanto ao nível, dos professores com poucos alunos, para os professores com muitos alunos, o mesmo ocorrendo com a ordenação do conhecimento que representa os inspetores.

Finalmente, além de ordenar as fórmulas quanto ao grau e ao nível, a hierarquia completa considera a ordenação quanto à descrição total. Dadas duas fórmulas de mesmo grau e mesmo nivel, a descrição total compara a estrutura de formação destas fórmulas. Assim, se o conhecimento de dois professores é representado por duas fórmulas Γ_1 e Γ_2, de mesmo nivel, a descrição total diferencia este conhecimento considerando a estrutura de formação e a ordem dos símbolos para tipo associados as constantes e variaveis de Γ_1 e Γ_2.

Observa-se que a hierarquização do conhecimento representado em um sistema conforme a hierarquia completa não é a única possível. Esta é apenas a proposta considerada neste trabalho para efeito de análise. No que se segue, os conceitos apresentados podem ser modificados caso a ordenação considerada na definição da hierarquia completa seja outra.

Propõe-se a seguir o algoritmo da hierarquia completa. Dado um conjunto $\mathbb{A}$ de fórmulas, este algoritmo determina uma lista $\mathbb{B}$, que é uma hierarquia completa, obtida a partir de $\mathbb{A}$, enumerando suas fórmulas. Inicialmente, define-se a relação de ordem

$$\ll$$

que é utilizada na definição de algoritmo da hierarquia completa. Esta relação de ordem estabelece uma ordenação em um conjunto de fórmulas indexadas por listas de inteiros.

6.2.3 DEFINICAO. Relacao de ordem "$\ll$".

Dadas duas fórmulas

$$\Phi_{[i_1,j_1,m_1,u_1]},$$

$$\Phi_{[i_2,j_2,m_2,u_2]}$$

onde os indices

$$[i_1,j_1,m_1,u_1]$$

e

$$[i_2,j_2,m_2,u_2]$$

são listas de inteiros, então

$$\Phi_{[i_1,j_1,m_1,u_1]} \ll \Phi_{[i_2,j_2,m_2,u_2]}$$

se e somente se, uma das condições a seguir ocorre.

(a) $i_1 < i_2$

(b) Se $i_1 = i_2$, então

$$j_1 < j_2$$

(c) Se

$$i_1 = i_2,$$

e

$$j_1 = j_2,$$

então

$$m_1 < m_2$$

(d) Se

$$i_1 = i_2,$$

$$j_1 = j_2,$$

e

$$m_1 = m_2,$$

então

$$u_1 < u_2$$

***

Antes de definir o algoritmo da hierarquia completa, observe que as relações

$$"=^g", \quad "=^n" \quad e \quad "=^{dt}"$$

são relações de equivalência conforme [Lipschutz,1972], [Simis, 1973].

6.2.4 ALGORITMO DA HIERARQUIA COMPLETA.

Seja $\mathbb{A} = \{\Phi, \xi, \Psi, \ldots, \Gamma\}$ um conjunto de fórmulas.

(i) Sejam

$$\mathbb{C}[g_1], \ \mathbb{C}[g_2], \ldots, \ \mathbb{C}[g_a]$$

as classes de equivalência de $\mathbb{A}$ conforme a relação $"=^g"$, tais que se

$$\Phi \in \mathbb{C}[g_i],$$

$$\xi \in \mathbb{C}[g_j],$$

onde $1 \leq i < j \leq a$, então

$$\Phi =^g g_i,$$

$$\xi =^g g_j$$

e

$$g_i < g_j.$$

isto é,

$$\Phi <^g \xi$$

(ii) Dada uma classe de equivalência $\mathbb{C}lg_i$, sejam

$$\mathbb{D}lg_{i,n_1}, \mathbb{D}lg_{i,n_2}, \ldots, \mathbb{D}lg_{i,n_b},$$

as classes de equivalência de $\mathbb{C}lg_i$ conforme a relação "$=^n$", tais que se

$$\Phi \in \mathbb{D}lg_{i,n_k},$$

$$\xi \in \mathbb{D}lg_{i,n_l},$$

onde $1 \leq k < l \leq b$, então

$$\Phi =^n n_k,$$

$$\xi =^n n_l$$

e

$$n_k < n_l.$$

isto é,

$$\Phi <^n \xi$$

(iii) Dada uma classe de equivalência $\mathbb{D}lg_{i,n_j}$, sejam

$$\mathbb{E}lg_{i,n_j,o_1}, \mathbb{E}lg_{i,n_j,o_2}, \ldots, \mathbb{E}lg_{i,n_j,o_c}$$

as classes de equivalência de $\mathbb{D}lg_{i,n_j}$ conforme a relação "$=^{dt}$" tais que se

$$\Phi \in \mathbb{E}lg_{i,n_j,o_m},$$

$$\xi \in \mathbb{E}lg_{i,n_j,o_n},$$

onde $1 \leq m < n \leq r$, então

$$\Phi =^{dt} o_m,$$

$$\xi =^{dt} o_n$$

e

$$o_m < o_n.$$

isto é,

$$\Phi <^{dt} \xi$$

(iv) Dada uma classe de equivalência

$$\mathbb{E}lg_{i,n_j,o_m} = (A,B,\ldots,C)$$

seja $\mathbb{G}lg_{i,n_j,o_m}$ o conjunto de fórmulas obtidas a partir de $\mathbb{E}lg_{i,n_j,o_m}$ enumerando suas fórmulas conforme qualquer critério. Isto é,

$$\mathbb{G}lg_{i,n_j,o_m} = (\Phi_1,\ldots,\Phi_v),$$

$$\Phi_u \in \mathbb{E}lg_{i,n_j,o_m}$$

(v) Dada uma fórmula $\Phi_u \in \mathbb{G}lg_{i,n_j,o_m}$ associe a ela a lista, (i,j,m,u), indicando-a por

$$\Phi_{(i,j,m,u)}$$

(vi) Seja então $\mathbb{B}$ a lista de fórmulas obtidas a partir de $\mathbb{A}$, ordenando suas fórmulas conforme relação de ordem "$\ll$", definição 6.2.3. Isto é,

$$\mathbb{B} = [\Psi_1,\ldots,\Psi_n]$$

onde $\Psi_i \ll \Psi_j$ se $i < j$.

***

A proposição a seguir determina a correção do algoritmo.

6.2.5 PROPOSICAO.

Dado um conjunto de fórmulas $\mathbb{A}$, seja $\mathbb{B}$ a lista de fórmulas obtidas a partir de $\mathbb{A}$ pelo algoritmo da hierarquia completa. Entao $\mathbb{B}$ contém todas as fórmulas de $\mathbb{A}$ e é uma hierarquia completa.

DEMONSTRACAO.

Observe que o algoritmo enumera todas as fórmulas de $\mathbb{A}$, pois dada uma relação de equivalência em $\mathbb{A}$, a união das classes de equivalência desta relação é igual a $\mathbb{A}$ [Lipschuts, 1972], [Simis,1973]. Logo, $\mathbb{B}$ contém todas as fórmulas de $\mathbb{A}$.

Resta demonstrar que $\mathbb{B}$ é uma hierarquia completa. Considere

$$\mathbb{B} = [\Phi_1, \ldots, \Phi_n],$$

logo, dadas duas fórmulas

$$\Phi_r, \Phi_s, \quad r<s,$$

então,

$$\Phi_r \ll \Phi_s.$$

Sejam

$$[i_r, j_r, m_r, u_r],$$

$$[i_s, j_s, m_s, u_s]$$

as listas de inteiros associadas as fórmulas Φ_r e Φ_s respectivamente. Como $\Phi_r \ll \Phi_s$, então $i_r \leq i_s$. Tem-se também que

$$i_r = \text{grau}[\Phi_r]$$

$$i_s = \text{grau}[\Phi_s]$$

isto é,

$$\Phi_r \leq^g \Phi_s$$

Se

$$\Phi_r <^g \Phi_s.$$

então o item (a) da definição 6.2.3 é satisfeito. Se

$$\Phi_r =^g \Phi_s.$$

então $i_r = i_s$. Logo, pela definição de "$\ll$",

$$j_r \leq j_s$$

Considere o caso em que

$$j_r < j_s$$

então deve-se demonstrar que a lista de fórmulas

$$[\Phi_r, \Phi_{r+1}, \ldots, \Phi_s]$$

é uma hierarquia quanto ao nível. Como

$$\Phi_r \ll \ldots \ll \Phi_s,$$

e

$$i_r = i_s,$$

então pela definição da relação "$\ll$" tem-se que,

$$i_r = i_{r+1} = \ldots = i_s$$

logo

$$j_r \leq j_{r+1} \leq \ldots \leq j_s$$

e então,

$$\Phi_r{}^n \leq \ldots {}^n \leq \Phi_s$$

o que verifica o item (b) da definição 6.2.3. No caso em que

$$j_r = j_s$$

pela definição de "$\ll$" tem-se que

$$m_r \leq m_s$$

No caso em que

$$m_r < m_s$$

segue-se um raciocínio análogo ao desenvolvido acima e demonstra-se que a lista de fórmulas

$$[\Phi_r, \Phi_{r+1}, \ldots, \Phi_s]$$

é uma hierarquia quanto à descrição total, o que verifica o item (c) da definição 6.2.3. Se

$$m_r = m_s$$

então pela definição de "$\ll$"

$$u_r \leq u_s$$

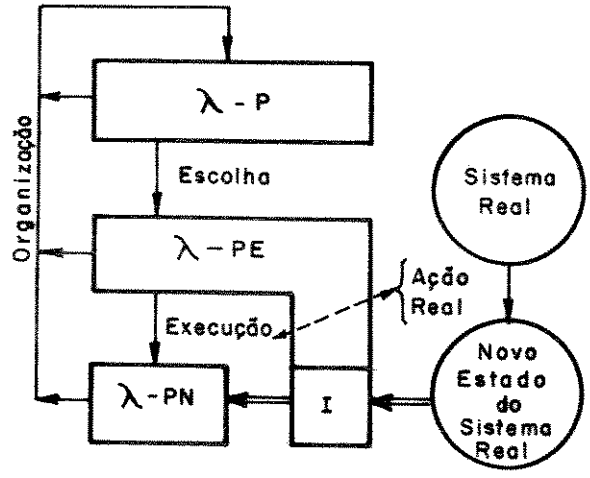
Seguindo também a um raciocínio análogo ao descrito acima, verifica-se o item (d) da definição 6.2.3.

CQD

***

6.3 UMA ARQUITETURA PARA A REPRESENTACAO HIERARQUICA DE CONHECIMENTO.

Propõe-se a seguir, figura 6.2, a arquitetura de um sistema com o objetivo de ilustrar a representação hierárquica de conhecimento a partir de fórmulas do λ-cálculo tipado. A arquitetura se fundamenta em três passos, denominados passos elementares. Estes passos são a ESCOLHA, a EXECUCAO e a ORGANIZACAO.



Arquitetura EEO.
FIGURA 6.2

6.3.1 PASSOS ELEMENTARES DA ARQUITETURA EEO.

A arquitetura é denominada EEO devido aos três passos elementares: ESCOLHA, EXECUCAO e ORGANIZACAO, que são representados no esquema pelas setas que unem os λ-programas

λ-P,

λ-P APOS A ESCOLHA, indicado por λ-PE e

NOVO λ-P, indicado por λ-PN.

Na figura 6.2, as três setas rotuladas com "ESCOLHA", "EXECUCAO" e "ORGANIZACAO" representam três tipos de funções, ou passos elementares da arquitetura EEO. A sequência de execução dos passos ou funções ocorre conforme é descrito a seguir.

(i) Inicialmente, há a ESCOLHA de uma ação a ser executada pelo sistema. Isto é, há a escolha de uma função Φ a ser executada.

(ii) Em seguida, é determinado como a função Φ deve ser EXECUTADA.

(iii) A função Φ, que representa a ação escolhida, é EXECUTADA.

(iv) Finalmente, há uma REORGANIZACAO do conhecimento do sistema a partir dos resultados da execução de Φ. Tem-se a ORGANIZACAO do conhecimento do sistema a partir dos resultados obtidos pela execução da ação escolhida anteriormente.

A proposta da sequência de execução dos passos elementares baseia-se na representação de conhecimento em sistemas de controle de processos onde é comum uma sequência de eventos análoga à descrita anteriormente. Deve-se também observar que a análise da representação hierárquica de conhecimento não é comprometida se for considerada uma outra sequência para os passos elementares, ou mesmo outro conjunto de passos elementares. A sequência, arbitrária, de eventos pode ser modificada e o estudo da representação hierárquica de conhecimento que se segue pode ser realizada analogamente. Considera-se a seguir, separadamente, a descrição de cada função que compõe os passos elementares.

6.3.1.1 ESCOLHA.

Uma função é do tipo escolha quando sua execução determina ou escolhe uma outra função Φ, que deve ser executada posteriormente. A função do tipo escolha tem como domínio o λ-programa λ-P, que é uma lista de fórmulas onde estão armazenadas todas as informações pertinentes ao domínio de conhecimento do sistema. Nesta lista estão armazenadas todas as fórmulas disponíveis no sistema e o conhecimento que permite classificá-las como executáveis ou não executáveis, em um determinado instante. Assim, a função escolha utiliza este conhecimento e escolhe uma função Φ a ser executada. A função escolhida é explicitada em λ-*P*, obtendo-se o λ-programa após a escolha, in icado por λ-*PE*.

6.3.1.2 EXECUCAO.

Uma função é do tipo execução quando sua execução determina o modo de execução da função Φ selecionada anteriormente. Ela possui como domínio o λ-programa λ-*PE* e como contradomínio λ-*PN*. O conhecimento resultante da execução de Φ é armazenado no λ-programa λ-*PN*.

6.3.1.3 ORGANIZACAO.

Uma função é do tipo organização quando sua execução determina uma atualização, do conhecimento representado no sistema. Nesta reorganização, a função considera todos os conhecimentos armazenados nos λ-programas λ-*P*, λ-*PE* e λ-*PN*. Como resultado de sua execução, tem-se o λ-programa λ-*P* atualizado, a partir do qual pode-se escolher outras funções, iniciando um novo ciclo de escolha, execução e organização.

Além dos λ-programas que compõem um sistema com arquitetura EEO, tem-se os elementos do sistema real sobre o qual o sistema está atuando.

6.3.2 SISTEMA REAL.

O sistema real representa o sistema que é objeto de ações ou análises. No caso de um sistema que executa, por exemplo, diagnóstico médico, o sistema real é o paciente. Se o sistema executa controle de processos, o processo propriamente dito é o sistema sob controle.

6.3.3 NOVO ESTADO DO SISTEMA REAL.

O sistema determina ações que são executadas sobre o sistema real. O sistema real obtido após a execução da ação é representado por novo sistema real.

6.3.4 ELEMENTOS FISICOS

As funções do tipo escolha, execução e organização e os λ-programas λ-*P*, λ-*PE* e λ-*PN* são fórmulas do λ-cálculo tipado. Correspondendo à função do tipo execução, representada por uma fórmula do λ-cálculo tipado, tem-se uma ação real, que determina modificações no sistema real. Há portanto dois tipos de ações: uma ação representada por uma função, ou fórmula do λ-cálculo tipado, e uma ação física, que determina novos resultados no problema físico e não em seu modelo.

A ligação entre o λ-programa λ-*PN* e o novo estado do sistema real é feita através da interface **0** do sistema. Um sistema inteligente, que possui a capacidade de aprender os resultados de suas ações, deve utilizar a interface **0**. A aprendizagem é feita a partir do novo estado do sistema real. Os resultados desta aprendizagem são transferidos ao λ-programa λ-*PN* fazendo parte do resultado obtido após a execução da função do tipo execução. Concluindo, **0** determina, a partir de um contexto real, físico, um modelo representado por fórmulas do λ-cálculo tipado. Neste trabalho não se considera a análise das fórmulas que compõem a interface **0**.

A representação hierárquica de conhecimento em uma arquitetura EEO é realizada por programas com critério $(\lambda\text{-}P, \mathbb{D})$. Na próxima seção, as fórmulas de $\lambda\text{-}P$ são definidas a partir dos fundamentos do λ-cálculo tipado. São definidas as funções que compõem o λ-programa $\lambda\text{-}P$.

6.4 O λ-PROGRAMA $\lambda\text{-}P$.

A seguir, define-se as funções da arquitetura EEO, que compõem o λ-programa $\lambda\text{-}P$. Estas funções são fórmulas definidas a partir das noções fundamentais do λ-cálculo tipado. Obtém-se uma lista de fórmulas, hierarquizadas conforme o algoritmo da hierarquia completa. O resultado é uma representação hierárquica de conhecimento.

As funções do tipo escolha, do tipo execução e do tipo organização com diferentes complexidades são definidas a partir do conceito de parâmetro.

6.4.1 DEFINICAO. Parametro.

Uma fórmula Φ_α é um parâmetro se e somente se $\alpha \in \mathbb{F}$.

***

6.4.2 DEFINICAO. k-meta-escolha.

Uma k-meta-escolha ou função do tipo escolha k-E, é uma função definida por

(i)

$$k\text{-}E = \lambda Y_1 \ldots \lambda Y_m \; \lambda X_1 \ldots \lambda X_n \; (A \; F_1 \ldots F_s)$$

onde X_i, $1 \leq i \leq n$, são parâmetros e

$$\mathbf{tipo}[Y_i] = (\alpha_1 \succ \ldots \succ \alpha_p \succ \gamma),$$

$$\mathbf{tipo}[(A \; F_1 \ldots F_s)] = \mathbf{tipo}[Y_i]$$

para $p > 0$, $1 \leq i \leq m$, e

$$k\text{-}E =^g k$$

(ii) Dada uma lista

$$[\Phi_1,\ldots,\Phi_m,B_1,\ldots,B_n]$$

compatível com k-E, então

$$k\text{-}E\ \Phi_1\ldots\Phi_m\ B_1\ldots B_n = \Phi_j$$

para algum j $1\leq j\leq m$.

***

Observe que k-E escolhe uma fórmula Φ_j no conjunto

$$(\Phi_1,\ldots,\Phi_m)$$

baseada no conjunto de parâmetros

$$(B_1,\ldots,B_n).$$

Como X_i, $1\leq i\leq n$, é um parâmetro, e

$$k\text{-}E =^g k$$

então

$$Y_i =^n k - 1$$

pois

$$\mathbf{tipo}[Y_i] = \mathbf{tipo}[Y_j]$$

onde $1\leq i,j\leq m$. A k-meta-escolha k-E escolhe uma açáo representada pela funçáo Φ_j, cujo nível é igual a k - 1.

Os critérios que decidem sobre a escolha, no conjunto de funçõeS

$$(\Phi_1\ldots\Phi_m),$$

são determinados pelos parâmetros

$$B_1\ldots B_n.$$

Isto é, os critérios que definem a escolha são representados por fórmulas associadas a símbolos para tipo fundamentais.

Deve-se enfatizar que o nível das fórmulas Φ_i $1\leq i\leq m$ é igual ao grau da funçáo k-E menos uma unidade. Observe também que o nível de k-E é superior ao nível das fórmulas Φ_i $1\leq i\leq m$.

6.4.3 DEFINICAO. k-meta-execucao.

Uma k-meta-execução, ou função do tipo execução k-A, é uma função definida como se segue,

$$k\text{-}A = \lambda Z\ \lambda X_1 \ldots \lambda X_n\ (Z)$$

onde X_i, $1 \leq i \leq n$, são parâmetros.

$$\mathbf{tipo}[Z] = (\alpha_1 \succ \ldots \succ \alpha_p \succ \gamma),\ p \rangle 0$$

e

$$k\text{-}A =^{g} k$$

***

Dada uma lista

$$[\Phi,\ B_1, \ldots, B_n]$$

compatível com k-A. Se

$$k\text{-}A\ \Phi\ B_1 \ldots B_n = \Psi$$

isto é,

$$\begin{aligned} k\text{-}A\ \Phi\ B_1 \ldots B_n &= (\lambda Z\ \lambda X_1 \ldots \lambda X_n\ (Z))\Phi\ B_1 \ldots B_n \\ &= (\lambda X_1 \ldots \lambda X_n\ \Phi)\ B_1 \ldots B_n \\ &= \Psi \end{aligned}$$

então a fórmula Ψ é obtida a partir de Φ, substituindo cada ocorrência livre das variáveis

$$X_1, \ldots, X_n$$

em Φ, pelas fórmulas

$$B_1, \ldots, B_n$$

respectivamente. A função execução k-A modifica a fórmula Φ, substituindo as variáveis

$$X_1, \ldots, X_n,$$

pelos parâmetros

$$B_1, \ldots, B_n,$$

respectivamente. k-A determina a maneira como a função Φ deve ser considerada. Os critérios que decidem esta condição são determinados pelos parâmetros

$$B_1, \ldots, B_n.$$

Analogamente à função escolha, as variáveis X_i $1 \leq i \leq n$ são parâmetros. Logo, como

$$k\text{-}A =^{g} k$$

então o nível da fórmula Φ é igual ao grau da função k-A menos uma unidade. Isto é,

$$\Phi =^{n} k-1$$

Observe que o nível de k-A é superior ao nível de Φ, ou seja,

$$k\text{-}A >^{n} \Phi$$

6.4.4 DEFINICAO. k-meta-organizacao.

Uma k-meta-organização, ou função do tipo organização k-ORG, é uma função definida por

$$k\text{-}ORG = \lambda Z\ \lambda Y_1 \ldots \lambda Y_m\ \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_s)$$

onde X_i, $1 \leq i \leq n$, são parâmetros e

$$\mathbf{tipo}[(A\ F_1 \ldots F_s)] = \gamma,$$

$$\mathbf{tipo}[Z] = \gamma,$$

$$\mathbf{ordem}[\ \mathbf{tipo}[Y_i]\ ] < \mathbf{ordem}[\gamma],\ 1 \leq i \leq m$$

e

$$k\text{-}ORG =^{g} k$$

onde

$$k - 1 = \mathbf{ordem}[\gamma]$$

***

Dada uma lista de fórmulas

$$[\Phi, A_1, \ldots, A_m, B_1, \ldots, B_n]$$

compatível com k-ORG. Se

$$k\text{-}ORG\ \Phi\ A_1 \ldots A_m\ B_1 \ldots B_n = \Psi$$

então Ψ representa uma reorganização do conhecimento representado pela fórmula Φ. Esta reorganização baseia-se no conhecimento representado pelas fórmulas

$$A_1, \ldots, A_m$$

e pelos parâmetros

$$B_1, \ldots, B_n.$$

Uma k-meta-organização é uma função cuja execução determina uma reorganização do conhecimento do sistema a partir de um novo conhecimento. Um programa que executa, por exemplo, transformações sobre outros programas [Stepankova, 1984] é uma função do tipo organização.

Novamente, esta função possui um nível hierárquico superior aos de seus argumentos e o seu grau é igual ao nível do conhecimento a ser reorganizado mais uma unidade. Se

$$k\text{-ORG} =^{g} k$$

então

$$\Phi =^{n} k - 1$$

Propõe-se que as funções escolha, execução e organização pertençam ao λ-programa λ-*P*. Tem-se funções escolha, execução e organização de diversas complexidades. Propõe-se também que λ-*P* seja um conjunto hierarquizado conforme uma hierarquia completa. Neste sentido, o conhecimento representado em λ-*P* é classificado em conhecimento básico, em meta-conhecimento, em meta-meta-conhecimento e assim por diante. O conhecimento de um mesmo grau é classificado conforme o nível. Finalmente, o conhecimento representado em um mesmo grau e nível é classificado conforme a descrição total. Caso a hierarquização das fórmulas de λ-*P* seja uma outra, a análise que se segue é análoga. O λ-programa λ-*P* é obtido aplicando o algoritmo da hierarquia completa sobre o conjunto de fórmulas a seguir.

$$\mathbf{A} = (\ 1\text{-E},\ 2\text{-E},\ \ldots,\ \mathbf{max}\text{-E},$$

$$1\text{-A},\ 2\text{-A},\ \ldots,\ \mathbf{max}\text{-A},$$

$$1\text{-ORG},\ 2\text{-ORG},\ \ldots,\ \mathbf{max}\text{-ORG},$$

$$\Phi_1,\ \Phi_2,\ \ldots,\ \Phi_n\ )$$

onde

$$\Phi_1,\ \Phi_2,\ \ldots,\ \Phi_n$$

são fórmulas que representam um conhecimento não representado pelos passos elementares. Tem-se também que "**max**" é o grau maximo do conhecimento representado no sistema.

Na próxima seção propõe-se a definição do critério de derivação **D**. Este critério determina como as funções de λ-*P* são executadas.

6.5 O CRITERIO DE DERIVACAO $\mathbb{D}$.

Propõe-se a seguir um critério de derivação $\mathbb{D}$, que compõe o programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. O critério de derivação $\mathbb{D}$ associado a um λ-programa $\lambda\text{-}P$ estabelece como as funções de $\lambda\text{-}P$ são executadas. Este critério escolhe os argumentos das funções de $\lambda\text{-}P$ e na sua definição são consideradas variáveis de controle associadas a estas funções. A escolha realizada pelo critério depende do valor das variáveis de controle presentes no sistema. Tem-se dois tipos de variáveis de controle: a "palavra de controle" e o "grau ativo do sistema", que são definidos a seguir.

6.5.1 DEFINICAO. Palavra de controle.

Dada uma função $\Psi \in \lambda\text{-}P$, então o parâmetro

$$\mathcal{Y}_{pc\text{-}\Psi} \in VAR$$

é a palavra de controle associada a Ψ. Convenciona-se que há apenas dois valores possíveis para a palavra de controle, que são:

$$\mathcal{Y}_{pc\text{-}\Psi} = 1$$

ou

$$\mathcal{Y}_{pc\text{-}\Psi} = \emptyset$$

***

Uma outra variável de controle é a variável que determina o grau ativo do sistema, definida a seguir.

6.5.2 DEFINICAO. Grau ativo do sistema.

A variável grau ativo do sistema é o parâmetro indicado por

$$\mathcal{X}_{grau}$$

Esta variável varia no intervalo,

$$1 \leq \mathcal{X}_{grau} \leq max$$

onde "**max**" é o grau máximo das fórmulas de $\lambda\text{-}P$.

***

As variáveis de controle são utilizadas conforme é analisado na próxima seção.

6.5.3 UTILIZACAO DAS VARIAVEIS DE CONTROLE.

Para determinar o sequenciamento da execução das funções de λ-P são utilizadas as variáveis de controle. Dependendo dos valores destas variáveis, os passos elementares do sistema são executados ou não. Considera-se a seguinte convenção:

(i) dada uma função Ψ, onde

$$\Psi =^{g} k,$$

(ii) para que Ψ seja executada é necessário que se tenha

$$\gamma_{pc\text{-}\Psi} = 1$$

e

$$\aleph_{grau} = k.$$

A função k-E, por exemplo, só é executada quando se tem

$$\gamma_{pc\text{-}kE} = 1,$$

e

$$\aleph_{grau} = k$$

6.5.4 DEFINICAO. Criterio de derivacao.

O critério de derivação $\mathbb{D}$, proposto a seguir, fundamenta-se no valor das variáveis de controle presentes em λ-P. Dependendo do valor destas variáveis, tem-se várias sequências possíveis para o sequenciamento da execução das funções de λ-P. Estas sequências são analisadas na próxima seção.

Dada uma fórmula

$$\Psi = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m),$$

onde

$$\Psi =^{g} k,$$

o critério de derivação associado a Ψ é definido por:

(i) se

$$\gamma_{pc\text{-}\Psi} = \emptyset$$

então,

$$\mathbb{D}[\Psi, \lambda\text{-}P] = \emptyset$$

(ii) se

$$\mathcal{V}_{pc\text{-}\Psi} = 1,$$

e

$$\mathcal{K}_{grau} = k,$$

então

$$\mathbb{D}[\Psi, \lambda\text{-}P] = [B_1 \ldots B_n]$$

onde

$$[B_1 \ldots B_n]$$

é uma lista de fórmulas compatíveis com Ψ e a fórmula

$$B_i$$

é a primeira fórmula em λ-P tal que

$$\mathbf{tipo}[B_i] = \mathbf{tipo}[X_i].$$

$$B_i \neq B_{i-1}$$

Caso não exista uma fórmula $B_i \in \lambda$-P, como indicado acima, faz-se

$$B_i = X_i$$

***

Observe que

$$\mathbb{D}[\Psi, \lambda\text{-}P] \neq \mathrm{NIL}$$

$$\mathcal{V}_{pc\text{-}\Psi} = 1$$

e

$$\mathcal{K}_{grau} = k,$$

Analisa-se na próxima seção o sequenciamento das execuções das funções de λ-P. Este sequenciamento fundamenta-se na definição do critério de derivação $\mathbb{D}$ e na hierarquia da fórmulas de λ-P.

6.6 SEQUENCIAMENTO DAS EXECUCOES DAS FUNCOES DE λ-P.

A utilização do conhecimento representado em um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$ é determinada pelo sequenciamento da execução das funções do λ-programa λ-P. Propõe-se a seguir um sequenciamento determinado pelo critério de derivação $\mathbb{D}$, juntamente com o valor das palavras de controle e do grau ativo do sistema. Esta proposta tem como objetivo ilustrar a utilizacao do λ-cálculo tipado na representação hierárquica de conhecimento.

Conforme é proposto na seção anterior, o critério de derivação $\mathbb{D}$ é função do valor das variáveis de controle presentes em λ-P. Dependendo do valor destas variáveis, tem-se várias seqüências possíveis para o sequenciamento da execução das funções de λ-P. Pode-se considerar inicialmente as funções cujo grau de complexidade é 1, em seguida cujo grau é 2 e assim por diante. Pode-se considerar também um sequenciamento inverso. Isto é, funções cujo grau de complexidade é "p", em seguida funções cujo grau é "p-1" e assim por diante. A sequência de execução dos passos elementares com grau fixo é analisada a seguir.

6.6.1 SEQUENCIAMENTO DOS PASSOS ELEMENTARES.

Propõe-se a seguir uma sequência de execução para os passos elementares de um determinado grau k. Este sequenciamento baseia-se no grau das funções da arquitetura EEO e nas suas relações de complexidade. A sequência é indicada na figura 6.3. Deve-se observar que a escolha de uma outra sequencia, diferente da apresentada na figura 6.3, não restringe a análise que se segue.

Na figura 6.3, a escolha é representada pela execução da função do tipo escolha k-E, que é uma fórmula de grau k. Esta função escolhe Φ_i e em seguida a função do tipo execução k-A determina como Φ_i deve ser executada. Isto é, a derivação de k-A em

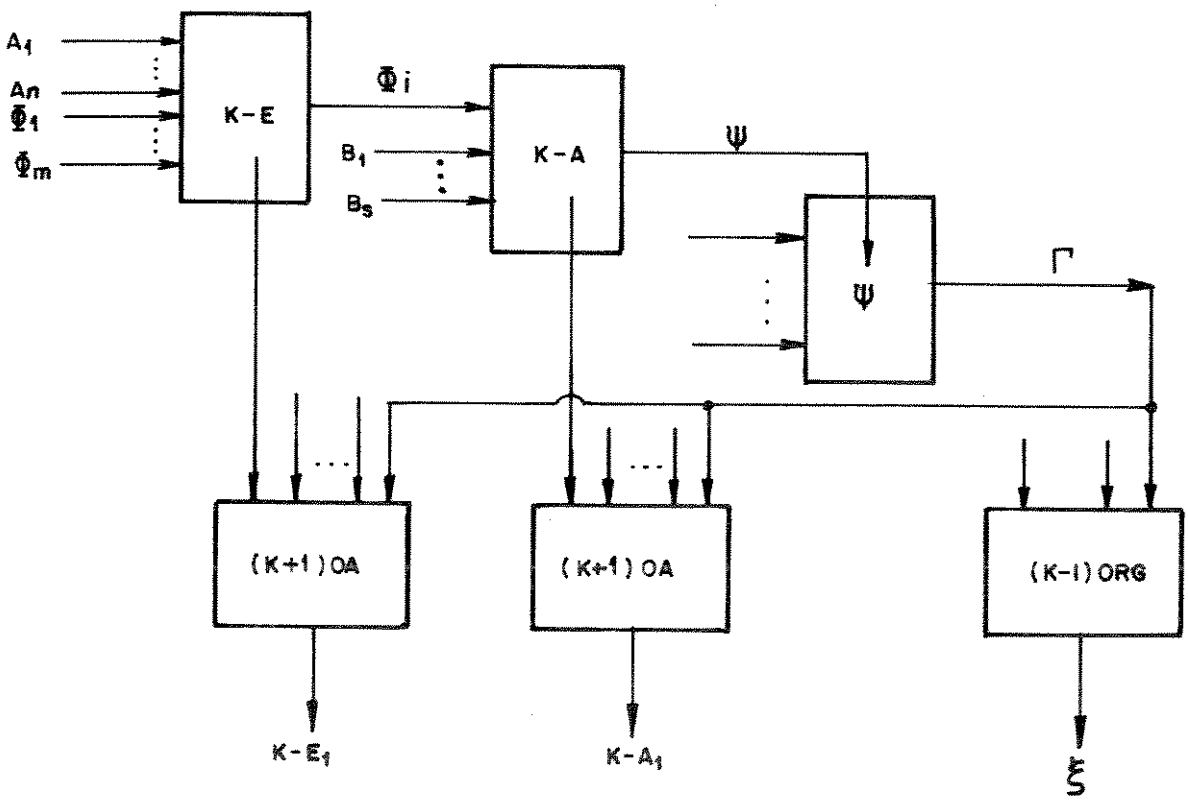
$$[\Phi_i, B_1 \ldots B_s]$$

é a nova função Ψ. Esta função , por sua vez, é executada, obtendo-se como resultado a fórmula Γ. Portanto, há a escolha de uma ação e sua execução obtendo-se o resultado representado pela fórmula Γ. Finalmente, a partir do conhecimento representado em Γ, há uma reorganização do conhecimento do sistema. Tem-se a execução de três funções do tipo organização .

(a) (k-1)-ORG, que reorganiza o conhecimento representado por Γ, tendo como resultado a fórmula ξ.

(b) (k+1)-OE, que reorganiza o conhecimento representado pela função escolha k-E. O resultado desta reorganização é a nova função escolha representada por $k\text{-}E_1$.

(c) (k+1)-OA, que reorganiza o conhecimento representado pela função execução k-A. Esta função é análoga a (k+1)-OE e o seu resultado é a nova função execução, indicada por $k\text{-}A_1$.



Sequencia de execucao dos passos elementares.
FIGURA 6.3

Observe que as funções do tipo escolha e do tipo execução, indicadas na figura 6.3, são funções de grau k, enquanto que as funções do tipo organização possuem graus iguais a k-1, e k+1. Como

$$k\text{-}E =^{g} k,$$

então

$$\Phi_i =^{n} k-1.$$

Tem-se também que

$$\Psi =^{n} \Phi_i,$$

logo

$$\Psi =^{n} k-1$$

Assim, conforme o teorema 5.3.1, o resultado da execução de Ψ, representado pela fórmula Γ, é tal que

$$\Gamma \leq^{n} k-2$$

Como a função Γ deve ser um argumento da função p-ORG, a função organização p-ORG, é tal que,

$$p\text{-}ORG \geq^{g} k-1$$

Considera-se então o caso extremo em que

$$p\text{-}ORG =^{g} k-1$$

isto é,

$$p = k-1$$

A escolha do caso extremo não compromete a análise que se segue. Tal análise seria semelhante caso fosse escolhido outro valor para p. O valor exato de "p" depende de cada representação específica de conhecimento, podendo ocorrer o caso em que

$$p > k-1$$

Por outro lado, como

$$k\text{-}E =^{g} k$$

então

$$k\text{-}E \geq^{n} k$$

logo a função q-OE, que reorganiza o conhecimento representado pela função escolha k-E deve ser tal que

$$q\text{-}OE \geq^{g} k+2$$

Considera-se novamente o caso extremo em que

$$q = k+2$$

De uma maneira analoga, conclui-se o grau da função (k+2)-OA.

6.6.2 SEQUENCIAMENTO DA EXECUCAO DE FUNCOES DE DIFERENTES GRAUS.

Os sequenciamentos das execuções dos passos elementares de diferentes graus sao denominados, neste trabalho, de **MODOS PRINCIPAIS** de operação do sistema. Os **MODOS PRINCIPAIS** de operação são definidos como a composição de outros mais simples, os **MODOS FUNDAMENTAIS.** A seguir, são analisados os modos fundamentais e posteriormente, suas diferentes composições.

6.6.2.1 MODOS FUNDAMENTAIS.

Os modos fundamentais se classificam em duas classes: **MODO FUNDAMENTAL DESCENDENTE** e **MODO FUNDAMENTAL ASCENDENTE.**

(i) MODO FUNDAMENTAL DESCENDENTE.

Neste modo, inicia-se pela execução das funções de grau máximo seguindo-se a execução das de graus inferiores.

(i) MODO FUNDAMENTAL ASCENDENTE.

Este modo de operação é o inverso do anterior. Inicia-se pela execução das funções de grau mínimo seguindo-se a execução das de graus superiores.

A partir da composição destes dois modos fundamentais são definidos os modos principais de operação do sistema.

6.6.2.2 MODOS PRINCIPAIS

Os modos principais de operação de um sistema com arquitetura EE0 são obtidos a partir de combinações dos modos fundamentais. Ha inúmeras possibilidades para tais combinações e algumas delas são analisadas a seguir.

(i) MODO PRINCIPAL ASCENDENTE.

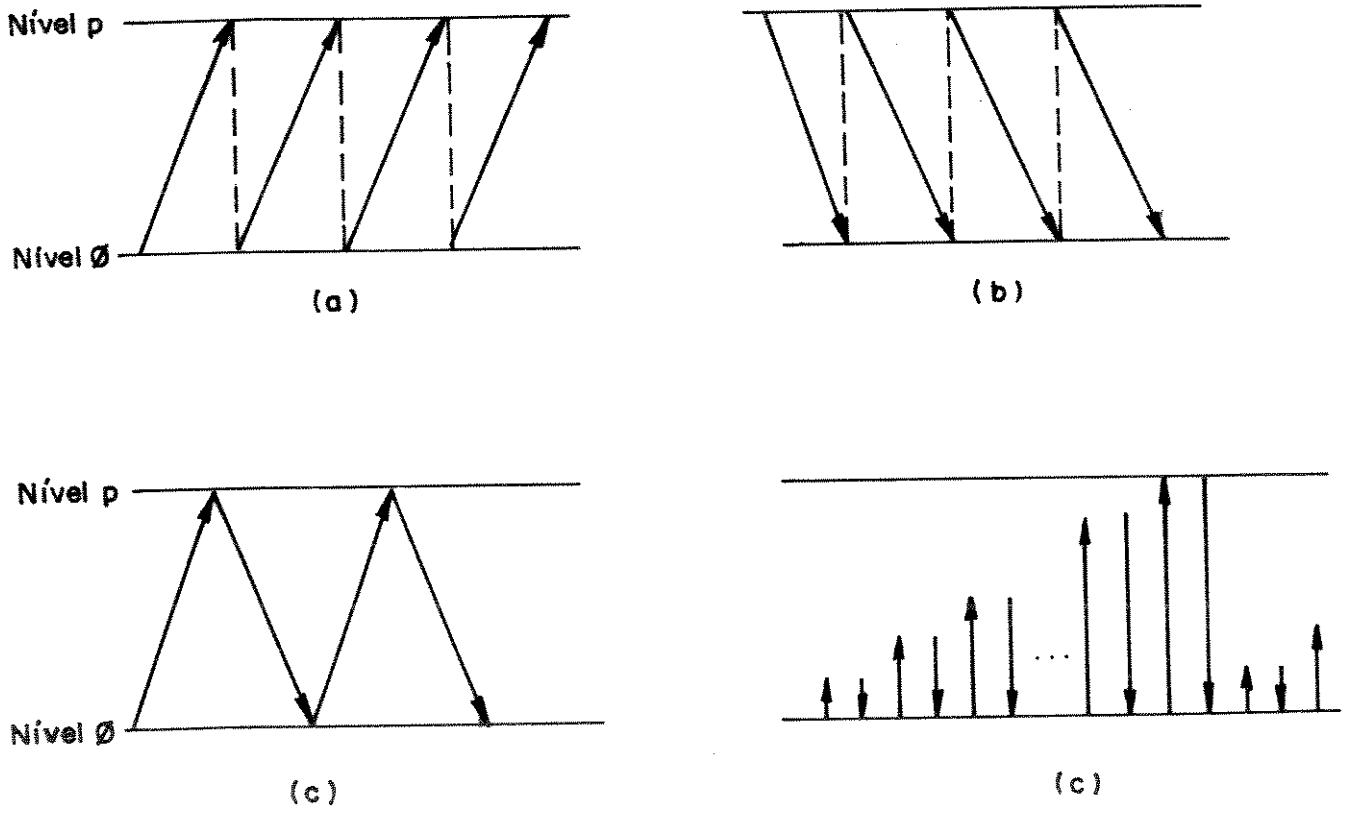
Neste caso, o sequenciamento da execução das funções de diferentes graus é feita comecando pelas funções de grau 0, em seguida as de grau 1, até as de grau máximo igual a p. Após a utilização do conhecimento de grau p, o sistema volta a utilizar o conhecimento de grau 0, repetindo o ciclo. Este modo é a repetição do modo fundamental ascendente, figura 6.4(a).

(ii) MODO PRINCIPAL DESCENDENTE.

O modo principal descendente é o oposto do modo principal ascendente. Inicia-se pela execução das funções de grau máximo p, sendo seguida pela execução das de grau p-1, até as do grau 0. O sistema volta então a executar as funções de grau p, repetindo o ciclo. Este modo é a repetição do modo fundamental descendente, figura 6.4(b).

(iii) OUTROS MODOS PRINCIPAIS.

Além dos modos principais de operação, apresentados nos itens anteriores, há inúmeros outros modos principais. Eles são formados por diferentes composições dos modos fundamentais. Nas figuras 6.4 (c) e 6.4 (d) são indicados os modos principais ascendente-descendente e ascendente conservador.



Modos principais de operacao de um sistema.
FIGURA 6.4

6.6.2.3 REPRESENTACAO DOS MODOS DE OPERACAO.

Propõe-se a seguir a representação dos modos de operação de um sistema com arquitetura EEO. Na proposta que se segue, o conhecimento que determina os modos de operação é representado pelas variáveis,

$$M_{modo} \quad e \quad MP_{modo}$$

onde

$$modo=(list\ int),$$

$$\langle list,1\rangle \in \mathbb{C}.$$

A primeira variável representa o modo fundamental e a segunda o modo principal de operação do sistema. $\mathbb{M}_{modo}$ é uma lista com três números inteiros

$$[Z,X,Y],$$

onde

(i)

$$\mathbb{M}_{modo} = [0,p,s],\ p>s,$$

indica que o modo de operação é o descendente e o sistema deve comecar executando as funções de grau igual a "p", depois com grau igual a "p-1" e assim por diante, até atingir o conhecimento de grau igual a "s".

(ii)

$$\mathbb{M}_{modo} = [1,s,p],\ s<p.$$

Neste caso, o modo de operação é o ascendente e o sistema deve começar utilizando o conhecimento de grau igual a "s", depois os de grau "s+1" e assim por diante, até atingir o conhecimento de grau "p".

A variável $\mathbb{MP}_{modo}$ é uma lista com dois inteiros

$$[X,Y]$$

onde

(a)

$$\mathbb{MP}_{modo} = [0,p],\ p>1,$$

indica que o modo principal de operação é o descendente com grau máximo igual a "p". Isto é, tem-se a repetição do modo fundamental descendente,

$$\mathbb{M}_{modo} = [0,p,0],$$

indicado na figura 6.4(b).

(b)

$$\mathbb{MP}_{modo} = [1,p],\ p>1.$$

Neste caso, tem-se o modo principal ascendente, com a repetição do modo fundamental ascendente,

$$\mathbb{M}_{modo} = [1,0,p],$$

indicado na figura 6.4(a).

(c)

$$\mathbb{MP}_{modo} = [2,p],\ p>1,$$

indica que o modo principal é o ascendente-descendente, com a repetição dos modos fundamentais

$$\mathbb{M}_{modo} = [1,0,p]$$

e

$$\mathbb{M}_{modo} = [0,p,0],$$

respectivamente.

(d)

$$\mathbb{MP}_{modo} = [3,p],\ p>0,$$

indica que o modo principal de operação é o ascendente conservador.

6.7 O CONTROLE DO SEQUENCIAMENTO DOS PASSOS ELEMENTARES.

A seguir analisa-se o controle do sequenciamento das execuções das funções que representam os passos elementares. Este controle é realizado por funções controladoras que determinam quando as funções, que representam os passos elementares, devem ser executadas. As funções controladoras atuam modificando os valores das palavras de controle

$$\mathbb{Y}_{pc-\Psi}$$

associadas aos passos elementares. Este controle evita, por exemplo, que uma função Ψ, que representa um passo elementar, seja executada indefinidamente pelo sistema. Observe que após a execução de uma função Ψ, para que ela não seja executada novamente, deve-se ter

$$\mathbb{Y}_{pc-\Psi} = 0.$$

Associadas a cada função Ψ, há duas funções,

$$\xi\Psi \text{ e } \Pi\Psi,$$

denominadas funções controladores de Ψ. Os resultados das execuções de $\xi\Psi$ e $\Pi\Psi$ determinam novos valores para as palavras de controle. A execução de $\xi\Psi$ faz

$$\gamma_{pc-\Psi} = 1$$

e o resultado de $\Pi\Psi$ faz

$$\gamma_{pc-\Psi} = \emptyset.$$

Para simplificar as definições é considerada a seguinte notação.

NOTACAO. Nas fórmulas que se seguem tem-se as seguintes equivalências entre os símbolos,

$$\lambda\gamma \equiv \lambda\gamma_{pc-kE}\,\lambda\gamma_{pc-kA}\,\lambda\gamma_{pc-(k-1)ORG}\,\lambda\gamma_{pc-(k+2)OE}\,\lambda\gamma_{pc-(k+2)OA}\,\lambda\aleph_{grau}$$

$$\gamma \equiv [\gamma_{pc-kE}, \gamma_{pc-kA}, \gamma_{pc-(k-1)ORG}, \gamma_{pc-(k+2)OE}, \gamma_{pc-(k+2)OA}, \aleph_{grau}]$$

$$\Theta \equiv (pc\text{-}kE \succ pc\text{-}kA \succ pc\text{-}(k-1)ORG \succ pc\text{-}(k+2)OE \succ pc\text{-}(k+2)OA \succ grau)$$

***

6.7.1 DEFINICAO. Funcoes controladoras.

As funções controladoras associadas aos passos elementares são definidas a seguir.

(i) funcao escolha.

$$\Pi kE = \lambda\gamma\ (\Pi E\ \gamma)$$

onde,

$$\Pi E\ \gamma = \begin{cases} \emptyset & \text{se } \gamma=[1,0,0,0,0,k] \\ \gamma_{pc-kE} & \text{se } \gamma\neq[1,0,0,0,0,k] \end{cases}$$

$$tipo[\Pi kE] = [\ \Theta \succ pc\text{-}kE\ ]$$

$$\xi kE = \lambda\gamma\ (\xi E\ \gamma)$$

onde,

$$\xi E\ \gamma = \begin{cases} 1 & \text{se } \gamma=[0,0,1,1,1,k] \\ \gamma_{pc-kE} & \text{se } \gamma\neq[0,0,1,1,1,k] \end{cases}$$

$$tipo[\xi kE] = [\ \Theta \succ pc\text{-}kE\ ]$$

(ii) funcao execucao.

$$\Pi kA = \lambda\Psi\ (\Pi A\ \Psi)$$

onde,

$$\Pi A\ \Psi = \begin{cases} \emptyset & \text{se } \Psi=[0,1,0,0,0,k] \\ \Psi_{pc\text{-}kA} & \text{se } \Psi\neq[0,1,0,0,0,k] \end{cases}$$

$$tipo[\Pi kA] = [\ \Theta \succ pc\text{-}kA\ ]$$

$$\xi kA = \lambda\Psi\ (\xi A\ \Psi)$$

onde,

$$\xi A\ \Psi = \begin{cases} 1 & \text{se } \Psi=[1,0,0,0,0,k] \\ \Psi_{pc\text{-}kA} & \text{se } \Psi\neq[1,0,0,0,0,k] \end{cases}$$

$$tipo[\xi kA] = [\ \Theta \succ pc\text{-}kA\ ]$$

(iii) funcao oganizacao "ORG".

$$\Pi(k-1)ORG = \lambda\Psi\ (\Pi ORG\ \Psi)$$

onde,

$$\Pi ORG\ \Psi = \begin{cases} \emptyset & \text{se } \Psi=[0,0,1,1,1,k] \\ \Psi_{pc\text{-}(k-1)ORG} & \text{se } \Psi\neq[0,0,1,1,1,k] \end{cases}$$

$$tipo[\Pi(k-1)ORG] = [\ \Theta \succ pc\text{-}(k-v)ORG\ ]$$

$$\xi(k-1)ORG = \lambda\Psi\ (\xi ORG\ \Psi)$$

onde,

$$\xi ORG\ \Psi = \begin{cases} 1 & \text{se } \Psi=[0,0,0,0,0,k] \\ \Psi_{pc\text{-}(k-1)ORG} & \text{se } \Psi\neq[0,0,0,0,0,k] \end{cases}$$

$$tipo[\xi(k-1)ORG] = [\ \Theta \succ pc\text{-}(k-v)ORG\ ]$$

(iv) funcao organizacao "OE".

$$\Pi(k+2)OE \equiv \lambda\Psi\ (\Pi OE\ \Psi)$$

onde,

$$\Pi OE\ \Psi = \begin{cases} \emptyset & \text{se } \Psi\ [0,0,1,1,1,k] \\ \Psi_{pc-(k+2)OE} & \text{se } \Psi \neq [0,0,1,1,1,k] \end{cases}$$

$$tipo[\Pi(k+2)OE] = [\ \Theta \succ pc-(k+2)OE\ ]$$

$$\xi(k+2)OE \equiv \lambda\Psi\ (\xi OE\ \Psi)$$

onde,

$$\xi OE\ \Psi = \begin{cases} 1 & \text{se } \Psi=[0,0,0,0,0,k] \\ \Psi_{pc-(k+2)OE} & \text{se } \Psi \neq [0,0,0,0,0,k] \end{cases}$$

$$tipo[\xi(k+2)OE] = [\ \Theta \succ pc-(k+2)OE\ ]$$

(v) funcao organizacao "OA".

$$\Pi(k+2)OA \equiv \lambda\Psi\ (\Pi OA\ \Psi)$$

onde,

$$\Pi OA\ \Psi = \begin{cases} \emptyset & \text{se } \Psi=[0,0,1,1,1,k] \\ \Psi_{pc-(k+2)OA} & \text{se } \Psi \neq [0,0,1,1,1,k] \end{cases}$$

$$tipo[\Pi(k+2)OA] = [\ \Theta \succ pc-(k+2)OA\ ]$$

$$\xi(k+2)OA \equiv \lambda\Psi\ (\xi OA\ \Psi)$$

onde,

$$\xi OA\ \Psi = \begin{cases} 1 & \text{se } \Psi=[0,0,0,0,0,k] \\ \Psi_{pc-(k+2)OA} & \text{se } \Psi \neq [0,0,0,0,0,k] \end{cases}$$

$$tipo[\xi(k+2)OA] = [\ \Theta \succ pc-(k+2)OA\ ].$$

***

Observe inicialmente, o símbolo para tipo associado às funções indicadas acima. A função ΠkE, por exemplo está associada ao símbolo para tipo,

$$tipo[\Pi kE] = (\Theta \succ pc\text{-}kE)$$

onde

$$\Theta \equiv (pc\text{-}kE \succ pc\text{-}kA \succ pc\text{-}(k\text{-}1)ORG \succ pc\text{-}(k+2)OE \succ pc\text{-}(k+2)OA \succ grau)$$

Isto significa que o resultado da aplicação de

$$\Pi kE$$

aos seus argumentos é uma fórmula associada ao símbolo para tipo

$$"pc\text{-}kE",$$

que é o símbolo associado à palavra de controle

$$\Psi_{pc\text{-}kE}"$$

A execução de ΠkE determina um novo valor para $\Psi_{pc\text{-}kE}"$. Observe ainda que as funções indicadas somente modificam o valor de uma palavra de controle quando se tem uma única configuração para os valores de todas as outras palavras de controle. A função ΠkE, por exemplo, faz

$$\Psi_{pc\text{-}kE} = \emptyset$$

somente quando se tem,

$$\Psi_{pc\text{-}kE} = 1,$$

$$\aleph_{grau} = k$$

e

$$\Psi_{pc\text{-}kA} = \Psi_{pc\text{-}(k\text{-}1)ORG} = \Psi_{pc\text{-}(k+2)OE} = \Psi_{pc\text{-}(k+2)OA} = \emptyset$$

Caso contrário, o valor de

$$\Psi_{pc\text{-}kE}$$

permanece com o mesmo valor. Isto é, ΠkE é uma identidade na variável

$$\Psi_{pc\text{-}kE}$$

Considere a figura 6.5 onde se tem o sequenciamento dos passos elementares da arquitetura EEO e a execução das funções controladoras. Nesta figura, indica-se a ativação e desativação das palavras de controle, dependendo das funções que acabam de ser executadas.

Na figura 6.5 estão indicadas somente as execuções que modificam os valores das palavras de controle. Observe o sequenciamento das execuções e os diferentes valores assumidos pela lista Ψ. Considera-se inicialmente

$$\Psi = (1,0,0,0,0,k)$$

Após a execução das funções ΠkE e ξkA, tem-se

$$\Psi = (0,1,0,0,0,k)$$

Com este valor de Ψ, a única função que modifica alguma palavra de controle é ΠkA, Após a execução de ΠkA é obtido,

$$\Psi = (0,0,0,0,0,k)$$

o que determina as execuções das funções $\xi(k-1)ORG$, $\xi(k+2)OE$ e $\xi(k+2)OA$, modificando Ψ para,

$$\Psi = (0,0,1,1,1,k)$$

Finalmente, a partir deste valor de Ψ, as funções $\Pi(k-1)ORG$, $\Pi(k+2)OE$, $\Pi(k+2)OA$ e ξkE são executadas, zerando as palavras de controle

$$\Psi_{pc-(k-1)ORG}$$

$$\Psi_{pc-(k+2)OE}$$

$$\Psi_{pc-(k+2)OA}$$

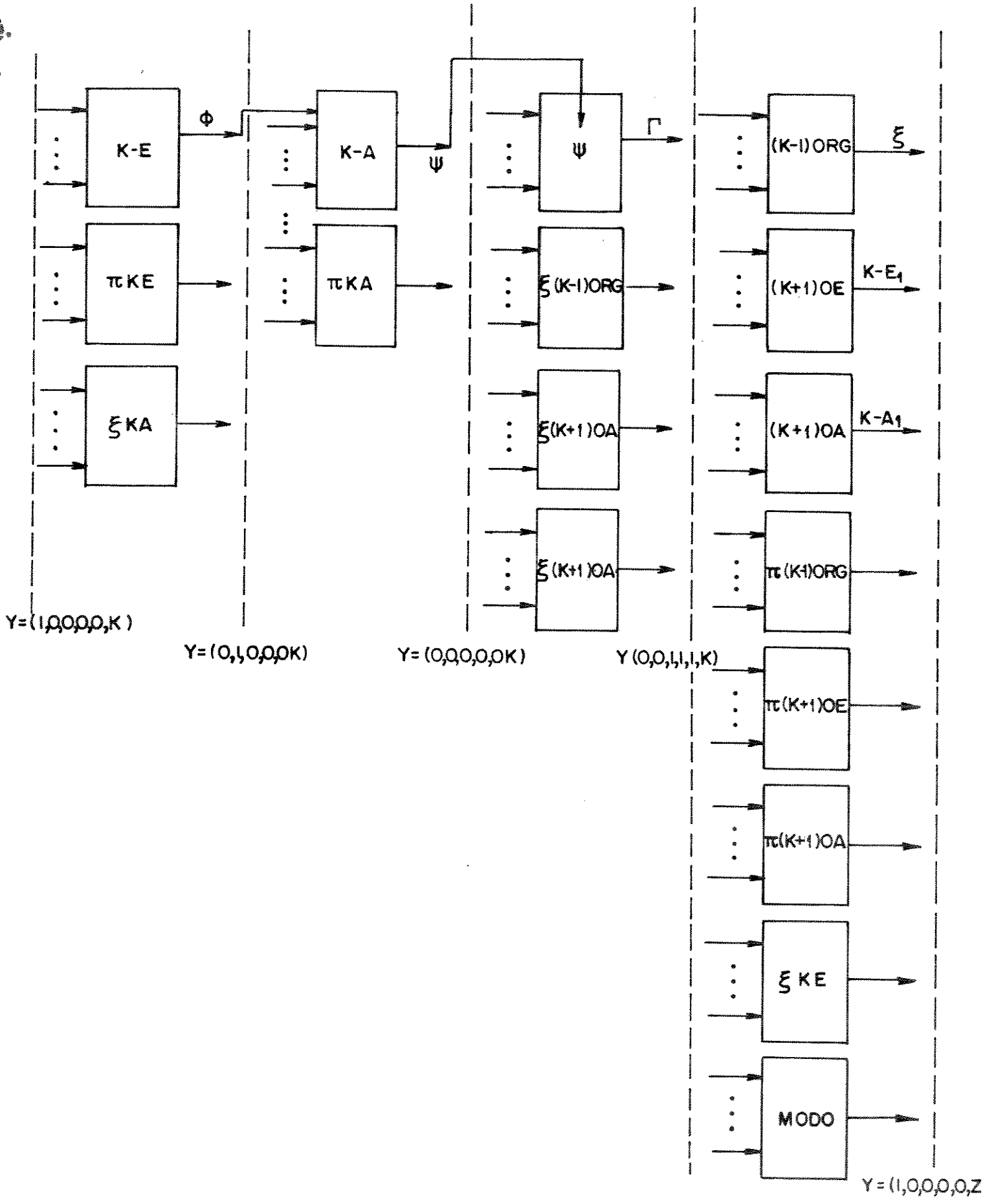
e fazendo

$$\Psi_{pc-kE} = 1.$$

A função **MODO**, indicada na figura 6.5, determina o valor do grau ativo

$$\aleph_{grau}$$

do sistema. Esta função é definida a partir de um algoritmo controlador analisado na proxima seção.



Sequenciamento da execucao dos passos elementares.

FIGURA 6.5

6.8 ALGORITMO CONTROLADOR DOS MODOS DE OPERACAO.

Propõe-se a seguir um algoritmo, que controla a utilização dos diferentes modos de operação do sistema. A partir dos valores das variáveis de controle

$$\mathbb{M}_{modo} \text{ e } \mathbb{MP}_{modo}$$

este algoritmo determina o valor do grau ativo do sistema e consequentemente, o grau das funções a serem executadas pelo sistema. O algoritmo controlador dos modos de operação é executado quando a função **MODO**, indicada na figura 6.5, é executada.

Define-se inicialmente a função **MODO**, e em seguida o algoritmo controlador dos modos de operação. Na análise do algoritmo considera-se a inicialização dos modos principais de operação do sistema, a descrição do algoritmo auxiliar fim de ciclo e do algoritmo controlador de modo de operação propriamente dito.

6.8.1 DEFINICAO. Funcao MODO.

A função modo é definida a seguir

$$\mathbf{MODO} = \lambda\Psi\ \lambda\mathbb{M}_{modo}\ \lambda\mathbb{MP}_{modo}\ (\ \Gamma\Gamma\ \Psi\ \mathbb{M}_{modo}\ \mathbb{MP}_{modo}\ )$$

onde,

$$\mathbf{tipo[MODO]} = (\Theta \succ \text{modo} \succ \text{modo} \succ \text{grau})$$

e $\Gamma\Gamma$ é uma função definida pelo algoritmo controlador dos modos de operação, definida na seção seguinte.

***

A função **MODO** determina o valor da variável $\mathbb{X}_{grau}$, que por sua vez determina qual o grau dos passos elementares que devem ser executados pelo sistema. Isto é, se o grau ativo do sistema é

$$\mathbb{X}_{grau} = k,$$

então o sistema deve executar os passos elementares cuja complexidade quanto ao grau das funções do tipo escolha e do tipo execução é igual a k. Observe na figura 6.5 que o valor de Ψ passa a ser:

$$\Psi = (1,0,0,0,0,\mathbb{Z}_{grau})$$

onde

$$\mathbb{Z}_{grau}$$

é o resultado da execução de **MODO**, o que determina o grau dos passos elementares que devem ser executados. No caso em que

$$\mathbb{Z}_{grau} = k,$$

então tem-se a sequência de execuções indicadas na figura 6.5.

6.8.2 O ALGORITMO.

Analisa-se a seguir um algoritmo controlador, que controla a utilização dos modos de operação do sistema. Este algoritmo determina o valor do grau ativo do sistema e consequentemente estabelece o grau de complexidade dos passos elementares que são executados a cada ciclo de operação do sistema. O algoritmo controlador de modo de operação determina o valor de

$$\mathbb{K}_{grau}$$

a partir dos valores das variáveis

$$\mathbb{M}_{modo}$$

e

$$\mathbb{MP}_{modo}$$

que indicam ao sistema os modos de operação. Assim, estas variáveis devem ser inicializadas para que o algoritmo possa ser executado. Indica-se a seguir as inicializações necessárias para a devida operação do sistema. Considera-se que o grau máximo de conhecimento representado no sistema é igual ao numero inteiro

$$"max",$$

Considera-se no restante desta análise, a seguinte notação.

NOTACAO: No sequenciamento das execuções das funções de λ-P, utiliza-se a variável t para representar estas funções em diferentes ciclos. Uma fórmula $\Psi \in \lambda$-P, no ciclo t é denotada como se segue,

$$\Psi(t)$$

Os ciclos representam as mudancas que ocorrem no sistema e a variável t pode assumir os valores:

$$0 \quad 0.5 \quad 1 \quad 1.2 \quad 2 \quad 2.5 \quad 3 \quad 3.5 \quad ...$$

onde

$$t=0$$

significa o início do ciclo numero 0,

$t=0.5$

o meio do ciclo numero 0,

$t=1$

o fim do ciclo numero 0 e o início do ciclo numero 1 e assim por diante. Deve-se esclarecer que t não representa o tempo pois, como é visto no decorrer do trabalho, é possível em um dado instante de tempo a presença de elementos em ciclos diferentes.

***

(i) INICIALIZACAO DO MODO PRINCIPAL ASCENDENTE.

Faça,

$$\mathbb{M}_{modo} = [1, 1, max]$$

$$\mathbb{MP}_{modo} = [1, max]$$

$$t=0$$

$$\mathbb{V} = (1,0,0,0,0,2)$$

Inicializa-se as variáveis que determinam o modo de operação do sistema. Faz-se t=0 em todos os elementos do sistema, sincronizando-os no início do ciclo 0 e também

$$\mathbb{V} = (1,0,0,0,0,2)$$

determinando o início da execução da função do tipo escolha

2-E(0).

Observe que neste caso,

$$\mathbb{K}_{grau} = 2.$$

e

$$t = 0$$

Com a execução de 2-E(0), o sistema começa pela utilização de um conhecimento cujo grau de complexidade é inferior. Quando uma escolha de grau 2 é executada, escolhe-se uma função de grau 1. Em seguida, a execução desta função modifica o conhecimento de grau zero. Isto é, a execução de 2-E(0) determina modificações no conhecimento de grau 0, ou seja, no conhecimento de grau inferior.

(ii) **INICIALIZACAO DO MODO PRINCIPAL DESCENDENTE.**

Faça,

$$M_{modo} = [0, max, 1],$$

$$MP_{modo} = [0, max]$$

$$t = 0$$

$$Y = (1,0,0,0,0,max)$$

Esta inicialização é análoga ao caso anterior. Entretanto, observe que é feito,

$$Y = (1,0,0,0,0,max)$$

determinando o início da execução da função do tipo escolha

max-E(0).

Observe que neste caso,

$$\aleph_{grau} = max.$$

e novamente

$$t = 0$$

Com a execução de **max**-E(0), o sistema inicia pela utilização de um conhecimento representado em graus hierárquicos superiores. Quando **max**-E(0) é executada, escolhe-se uma função de grau **max** - 1. Em seguida, a execução desta função modifica o conhecimento representado por funções de grau **max** -2. Portanto, a execução da função do tipo escolha **max**-E(0) determina modificações no conhecimento representado por fórmulas de graus superiores.

(iii) **INICIALIZACAO DO MODO ASCENDENTE-DESCENDENTE.**

Faça

$$M_{modo} = [1, 1, max],$$

$$MP_{modo} = [2, max]$$

$$t = 0$$

$$Y = (1,0,0,0,0,2)$$

(iv) INICIALIZACAO DO MODO ASCENDENTE CONSERVADOR.

Faça

$$M_{modo} = [1,1,2],$$

$$MP_{modo} = [3,max]$$

$$t=\emptyset$$

$$\Psi = (1,0,0,0,0,2)$$

6.8.2.1 A FUNCAO $\Gamma\Gamma$.

A função $\Gamma\Gamma$ é definida pelo algoritmo controlador de modo de operação . Este algoritmo estabelece o controle do sequenciamento da utilização do conhecimento representado em diferentes graus hierárquicos. Isto é, ele estabelece como são utilizados os diferentes modos de operação do sistema.

Inicia-se a análise considerando o algoritmo auxiliar fim de ciclo. Este algoritmo é utilizado pelo algoritmo controlador de modo de operação.

(i) ALGORITMO AUXILIAR FIM DE CICLO.

Em um sistema com arquitetura EEO, o λ-programa λ-P é uma lista de fórmulas que representa todo o conhecimento do sistema. Considera-se nesta lista, duas categorias de fórmulas: as fórmulas que representam um conhecimento a ser obtido pelo sistema e aquelas que representam um conhecimento existente. O conhecimento que é objetivo a ser atingido é representado no conjunto de fórmulas

SOB(t)

denominado situação objetivo. Observe que tal conjunto de fórmulas depende da variável "t", o que significa que os objetivos do sistema modificam-se de um ciclo para outro. Da mesma forma, o conhecimento já obtido pelo sistema é representado no conjunto de fórmulas

SAT(t)

denominado por situação atual. Observe que este conjunto também depende da variável "t".

O algoritmo auxiliar fim de ciclo estabelece se a situação atual,

$$SAT(t+1),$$

obtida no final do ciclo t e início do ciclo t+1, é "igual" a situação objetivo

$$SOB(t+1).$$

Caso esta "igualdade" esteja dentro de padrões especificados, então o sistema tem uma solução para o problema que é o objeto de trabalho do sistema. Caso contrário, o algoritmo incrementa o ciclo, fazendo

$$t=t+1,$$

e volta ao algoritmo principal que o solicitou. Deve-se enfatizar que os critérios que estabelecem a igualdade das situações indicadas acima dependem do problema a ser resolvido e da precisão desejada. O algoritmo é indicado a seguir.

****

if

$SOB(t+1) \cong SAT(t+1)$

then **END.**

else $t=t+1$, **RETURN.**

****

(ii) ALGORITMO CONTROLADOR DE MODO DE OPERACAO.

O algoritmo controlador de modo de operação define a função ΓΓ, que por sua vez determina a função **MODO.** Observe que a função **MODO** estabelece os graus das aplicações a serem executadas. O resultado deste algoritmo (o resultado da função ΓΓ) é um valor para a variável $\aleph_{grau}$, ou o fim da operação do sistema. O algoritmo é indicado a seguir.

****
****

{ Este algoritmo considera que o grau maximo dos conhecimentos representados no sistema e "max" e que os unicos modos principais de operacao do sistema sao os modos principais ascendente, descendente, ascendente-descendente e ascendente-conservador }

(i) if $\mathcal{V} = (0,0,1,1,\mathcal{K}_{grau})$, $\mathcal{K}_{grau} \in \mathbb{N}$,

then GOTO (ii),
else GOTO (i).

(ii) if $\mathcal{M}_{modo} = [1,1,\mathbb{Z}]$,

then GOTO (iii),
else GOTO (viii)

(iii) if $\mathcal{K}_{grau} + 1 > max$,

then EXECUTE O ALGORITMO FIM DE CICLO,
GOTO (iv),
else $\mathcal{K}_{grau} = \mathcal{K}_{grau} + 1$ END.

{ o valor de $\mathcal{K}_{grau}$ e incrementado e o algoritmo termina }

(iv) if $\mathcal{MP}_{modo} = [1,max]$,

then $\mathcal{K}_{grau} = 1$, END.

else GOTO (v).

(v) if $\mathcal{MP}_{modo} = [2,max]$,

then $\mathcal{M}_{modo} = [0,max,1]$, $\mathcal{K}_{grau} = max-2$ END.

else GOTO (vi).

{ Com $\mathcal{M}_{modo}=[0,max,1]$ o modo fundamental se torna o descendente. Em seguida faz-se $\mathcal{K}_{grau}=max-2$, isto e, os conhecimentos de grau max-2 passam a ser utilizados e o algoritmo termina. }

(vi) if MP_{modo} = [3,max],

then $Z = \aleph_{grau}$, M_{modo} = [0,Z,1], END.

else GOTO (vii).

{ Com M_{modo}=[0,$\aleph_{grau}$,0] o modo fundamental se torna o descendente, a partir dos conhecimentos de grau $\aleph_{grau}$. Em seguida, o algoritmo retorna o valor anterior de $\aleph_{grau}$ }

(vii) print(o valor da variável M_{modo} ou MP_{modo} não está associada a um valor correto), ERRO.

(viii) if M_{modo} = [0,Z,1],

then GOTO (ix),
else GOTO (vii)

(ix) if $\aleph_{grau}$ = 2,

then EXECUTE O ALGORITMO FIM DE CICLO,
GOTO (x),
else $\aleph_{grau} = \aleph_{grau} - 1$ END.

{ o valor de $\aleph_{grau}$ e decrementado e o algoritmo termina }

(x) if MP_{modo} = [0,max],

then $\aleph_{grau}$ = max,

else GOTO (xi).

(xi) if MP_{modo} = [2,max],

then M_{modo} = [1,0,max], $\aleph_{grau}$ = 2 END.

else GOTO (xii).

{ Com M_{modo}=[1,0,max] o modo fundamental se torna o ascendente. Em seguida faz-se $\aleph_{grau}$=2, isto e, os conhecimentos de grau 2 passam a ser utilizados e o algoritmo termina. }

(xii) if $\mathbb{MP}_{modo}$ = [3,**max**],

then GOTO (xiii),

else GOTO (vii).

(xiii) if $\mathbb{Z}$ + 1 > **max**,

then $\mathbb{Z}$ = 3, $\mathbb{M}_{modo}$ = [1,1,2], GOTO (xiv),

else $\mathbb{Z}$ = $\mathbb{Z}$ + 1, $\mathbb{M}_{modo}$ = [1,1,$\mathbb{Z}$], GOTO (xiv).

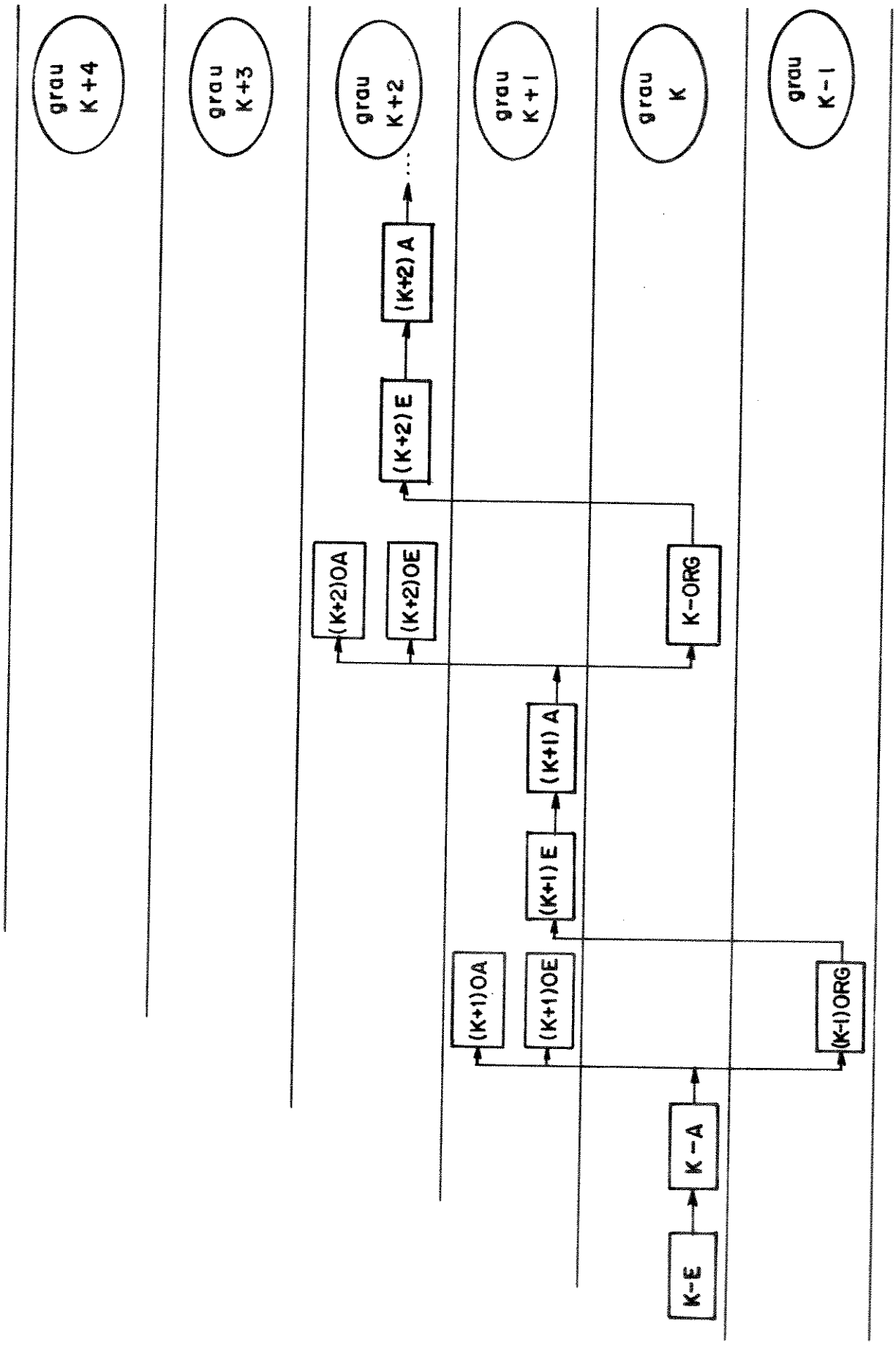
(xiv) $\mathbb{X}_{grau}$ = 2, END.

****
****

O algoritmo indicado acima controla o sequenciamento das execuções das funções de λ-P. O algoritmo determina este controle a partir do valor das variáveis

$$\mathbb{M}_{modo} \text{ e } \mathbb{MP}_{modo}$$

Faz-se testes para identificar os valores de tais variáveis, determinando em seguida o valor do grau ativo do sistema. Um detalhamento do funcionamento deste algoritmo é encontrado em [Souza, 1989]. As figuras 6.6 e 6.7, indicadas a seguir mostram o sequenciamento das execuções dos passos elementares de diferentes graus de complexidade quando os modos de operação são respectivamente o ascendente e o descendente. Este sequenciamento é controlado pelo algoritmo controlador de modo de operação.



Sequenciamento no modo fundamental ascendente.

FIGURA 6.6

Na figura 6.6, a primeira função executada é

k-E(t),

depois a função

k-A(t)

e em seguida são executadas, paralelamente, as funções

(k-1)-ORG,

(k+2)OE

e

(k+2)OA.

Após tais execuções tem-se a sequência de execuções,

(k+1)-E(t),

(k+1)-A(t),

nesta ordem, e paralelamente

k-ORG,

(k+3)-OE

e

(k+3)-OA,

e assim por diante, sempre incrementando o grau hierárquico das funções consideradas. Sendo este incremento o que caracteriza o modo fundamental de operação ascendente. Observe que o resulado da execução da função k-A(t) não é indicado.

Na figura 6.7, a primeira função executada é

k-E(t),

depois a função

k-A(t)

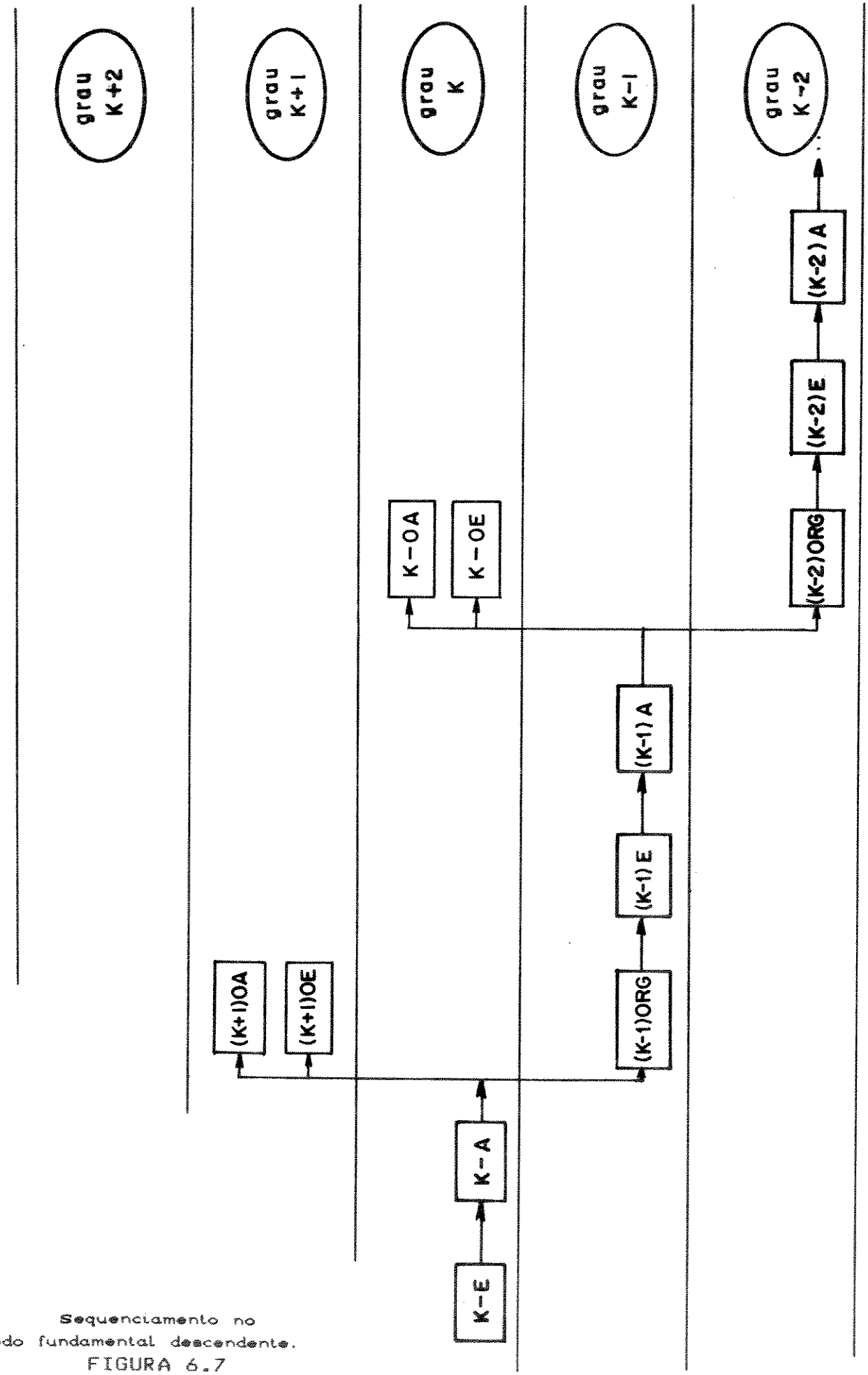
e em seguida são executadas, paralelamente, as funções

(k-1)-ORG,

(k+2)OE

e

(k+2)OA.



Sequenciamento no modo fundamental descendente.

FIGURA 6.7

Após tais execuções tem-se a sequência de execuções,

(k-1)-E(t),

(k-1)-A(t),

nesta ordem, e paralelamente

(k-2)-ORG,

(k+1)-OE

e

(k+1)-OA,

e assim por diante, sempre decrementando o grau hierárquico das funções consideradas. É este decremento que caracteriza o modo fundamental de operação descendente. Observe que ocorre o inverso do modo fundamental ascendente.

6.9 CONCLUSAO.

Neste capítulo foi ilustrada a representação hierárquica de conhecimento a partir de programas com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$. Obteve-se uma representação onde foi considerada a hierarquização das fórmulas do λ-cálculo tipado. Esta hierarquização foi definida ordenando-se as fórmulas do λ-cálculo tipado a partir das relações de complexidade, introduzidas no capítulo 4, resultando na hierarquia completa. Esta hierarquia traduz a semântica da classificação hierárquica do conhecimento representado pelas fórmulas. A sua utilização foi realizada considerando-se uma representação de conhecimento em um sistema com uma arquitetura, denominada EEO. Esta arquitetura baseia-se nos passos elementares ESCOLHA, EXECUCAO e ORGANIZACAO, definidos por fórmulas do λ-cálculo tipado. Foi proposto um critério de derivação $\mathbb{D}$, que determinou a sequência de execução das funções de $\lambda\text{-}P$. A utilização de um programa com critério permitiu explicitar conhecimentos em diferentes complexidades.

CAPITULO 7

CONCLUSAO.

Foi proposta uma abordagem para a representação de conhecimento utilizando um sistema lógico de ordem superior, fundamentado no λ-cálculo tipado, que considera explicitamente a presença de símbolos para tipo associados aos elementos da linguagem. Esta abordagem, que corresponde a um novo enfoque na utilizaχ o de elementos de lógica de ordem superior, ordena as fórmulas do λ-cálculo tipado, e permite classificar hierarquicamente o conhecimento de um dado sistema.

Para estabelecer uma classificação hierárquica foram definidas relações de "complexidade" entre as fórmulas do λ-cálculo tipado. Foi proposto um método para a classificação das relações de "complexidade", que se fundamenta na ordem dos símbolos para tipo associados às variáveis, às constantes e às fórmulas utilizadas na representação do conhecimento.

Em adição à classificação hierárquica também foi proposto que o conhecimento de um sistema seja representado por um programa com critério. Isto é, ele é representado por um conjunto de fórmulas associado a um critério de derivação. Para tanto foi apresentado o conceito formal de programa com critério e em seguida, a derivação de conhecimento a partir destes programas. Definiu-se um programa com critério como uma lista de fórmulas do λ-cálculo tipado associada a um mecanismo de derivação de conhecimento.

Utilizando as relações de complexidade e a definição de programa com critério foram determinadas condições necessárias para que uma dada fórmula seja uma dedução por derivação a partir de um programa com critério.

A seguir, introduziu-se uma representação hierárquica de conhecimento a partir de programas com critério P. A hierarquia da representação do conhecimento fundamentou-se em uma hierarquia das fórmulas de P, que foi definida utilizando-se as relações de complexidade entre as fórmulas do λ-cálculo tipado. A partir da hierarquia das fórmulas de P foi definido o sequenciamento da execução a funções representadas pelas fórmulas do programa com critério P. Foram considerados vários sequenciamentos: do conhecimento menos complexo para o mais complexo (modo ascendente), do conhecimento mais complexo para o menos complexo (modo descendente), etc.

Finalmente, definiu-se uma arquitetura com objetivo de ilustrar a representação hierárquica de conhecimento a partir de fórmulas do λ-cálculo tipado. Esta arquitetura fundamentou-se em três passos elementares, denominados ESCOLHA, EXECUÇÃO e ORGANIZAÇÃO. Estes passos elementares foram definidos utilizando-se elementos do λ-cálculo tipado e as relaχões de complexidade.

Resumindo, a utilização de programas com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$ e as relaχões de complexidade permitem:

(i) uma representação hierárquica a partir da ordenação das fórmulas de λ-P. . Esta ordenaχão , obtida por relaχões de complexidade, classifica as fórmulas do λ-cálculo tipado,

ii) novo enfoque no estudo das fórmulas do λ-cálculo tipado e do conhecimento por elas representado,

iii) estabelecer relações de ordem no conjunto das fórmulas do λ-cálculo tipado,

iv) obter condições necessárias para que uma fórmula Φ seja uma dedução por derivação a partir de um programa com critério. Quando a fórmula Φ possui complexidades quanto ao grau, quanto ao nível e quanto à descrição total e parcial menores que as respectivas complexidades das fórmulas pertencentes a λ-P, então esta fórmula pode ser uma dedução por derivação a partir do programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

Decorrente das propostas introduzidas neste trabalho sugere-se os seguintes temas para futuras pesquisas.

(a) Implementação de uma programação baseada em programas com critério.

(b) Definição de novas relações de complexidade, que representem outras ordenações das fórmulas do λ-cálculo tipado.

(c) Obtenção de outras condições necessárias à dedução de conhecimento a partir de um programa com critério.

(d) Aplicação da representação hierárquica de conhecimento em outras áreas de conhecimento, utilizando-se os programas com critério como descrito no capítulo 6.

(e) Aplicação dos conceitos de programação com critério em processamento paralelo da informação.

(f) Pesquisas de novas hierarquizações das fórmulas do λ-cálculo tipado e suas utilizações na representação hierárquica de conhecimento. Observe que no capítulo 6 é proposta uma hierarquização e é ilustrada a sua utilização.

(g) A extensão dos conceitos propostos neste trabalho considerando o sistema F de Girard, onde os símbolos para tipo ocorrem também na forma de variáveis.

APENDICE A

REPRESENTACAO DE REDES DE PETRI.

Apresenta-se neste apendice uma ilustracao dos conceitos propostos no capitulo 3. Os conceitos de derivacao de conhecimentos a partir de um programa com criterio sao ilustrados considerando-se um programa com criterio que simula uma rede de Petri.

A.1 INTRODUCAO.

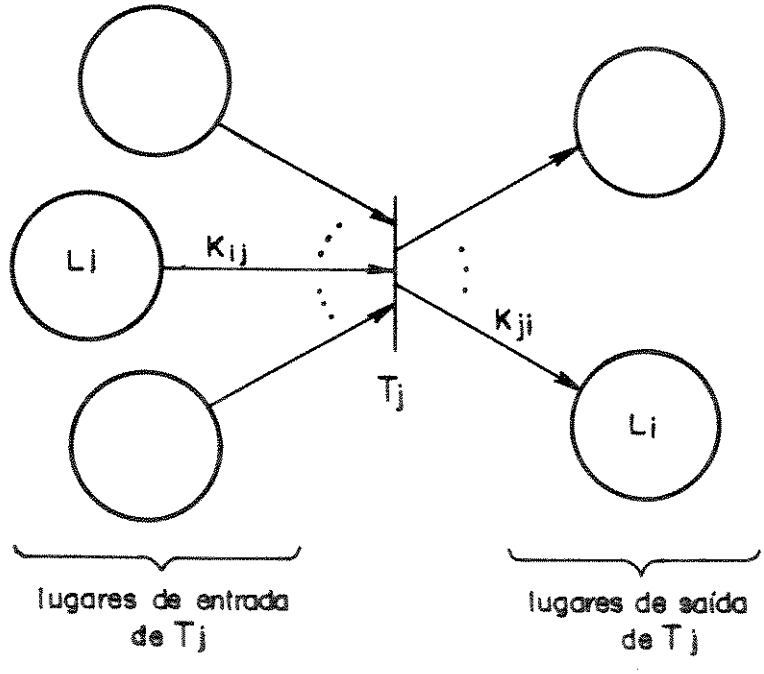
Apresenta-se neste apêndice uma ilustração dos conceitos definidos no capítulo 3. Propõe-se um programa com critério, que representa a simulação de redes de Petri [Peterson, 1977], [Peterson, 1981], ilustrando o conceito de derivaçao de conhecimentos a partir de um programa com critério.

Para definir um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$ define-se, inicialmente, a lista de fórmulas do λ-programa $\lambda\text{-}P$ e em seguida o critério $\mathbb{D}$. As fórmulas de $\lambda\text{-}P$ devem representar cada transição e cada lugar da rede. Estas fórmulas associadas a um critério de derivação determinam o sequênciamento da utilização dos conhecimentos representados. Finalmente, são consideradas as sequências de derivação com e sem memória, obtidas a partir do programa com critério. Estas sequências representam os conhecimentos obtidos quando as transições da rede de Petri são disparadas.

Inicialmente, na seção A.2, são apresentados os conceitos básicos sobre redes de Petri. Na seção A.3 propõe-se a representação básica do sistema. é proposto um conjunto de conceitos básicos que é utilizado na representação de redes de Petri por programas com critério. Em seguida, na seção A.4, são definidas as fórmulas básicas, que representam as trasições e os lugares da rede de Petri. Na seção A.5 são propostas as fórmulas de controle, que determinam o sequenciamento da execução das fórmulas básicas. O sequenciamento das fórmulas do λ-programa $\lambda\text{-}P$ é proposto na seção A.6. Finalmente, nas seções A.7, A.8 e A.10 são definidos formalmente o λ-programa $\lambda\text{-}P$, o critério de derivação $\mathbb{D}$ e as sequências de derivação.

A.2 REDES DE PETRI.

As redes de Petri constituem uma linguagem para simulação de eventos, principalmente eventos sequênciais [Peterson, 1981]. Elas são contituídas por dois elementos básicos: **(i)** as transições e **(ii)** os lugares, que são interligados, determinando a configuração da rede. A figura 3.3 mostra uma rede contituída pela transição T_j e os lugares $L_1,\ldots,L_s,W_1,\ldots,W_r$. Os lugares $L_1,\ldots,L_s$ são os lugares de entrada da transição T_j e $W_1,\ldots,W_r$ os lugares de saída. A transição T_j é a transição de saída dos lugares $L_1,\ldots,L_s$ é a transição de entrada dos lugares $W_1,\ldots,W_r$. Para disparar T_j é necessário a presença de marcas nos seus lugares de entrada. Após o seu disparo são retiradas marcas dos lugares de entrada e colocado nos lugares de saída. O número de marcas retiradas, ou colocadas em um lugar, depende do número de arestas que liga este lugar a transição disparada. Assim, se há duas arestas, ligando T_j ao lugar de entrada L_i, então após o disparo de T_j são retiradas duas marcas em L_i. Observa-se que T_j só é disparada quando há marcas suficientes nos seus lugares de entrada. Analisa-se a seguir uma representação de redes de Petri por um programa com critério. Nesta análise, considera-se a notação e os conceitos fundamentais para redes de Petri encontrados em [Peterson, 1981].



Uma rede de Petri.

FIGURA A.1

A.3 REPRESENTACAO BASICA.

Propõe-se a seguir um conjunto de conceitos básicos que são utilizados na representação de redes Petri por programas com critério. Inicia-se a análise pela caracterização de dois tipos de fórmulas.

A.3.1 DEFINICAO. Formas CONST e LIVRE.

Seja a fórmula

$$\Omega = \lambda X_1 \ldots \lambda X_n \; \Psi$$

onde $n \neq \emptyset$,

$$\Psi = (\xi \; F_1 \ldots F_m)$$

e ξ é uma constante, ou variável. A forma "CONST" de Ω, indicada por ΩC, é definida por,

$$\Omega C = \lambda X_1 \ldots \lambda X_n \; T_\alpha^\Psi$$

ΩC é obtida a partir de Ω trocando a fórmula

$$(\xi \; F_1 \ldots F_m)$$

pela constante T_α^Ψ onde

$$\alpha = \text{tipo}[\; \Psi \;]$$

Inversamente, diz-se que Ω é a forma "LIVRE" de ΩC.

***

Tem-se que

$$\text{tipo}[\Omega] = \text{tipo}[\Omega C]$$

pois

$$\text{tipo}[T_\alpha^\Psi] = \text{tipo}[\Psi]$$

e ΩC é uma função constante igual a T_α^Ψ.

O resultado da execução de uma função na forma "CONST" não adiciona conhecimentos a um sistema pois tal resultado é sempre igual à mesma constante. O mesmo não ocorre quando a função está em sua forma "LIVRE". Assim, transformar um função Ω da forma "LIVRE" para a forma "CONST", ΩC equivale a "desativar" a função. A partir de tal tranformação a execução de ΩC não adiciona conhecimentos ao sistema.

Como as fórmulas de um λ-programa são fórmulas do λ-cálculo tipado, as constantes e variáveis pertencentes a tais fórmulas estão associadas a símbolos para tipo. Na representação que se segue estes símbolos para tipo são utilizados explícitamente na representação do conhecimento envolvido. Dada uma constante A_α, o símbolo para tipo α ! usado indica qual é o conhecimento representado por A_α. Inicia-se pela representação do número de marcas presentes nos lugares da rede. Sejam

$$L_1, \ldots, L_n$$

os lugares da rede de Petri. Em cada lugar L_i existe um número inteiro de marcas, maior ou igual a zero. Na representação proposta, o número de marcas é representado por um inteiro associado ao símbolo para tipo

$$int_i \in \mathbb{F}$$

onde "i" representa o índice do lugar L_i. Se há 4 marcas no lugar L_i, este fato é indicado pela constante

$$4_{int}$$

A variável

$$X_{int_i}$$

representa o número de marcas no lugar L_i. Logo, se

$$X_{int_i} = 4_{int_i}$$

então o número de marcas no lugar L_i é igual a 4.

Considere agora

$$T_1, \ldots, T_m$$

as transições da rede. De uma maneira analoga, associa-se a cada transição T_i uma variável

$$Y_{inte_i}$$

associada ao símbolo para tipo

$$inte_i \in \mathbb{F}$$

onde "i" representa o índice da transição T_i. A variável Y_{inte_i} pode assumir apenas os valores 1_{inte_i} e 0_{inte_i}.

Convenciona-se que no primeiro caso, quando

$$Y_{inte_i} = 1_{inte_i},$$

a trancição T_i já foi disparada. Já no segundo caso, quando

$$Y_{inte_i} = \emptyset_{inte_i},$$

tem-se que T_i ainda não foi disparada. Além disso, associa-se às transições da rede a palavra de controle

$$PC_{trans}$$

onde $trans \in \mathbb{F}$. PC_{trans} pode assumir qualquer valor inteiro no intervalo $[1,m]$. Convenciona-se que uma transição T_j so é disparada quando se tem

$$PC_{trans} = j$$

e os números de marcas nos seus lugares de entrada são suficientes. Isto é, após o disparo de T_j os números de marcas nos seus lugares de entrada permanecem maiores que zero.

A.4 FORMULAS BASICAS.

As fórmulas básicas do λ-programa λ-P são definidas a seguir. Inicia-se pelas fórmulas que representam as transições da rede. Em seguida são consideradas as fórmulas que representam o incremento e o decremento do número de marcas de um lugar L_i.

Para representar cada transição T_j propõe-se a seguinte fórmula

$$TR^j = \lambda PC_{trans}\ \lambda X_{int_{s1}} \ldots \lambda X_{int_{sn}} \left(f^j\ PC_{trans}\ X_{int_{s1}} \ldots X_{int_{sn}} \right)$$

onde

$$f^j\ PC_{trans}\ X_{int_{s1}} \ldots X_{int_{sn}} = \begin{cases} \emptyset_{inte_j} & \text{se } PC_{trans} \neq j \\ R^j\ X_{int_{s1}} \ldots X_{int_{sn}} & \text{se } PC_{trans} = j \end{cases}$$

e

$$R^j\ X_{int_{s1}} \ldots X_{int_{sn}} = D\left(\left(\sum_{i=s1}^{sn} D(X_{int_i} \div K_{ij})\right) \div sn\right)$$

onde

$$K_{ij} \neq 0,\ i = s1, s2, \ldots, sn$$

Esta constante é igual ao número de entradas da transição T_j referentes ao lugar L_i. Isto é, K_{ij} é igual ao número de arestas ligando o lugar L_i à transição T_j. Tem-se também que,

$$tipo[K_{ij}] = int$$

para todo "ij". As funções somatório "Σ" e divisão "$\div$" possuem a interpretação usual e consideram fórmulas associadas ao símbolo para tipo "int_i", "ij", como sendo números inteiros para quaisquer índices "i","j". A função D é tal que,

$$D(X) = \begin{cases} 1 & \text{se } x \geq 1 \\ 0 & \text{se } x < 1 \end{cases}$$

As variáveis

$$X_{int_{s1}}, X_{int_{s2}}, \ldots, X_{int_{sn}}$$

são as variáveis que indicam os números de marcas nos lugares de entrada da transição T_j. O símbolo para tipo associado a TR^j é

$$tipo[TR^j] = (trans \succ int_{s1} \succ \ldots \succ int_{sn} \succ inte_j)$$

Logo, o resultado da aplicação TR^j aos seus argumentos está associada a um símbolo para tipo $inte_j$. Considerando

$$A_{int_{s1}}, \ldots, A_{int_{sn}}$$

as constantes que representam o número de marcas nos lugares

$$L_{s1}, \ldots, L_{sn}$$

respectivamente, e

$$PC_{trans} = 1,$$

então,

$$TR^j\ PC_{trans}\ A_{int_{s1}} \ldots A_{int_{sn}} = 1_{inte_j}$$

se e somente se

$$A_{int_{si}} \geq K_{ij}$$

onde $1 \leq i \leq n$. Quando o resultado da aplicação de TR^j a seus argumentos é igual a 1_{inte_j} significa que T_j é disparada. No

outro caso, se o resultado é igual a $\emptyset_{inte_j}$, então a transição não é disparada. Neste caso, existe "i", $1 \leq i \leq n$, tal que

$$A_{int_{ei}} < K_{ij}$$

ou

$$PC_{trans} \neq j$$

Quando ocorre o disparo de uma transição T_j, são retiradas marcas dos seus lugares de entrada e são colocados em seus lugares de saída. Define-se a seguir as funções que representam o incremento e o decremento destas marcas. Inicia-se pela função $\mathbf{DEC}^{ij}$, denominada de função "decremento". Ela retira marcas do lugar de entrada L_i quando a transição T_j é disparada. Considere "i", $1 \leq i \leq n$, tal que L_i é um lugar de entrada da transição T_j. Então,

$$\mathbf{DEC}^{ij} = \lambda X_{int_i} \lambda Y_{inte_j} \left[g^{ij} \; X_{int_i} \; Y_{inte_j} \right]$$

onde

$$g^{ij} \; X \; Y = \begin{cases} X - K_{ij} & \text{se } Y=1 \\ X & \text{se } Y=\emptyset \end{cases}$$

K_{ij} é o número de entradas, ou arestas que conectam o lugar L_i à transição T_j. A operação "−" é a subtração usual, que considera fórmulas associadas aos símbolos para tipo int_i como inteiros. Tem-se também que,

$$\mathbf{tipo}[\mathbf{DEC}^{ij}] = (int_i \succ inte_j \succ int_i)$$

logo, o resultado da aplicação de $\mathbf{DEC}^{ij}$ aos argumentos X_{int_i} e Y_{inte_j} é uma fórmula associada ao símbolo para tipo int_i. Considere

$$A_{int_i}$$

o número de marcas no lugar L_i e que

$$Y_{inte_j} = 1_{inte_j}$$

Logo, se

$$B_{int_i} = \mathbf{DEC}^{ij} \; A_{int_i} \; Y_{inte_j}$$

então

$$B_{int_i} = A_{int_i} - K_{ij}$$

B_{int_i} é igual ao número de marcas no lugar L_i, após o disparo da transição T_j. Observe que quando a transição T_j é disparada, tem-se $Y_{inte_j} = 1$. Em decorrência disso, após a execução de $\mathbf{DEC}^{ij}$ há um decremento igual a K_{ij} no valor de X_{int_i} se T_j não é disparada, o valor de X_{int_i} fica inalterado.

Observe ainda que na definição acima considera-se que L_i é um lugar de entrada da transição T_j. Caso isto não ocorra, então a definição de $\mathbf{DEC}^{ij}$ é a seguinte

$$\mathbf{DEC}^{ij} = \lambda X_{int_i} \lambda Y_{inte_j} \; X_{int_i}$$

isto é, $\mathbf{DEC}^{ij}$ é a identidade na primeira coordenada. Neste caso o número de marcas no lugar L_i permanece inalterado.

Define-se a seguir a função $\mathbf{INC}^{jk}$, denominada função "incremento". Quando a transição T_j é disparada, ela incrementa o número de marcas no lugar de saída L_k. Considere k, $1 \leq k \leq n$, tal que L_k é um lugar de saída da transição T_j. Então,

$$\mathbf{INC}^{jk} = \lambda X_{int_k} \lambda Y_{inte_j} \left(h^{jk} \; X_{int_k} \; Y_{inte_j} \right)$$

onde,

$$h^{jk} \; X \; Y = \begin{cases} X + K_{jk} & \textbf{se } Y=1 \\ X & \textbf{se } Y=0 \end{cases}$$

K_{jk} é o número de saídas da transição T_j em relação ao lugar L_k. A operação "+" é a adição usual, que considera fórmulas associadas ao símbolo para tipo int_k com inteiros. Tem-se também que

$$\mathbf{tipo}[\mathbf{INC}^{jk}] = (int_k \succ inte_j \succ int_k) .$$

Considere

$$A_{int_k}$$

o número de marcas no lugar L_k. Se a transição T_j foi disparada, isto é,

$$Y_{inte_j} = 1$$

e

$$B_{int_k} = \mathbf{INC}^{jk} \; A_{int_k} \; Y_{inte_j}$$

então

$$B_{int_k} = A_{int_k} + K_{jk}$$

B_{int_k} é o número de marcas no lugar L_k após o disparo da transição T_j. Observe que após a execução de $\mathbf{INC}^{jk}$, quando se tem $Y_{int\bullet_j} = 1$, há um incremento igual a K_{jk} no valor de A_{int_k}. Isto significa que são adicionadas K_{jk} marcas ao lugar L_k. Por outro lado, quando T_j não é disparada, o número de marcas em L_k fica inalterado.

Na definição acima, da função incremento, considera-se que L_k é um lugar de saída da transição T_j. Quando L_k não é um lugar de saída da transição T_j, propõe-se então a seguinte definição de $\mathbf{INC}^{jk}$

$$\mathbf{INC}^{jk} = \lambda X_{int_k}\, \lambda Y_{int\bullet_j}\, X_{int_k}$$

isto é, $\mathbf{INC}^{jk}$ é a identidade na primeira coordenada. Neste caso o número de marcas no lugar L_k permanece inalterado.

A.5 FORMULAS DE CONTROLE.

As funções

$$\mathbf{TR}^j,\ \mathbf{DEC}^{ij},\ \mathbf{INC}^{jk}$$

representam as transições da rede, o decremento e o incremento de marcas nos lugares de entrada e de saída respectivamente. A seguir são definidas as funções

$$\Pi\mathbf{TR}^{ijk},\ \Pi\mathbf{DEC}^{ijk} \text{ e } \Pi\mathbf{INC}^{ijk},$$

que controlam o sequenciamento da execução das funções

$$\mathbf{TR}^j,\ \mathbf{DEC}^{ij} \text{ e } \mathbf{INC}^{jk}.$$

Tais funções também pertencem ao λ-programa λ-P. Deve-se observar que o controle deste sequênciamento é necessário, pois as funções incremento e decremento só podem ser executadas uma única vez após o disparo de uma transição t_j, ou seja após a execução de $\mathbf{TR}^j$. Se a função decremento $\mathbf{DEC}^{ij}$ é executada indefinidamente, o número de marcas do lugar L_i se torna negativo, fato que não deve ocorrer. As funções

$$\Pi\mathbf{TR}^{ijk},\ \Pi\mathbf{DEC}^{ijk} \text{ e } \Pi\mathbf{INC}^{ijk}$$

transformam as funções

$$\mathbf{TR}^{j}, \mathbf{DEC}^{ij} \quad e \quad \mathbf{INC}^{jk},$$

inibindo ou permitindo a sua aplicação. Finalmente, observa-se que o controle completo somente é obtido quando se considera o critério de derivação do programa. Antes de definir as funções ou fórmulas de controle considere

$$\alpha^{j} = \mathbf{tipo}[\mathbf{TR}^{j}]$$

$$\beta^{ij} = \mathbf{tipo}[\mathbf{DEC}^{ij}]$$

$$\gamma^{jk} = \mathbf{tipo}[\mathbf{INC}^{jk}]$$

e

$$\mathbf{TR}\mathbb{C}^{j}, \quad \mathbf{DEC}\mathbb{C}^{ij}, \quad \mathbf{INC}\mathbb{C}^{jk}$$

as formas "CONST" associadas às fórmulas

$$\mathbf{TR}^{j}, \quad \mathbf{DEC}^{ij} \quad e \quad \mathbf{INC}^{jk}$$

respectivamente. As funções são definidas a seguir.

$$\Pi\mathbf{TR}^{ijk} = \lambda\mathbf{TR}^{j}\ \lambda\mathbf{DEC}^{ij}\ \lambda\mathbf{INC}^{jk}\ \left(\mathbf{INIB1}\ \mathbf{TR}^{j}\ \mathbf{DEC}^{ij}\ \mathbf{INC}^{jk} \right)$$

onde

$$\mathbf{tipo}[\Pi\mathbf{TR}^{ijk}] = (\alpha^{j} \succ \beta^{ij} \succ \gamma^{jk} \succ \alpha^{j})$$

e

$$\mathbf{INIB1}\ \mathbf{TR}\mathbb{C}^{j}\ \mathbf{DEC}\mathbb{C}^{ij}\ \mathbf{INC}\mathbb{C}^{jk} = \mathbf{TR}^{j}$$

$$\mathbf{INIB1}\ \mathbf{TR}^{j}\ \mathbf{DEC}\mathbb{C}^{ij}\ \mathbf{INC}\mathbb{C}^{jk} = \mathbf{TR}\mathbb{C}^{j}$$

e identidade na primeira coordenada nos outros casos. Isto é, se

$$(X,Y,Z) \neq (\mathbf{TR}\mathbb{C}^{j},\mathbf{DEC}\mathbb{C}^{ij},\mathbf{INC}\mathbb{C}^{ji})$$

$$(X,Y,Z) \neq (\mathbf{TR}^{j},\mathbf{DEC}\mathbb{C}^{ij},\mathbf{INC}\mathbb{C}^{jk})$$

então

$$\mathbf{INIB1}\ X\ Y\ Z = X$$

A função $\Pi\mathbf{TR}^{ijk}$ habilita e desabilita a função que representa a transição T_j. A sua execução modifica $\mathbf{TR}^{j}$ para a sua forma "CONST" ou "LIVRE". Assim, quando as funções

$$\mathbf{TR}^{j}, \ \mathbf{DEC}^{ij}, \ \mathbf{INC}^{jk}$$

estão em sua forma "CONST", $\Pi\mathbf{TR}^{ijk}$ modifica a fórmula $\mathbf{TR}^{j}$ para sua forma "LIVRE". Por outro lado, quando se tem

$$\mathbf{DEC}^{ij} \ e \ \mathbf{INC}^{jk}$$

na forma "CONST" e $\mathbf{TR}^{j}$ na forma "LIVRE", $\Pi\mathbf{TR}^{ijk}$ transforma $\mathbf{TR}^{j}$ para a forma "CONST". Finalmente, nos casos em que os

argumentos de Π_{TR}^{ijk} não satisfazem tais condições, Π_{TR}^{ijk} é a identidade na primeira coordenada. A razão de se definir Π_{TR}^{ijk} como descrito acima é identificada a partir da figura 3.4, que é analisada após a definição das funções

$$\Pi_{DEC}^{ijk},\ \Pi_{INC}^{ijk} \in Y^{j},$$

indicadas a seguir.

$$\Pi_{DEC}^{ijk} = \lambda_{TR}^{j}\ \lambda_{DEC}^{ij}\ \lambda_{INC}^{jk}\ \left(\ \mathbf{INIB2}\ TR^{j}\ DEC^{ij}\ INC^{jk}\ \right)$$

onde

$$tipo[\Pi_{DEC}^{ijk}] = (\alpha^{j} \succ \beta^{ij} \succ \gamma^{jk} \succ \beta^{ij})$$

$$\mathbf{INIB2}\ TR^{j}\ DEC\mathbb{C}^{ij}\ INC\mathbb{C}^{jk} = DEC^{ij}$$

$$\mathbf{INIB2}\ TR\mathbb{C}^{j}\ DEC^{ij}\ INC\mathbb{C}^{jk} = DEC\mathbb{C}^{ij}$$

e nos outros casos, **INIB2** é a identidade na primeira coordenada, isto é,

$$\mathbf{INIB2}\ X\ Y\ Z = X$$

Esta função habilita e desabilita a função DEC^{ij}, que decrementa o número de marcas dos lugares de entrada da transição T_j. Sua execução modifica a função DEC^{ij} para a sua forma "CONST" ou "LIVRE". Quando as funções

$$DEC^{ij} \in INC^{jk}$$

estão em sua forma "CONST" e TR^{j} na forma "LIVRE", Π_{DEC}^{ijk} modifica a fórmula DEC^{ij} para sua forma "CONST". O resultado da execução de Π_{DEC}^{ijk} é a transformação da função DEC^{ijk} para a função $DEC\mathbb{C}^{ij}$ ($DEC\mathbb{C}^{ij}$ é a função DEC^{ij} na sua forma "CONST"). Quando se tem

$$DEC^{ij} \in INC^{jk},$$

em sua forma "CONST" e TR^{j} em sua forma"LIVRE", DEC^{ij} volta a se habilitar. Finalmente, nos casos em que os argumentos de Π_{DEC}^{ijk} não satisfazem tais condições, Π_{DEC}^{ijk} é a identidade na primeira coordenada. A função Π_{INC}^{ijk} é definida a seguir,

$$\Pi_{INC}^{ijk} = \lambda_{TR}^{j}\ \lambda_{DEC}^{ij}\ \lambda_{INC}^{jk}\ \left(\ \mathbf{INIB3}\ TR^{j}\ DEC^{ij}\ INC^{jk}\ \right)$$

onde

$$tipo[\Pi_{INC}^{ji}] = (\alpha^{j} \succ \beta^{ij} \succ \gamma^{jk} \succ \gamma^{jk})$$

$$\mathbf{INIB3}\ TR\mathbb{C}^{j}\ DEC^{ij}\ INC\mathbb{C}^{jk} = INC^{jk}$$

$$\mathbf{INIB3}\ TR\mathbb{C}^{j}\ DEC\mathbb{C}^{ij}\ INC^{jk} = INC\mathbb{C}^{jk}$$

Nos outro casos, **INIB3** é a identidade na primeira coordenada. Π_{INC}^{ijk} habilita e desabilita a função INC^{jk}, que incrementa o número de marcas dos lugares de saída da transição T_j. A sua execução modifica a função INC^{jk} para a sua forma "CONST" ou

"LIVRE", conforme é determinado pela função **INIB3.**

Quando a transição T_j é disparada, o resultado da execução da função $\mathbf{TR}^j$ é a constante

$$1_{inte_j}$$

O valor desta constante é utilizado na execução das funções incremento $\mathbf{INC}^{jk}$ e decremento $\mathbf{DEC}^{ij}$. Para que tais funções não sejam executadas indefinidamente, a constante associada ao símbolo para tipo $inte_j$ deve ser zerada. Define-se a seguir a função Θ^j, que adiciona ao sistema a constante

$$0_{inte_j}$$

Isto é, Θ^j zera o valor da constante associada à transição T_j. Esta função é indicada a seguir.

$$\Theta^j = \lambda Y_{inte_j}\ \lambda \mathbf{TR}^j\ \lambda \mathbf{DEC}^{ij}\ \lambda \mathbf{INC}^{jk}\ \left[\ h^j\ Y_{inte_j}\ \mathbf{TR}^j\ \mathbf{DEC}^{ij}\ \mathbf{INC}^{jk}\ \right]$$

onde

$$tipo[\Theta^j] = (inte_j \succ \alpha^j \succ \beta^{ij} \succ \gamma^{jk} \succ inte_j)$$

e

$$h^j\ 1_{inte_j}\ \mathbf{TR}\mathbb{C}^j\ \mathbf{DEC}\mathbb{C}^{ij}\ \mathbf{INC}\mathbb{C}^{jk} = 0_{inte_j}$$

$$h^j\ Y\ X\ Z\ W = Y$$

$$\text{se } (X,Z,W) \neq (\mathbf{TR}\mathbb{C}^j, \mathbf{DEC}\mathbb{C}^{ij}, \mathbf{INC}\mathbb{C}^{jk})$$

Observe que no caso em que

$$Y_{inte_j} = 1$$

e as funções

$$\mathbf{TR}^j,\quad \mathbf{DEC}^{ij} \quad e \quad \mathbf{INC}^{jk}$$

estão na forma "CONST", então o resultado da aplicação de h^j aos seus argumentos é

$$0_{inte_j}$$

isto é, Y_{inte_j} é zerado. Nos outros casos, tem-se a identidade na primeira coordenada.

Na representação de uma rede de Petri, proposta neste trabalho, as transições são disparadas uma a uma. O controle destes disparos é determinado pelo valor de uma palavra de controle

$$PC_{trans}$$

Quando $PC_{trans} = j$, somente a transição T_j pode ser disparada. A seguir define-se uma função **INCPC**, que incrementa o valor da palavra de controle. Com isto é possível obter o disparo de todas as transições da rede.

$$\mathbf{INCPC} = \lambda PC_{trans}\ \lambda \mathbf{TR}^{j}\ \lambda \mathbf{DEC}^{ij}\ \lambda \mathbf{INC}^{jk}\ \left[\ P\ \ PC_{trans}\ \ \mathbf{TR}^{j}\ \ \mathbf{DEC}^{ij}\ \ \mathbf{INC}^{jk}\ \right]$$

onde

$$\mathbf{tipo[INCPC]} = (trans \succ \alpha^{j} \succ \beta^{ij} \succ \gamma^{jk} \succ trans)$$

e

$$P\ j\ \mathbf{TR}\mathbb{C}^{j}\ \mathbf{DEC}\mathbb{C}^{ij}\ \mathbf{INC}\mathbb{C}^{jk} = (j + 1)$$

se $j < m$, onde "m" é o número de transições da rede. Se $j > m$, então

$$P\ j\ \mathbf{TR}\mathbb{C}^{j}\ \mathbf{DEC}\mathbb{C}^{ij}\ \mathbf{INC}\mathbb{C}^{jk} = 1$$

Tem-se também,

$$P\ Y\ X\ Z\ W = Y$$

se

$$(X,Z,W) \neq (\mathbf{TR}\mathbb{C}^{j}, \mathbf{DEC}\mathbb{C}^{ij}, \mathbf{INC}\mathbb{C}^{jk})$$

Portanto, no caso em que se tem

$$PC_{trans} = j$$

e as funções

$$\mathbf{TR}^{j},\ \mathbf{DEC}^{ij}\ e\ \mathbf{INC}^{jk}$$

na forma "CONST", o resultado da aplicação da função "P" aos seus argumentos é igual a

$$j + 1$$

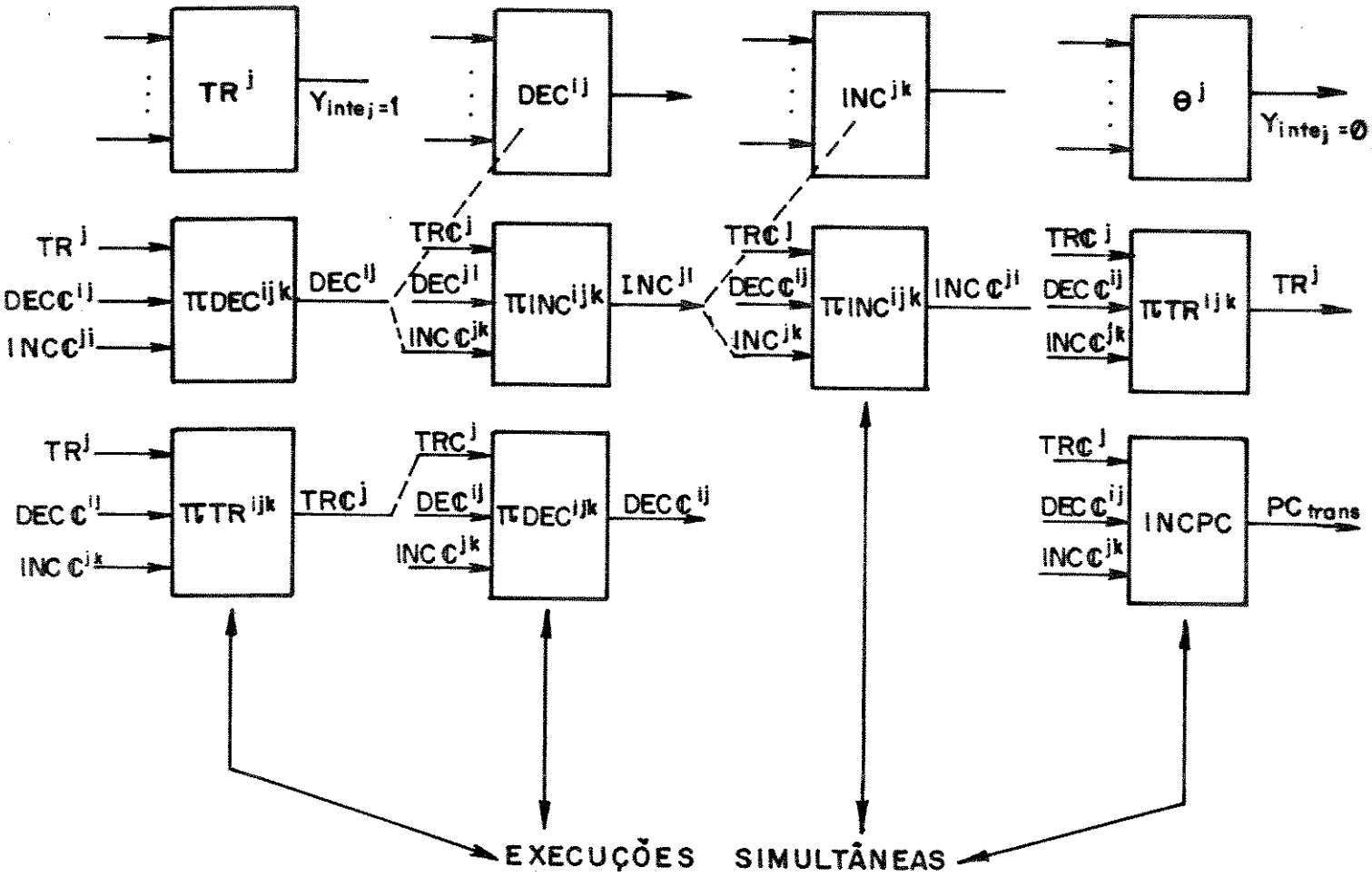
Isto é, a palavra de controle é incrementada. Tem-se que

$$PC_{trans} = j + 1$$

Nos outros casos, tem-se a identidade na primeira coordenada.

A.6 SEQUENCIAMENTO DA EXECUCAO DAS FORMULAS DO λ-PROGRAMA.

Nesta seção é analisado o sequênciamento da execução das funções que representam o disparo das transições, o incremento e o decremento do número de marcas dos lugares da rede de Petri. Este sequênciamento é representado na figura 3.4. Deve-se observar que o sequênciamento apresentado a seguir não considera todas as funções do λ-programa definidas até agora. É considerado apenas o controle do sequênciamento da execução das funções $\mathbf{TR}^{j}, \mathbf{DEC}^{ij}$ e $\mathbf{INC}^{jk}$ para um índice j fixo. No λ-programa tem-se $1 \leq j \leq n$. O critério a ser definido na próxima seção considerará o sequênciamento da execução de todas as funções do λ-programa λ-P.



Sequenciamento da execucao das formulae de λ-P.

FIGURA A.2

Na figura 3.4 são indicadas apenas as execuções das funções que modificam conhecimentos no sistema. O caso em que o resultado da execução de uma função é a identidade na primeira coordenada não é indicado. Nesta figura é representado o sequênciamento da execução de cada uma das funções definidas anteriormente, para i, j, k fixos. Observe que várias funções são executadas ao mesmo tempo, de um modo paralelo. As funções

$$TR^j, \quad \Pi DEC^{ijk} \quad e \quad \Pi TR^{ijk},$$

por exemplo, são executadas simultaneamente. É indicado também se as funções

$$TR^j, DEC^{ij} \quad e \quad INC^{jk}$$

estão na forma "CONST" ou "LIVRE", o que determina a possibilidade de execução das funções

$$\Pi_{TR}^{ijk} \quad \Pi_{DEC}^{ijk} \quad \Pi_{INC}^{ijk},$$

como também o seu resultado.

As funções

$$TR^{j}, DEC^{ij} \quad e \quad INC^{jk}$$

são executadas de uma maneira sequêncial, não ocorrendo a execução de mais de uma delas ao mesmo tempo, o que impossibilita, por exemplo, um segundo disparo de uma transição sem que as marcas tenham sido retiradas dos lugares de entrada e colocadas nos lugares de saída desta transição. Considere inicialmente que se tenha no sistema as seguintes fórmulas

$$TR^{j}, \quad DEC\mathbb{C}^{ij}, \quad INC\mathbb{C}^{jk}$$

o que possibilita a execução de TR^{j} pois ela esta na forma "LIVRE". Utilizando as fórmulas acima como argumentos das funções

$$\Pi_{TR}^{ijk} \quad e \quad \Pi_{DEC}^{ijk}$$

também são executadas. Isto significa que simultaneamente ao disparo da transição T_j (representado pela execução de TR^{j}), tem-se a sua desabilitação (execução de Π_{TR}^{ijk}) e a habilitação de DEC^{ijk} (execução de Π_{DEC}^{ijk}). Ocorre a execução da função TR^{j} e sua desabilitação ao mesmo tempo. Como resultado da execução de

$$TR^{j}, \quad \Pi_{DEC}^{ijk} \quad e \quad \Pi_{TR}^{ijk}$$

tem-se a presenca no sistema das fórmulas

$$TR\mathbb{C}^{j}, \quad DEC^{ij}, \quad INC\mathbb{C}^{jk}$$

e

$$Y_{inte_j} = 1$$

Em seguida tem-se a execução da função DEC^{ij} e o número das marcas do lugar de entrada de L_i é decrementado. Ocorre também a execução das funções

$$\Pi_{INC}^{ijk} \quad e \quad \Pi_{DEC}^{ijk},$$

que utilizam como argumentos as fórmulas

$$TR\mathbb{C}^{j}, \quad DEC^{ij}, \quad INC\mathbb{C}^{jk}.$$

Tais execuções habilitam a função INC^{jk} e desabilitam a função DEC^{ij}. Analogamente ao caso anterior, DEC^{ij} é executada e desabilitada ao mesmo tempo. Como resultado tem-se no sistema o seguinte conjunto de fórmulas.

$$TR\mathbb{C}^{j}, \quad DEC\mathbb{C}^{ij}, \quad INC^{jk}$$

De uma maneira análoga ao que ocorreu anteriormente, a função $\mathbf{INC}^{jk}$ é executada, incrementando o número de marcas do lugar de saída L_k. Tem-se também a execução de $\Pi_{\mathbf{INC}}{}^{ijk}$, que desabilita a função $\mathbf{INC}^{jk}$. Como resultado de tais execuções tem-se no sistema as seguintes funções.

$$\mathbf{TR}\mathbb{C}^{j},\ \mathbf{DEC}\mathbb{C}^{ij},\ \mathbf{INC}\mathbb{C}^{jk}$$

Finalmente, tem-se a execução das funções

$$\Theta^{j},\ \Pi_{\mathbf{TR}}{}^{ijk} \in \mathbf{INCPC}.$$

O resultado da execução de Θ^{j} é

$$Y_{inte_j} = \emptyset$$

Já o resultado da execução de **INCPC** depende do valor atual da palavra de controle. Se $j < m$, onde "m" é o número de transições da rede, então tem-se o incremento de PC_{trans}

$$PC_{trans} = PC_{trans} + 1$$

Se $j = m$, então

$$PC_{trans} = 1$$

A execução de $\Pi_{\mathbf{TR}}{}^{ijk}$ habilita o disparo da transição $\mathbf{TR}^{j}$, o que possibilita o inicio de um novo ciclo.

A.7 O λ-PROGRAMA.

Para definir um programa com critério $\langle \lambda\text{-}P,\ \mathbb{D} \rangle$ é necessário definir a lista de fórmulas λ-P e o critério $\mathbb{D}$. Nas seções anteriores é definido um conjunto de fórmulas que pertencem a λ-P. Estas fórmulas são as que representam as transições, as funções incremento e decremento e as funções controladoras.

A seguir, define-se formlmente o λ-programa λ-P. Ele é constituido por uma lista de fórmulas que representam os elementos da rede de Petri. Considere inicialmente as listas de constantes,

$$MARCAS = [A_{int_1}, \ldots, A_{int_n}]$$

onde A_{int_i} é o número de marcas no lugar L_i,

$$DISPARO = [\emptyset_{inte_1}, \ldots, \emptyset_{inte_m}]$$

que é uma lista de constantes indicando que não há nenhuma transição disparada,

$$\text{TRANSICOES} = [\mathbf{TR}^1, \mathbf{TR}^2, \ldots, \mathbf{TR}^m]$$

que é a lista das funções que representam as transições da rede. Observe que apenas $\mathbf{TR}^1$, que representa a transição T_1, é a única função que esta na forma "LIVRE". Isto é, apenas T_1 pode ser disparada. Considere também a lista,

$$\text{LUGARES} = [\mathbf{DEC}^{ij}, \mathbf{INC}^{jk}],$$

onde

$$1 \leq i,k \leq n, \quad 1 \leq j \leq m$$

Neste caso tem-se a lista das funções decremento e incremento de marcas. Todas as funções estão na forma "CONST", isto é, estão desativadas. A lista das funções de controle é indicada a seguir,

$$\text{CONTROLE} = [\Pi_{\mathbf{TR}}^{ijk}, \Pi_{\mathbf{DEC}}^{ijk}, \Pi_{\mathbf{INC}}^{ijk}],$$

onde

$$1 \leq i,k \leq n, \quad 1 \leq j \leq m,$$

Finalmente, considere que

$$PC_{trans} = 1_{trans}$$

o que possibilita o disparo da transição T_1, já que $\mathbf{TR}^1$ está na forma "LIVRE". λ-P é a lista de todas as constantes e funções que representam a rede de Petri.

$$\lambda\text{-}P = [\ PC_{trans} \mid \text{MARCAS} \mid \text{DISPARO} \mid$$

$$\text{TRANSICOES} \mid \text{LUGARES} \mid \text{CONTROLE}\ ]$$

A.8 CRITERIO DE DERIVACAO.

Para finalizar a representação da rede de Petri por um programa com critério, resta determinar o critério de derivação, a partir do qual as derivações são estabelecidas. Este critério determina o sequênciamento da execução de todas as fórmulas do λ-programa. O critério $\mathbb{D}$ é definido a seguir,

Dada uma fórmula $\Phi \in \lambda\text{-}P$, onde

$$\Phi = \lambda X_{\alpha_1} \ldots \lambda X_{\alpha_p} \ (A\ F_1 \ldots F_e)$$

então o critério de derivação $\mathbb{D}$ é dado por:

$$\mathbb{D}\ [\ \Phi,\ \lambda\text{-}P\ ] = [\Psi_{\alpha_1}, \ldots, \Psi_{\alpha_p}]$$

onde Ψ_{α_i} é a primeira fórmula associada ao símbolo para tipo α_i, encontrada em λ-P. Caso não exista uma fórmula em λ-P associada ao símbolo para tipo α_i, então

$$\Psi_{\alpha_i} = X_{\alpha_i}$$

Observe que λ-P é uma lista de fórmulas, sendo portanto um conjunto ordenado de fórmulas, o que possibilita a identificação da primeira fórmula associada ao símbolo para tipo α_i.

A.9 SEQUENCIAS DE DERIVACAO.

Analisa-se a seguir as sequências de derivação (com memória e sem memória) obtidas a partir do programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$.

(i) Sequencia de derivacao sem memoria.

A sequência de derivação sem memória é dada por

$$\langle \lambda\text{-}P_0, \mathbb{D} \rangle, \langle \lambda\text{-}P_1, \mathbb{D} \rangle \ \ldots, \langle \lambda\text{-}P_n, \mathbb{D} \rangle, \ldots$$

onde

$$\lambda\text{-}P_0 = \lambda\text{-}P$$

Considerando

$$\lambda\text{-}P_0 = [\Phi_1, \ldots, \Phi_m]$$

então,

$$\lambda\text{-}P_i = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_0]$$

onde Ψ_j é uma derivação em Φ_j conforme

$$\mathbb{D}\left[\Phi_j, \lambda\text{-}P_i\right]$$

e

$$\langle \lambda\text{-}P_i, \mathbb{D} \rangle$$

são programas com critério para todo i.

Nesta sequência, cada λ-programa λ-P_i é formado por uma lista de fórmulas que representam as transições, os lugares e a configuração atual das marcas da rede de Petri. Inicialmente,

$$\lambda\text{-}P_0 = [\ PC_{trans} \mid MARCAS \mid DISPARO \mid$$

$$TRANSICOES \mid LUGARES \mid CONTROLE\ ]$$

Dada uma fórmulas Φ_j onde

$$\Phi_j \in [\ PC_{trans} \mid MARCAS \mid DISPARO \mid LUGARES\ ]$$

então Φ_j é uma constante ou uma função na forma "CONST". Logo, não há derivação de novos conhecimentos em Φ_j e as fórmulas de $\lambda\text{-}P_1$ são obtidas por derivações em fórmulas pertencentes à lista

$$[TRANSICOES \mid CONTROLE\ ]$$

Por definição,

$$TRANSICOES = [TR^1, TR^2, \ldots, TR^m]$$

$$CONTROLE = [\Pi TR^{ijk},\ \Pi DEC^{ijk},\ \Pi INC^{ijk}],$$

onde

$$1 \leq i,k \leq n,\quad 1 \leq j \leq m,$$

Logo, há derivação em TR^1 e em todas as fórmulas pertencentes a CONTROLE. Entretanto, como TR^1 esta na forma "LIVRE" e as fórmulas pertencentes a LUGARES estão na forma "CONST", soménte as funções

$$\Pi TR^{i1k},\ \Pi DEC^{i1k},\quad 1 \leq i,k \leq n,$$

pertencentes a CONTROLE, não se comportam como identidades. Tem-se a execução, em paralelo, das funções

$$TR^1,\ \Pi TR^{i1k},\ \Pi DEC^{i1k},\quad 1 \leq i,k \leq n$$

Em seguida, a execução das funções do programa com critério ocorre como indicado na figura 4, considerando-se j=1. No final desta sequência o valor da palavra de controle

$$PC_{trans}$$

é incrementado e a sequência da figura 3.4 é novamente repetida, mas a partir da execução em paralelo das funções,

$$TR^2,\ \Pi TR^{i2k},\ \Pi DEC^{i2k},\quad 1 \leq i,k \leq n$$

Seguindo este raciocinio, as funções TR^j, $1 \leq j \leq m$, que representam as transições da rede, são executadas. Observe que após a execução de

$$TR^m$$

a palavra de controle PC_{trans} é decrementada, fazendo,

$$PC_{trans} = 1$$

o que possibilita novamente a execução da função

$$TR^{1}$$

iniciando um novo ciclo de execução das funções que representam a rede. O resultado da execução das funções pertencentes à lista LUGARES são constantes associadas aos símbolos para tipo int_i onde $1 \leq i \leq n$. Tais resultados são armazenados nos λ-programas

$$\lambda\text{-}P_0,\ \lambda\text{-}P_1,\ \ldots,\ \lambda\text{-}P_n,\ldots$$

e representam a configuração das marcas na rede de Petri.

(ii) Sequencia de derivacao com memoria.

A seqüência de derivação com memória é dada por

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_1, \mathbb{D}\rangle \ \ldots, \langle \lambda\text{-}P_n, \mathbb{D}\rangle, \ldots$$

onde

$$\lambda\text{-}P_0 = \lambda\text{-}P$$

Considerando

$$\lambda\text{-}P_i = [\Phi_1, \ldots, \Phi_m]$$

então,

$$\lambda\text{-}P_{i+1} = [\Psi_1, \ldots, \Psi_m \mid \lambda\text{-}P_i]$$

onde Ψ_j é uma derivação em Φ_j conforme

$$\mathbb{D}\left[\Phi_j, \lambda\text{-}P_i\right]$$

e

$$\langle \lambda\text{-}P_i, \mathbb{D}\rangle$$

são programas com critério para todo i.

A representação de conhecimentos nesta seqüência é análoga ao caso anterior. A única diferença é que na seqüência de derivação com memória a última configuração de marcas da rede de Petri e as configurações intermediárias são todas representadas nos λ-programas $\lambda\text{-}P_i$.

APENDICE B

TRANSFORMACAO DE UMA SEQUENCIA DE DERRIVACAO POR RETROCESSO EM UMA SEQUENCIA DE DERIVACAO "FORWARD".

Considera-se neste apendice a transformacao da sequencia de derivacao por retrocesso proposta no exemplo 3.1 em uma sequencia de derivacao "forvard".

B.1 INTRODUCAO.

É proposto no exemplo 3.1 uma sequência de derivação com sucesso a partir de

$$\langle \lambda\text{-}P,\ \theta_1,\ OB_1 \rangle$$

onde

$$\lambda\text{-}P = [\text{BENEDITO, SONIA, AGUINALDO,}$$

$$\text{PAI BENEDITO SONIA,}$$

$$\text{ESPOSA SONIA AGUINALDO,}$$

$$\text{RELACAO-PRIMITIVA PAI,}$$

$$FC1,\ FC2\ ]$$

$$\theta_1 = \{\ \}$$

e

$$OB_1 = \{\ (\text{RELACAO-FIMILIAR R}) \wedge (\text{R BENEDITO AGUINALDO})\ \}$$

Considera-se a seguir a transformação daquela sequência de derivação por retrocesso em uma sequência de derivação "forward". Para definir a sequência de derivação "forward" adiciona-se ao λ-programa $\lambda\text{-}P$ um conjunto de funções que representa os passos de derivação por retrocesso, obtendo-se um λ-programa $\lambda\text{-}P_0$. Em seguida, obtem-se a sequência "forward" seguindo, em sentido inverso, a sequência de derivação por retrocesso. Finalmente, é definido um critério de derivação $\mathbb{D}$ tal que a fórmula

Φ : (RELACAO-FIMILIAR R) $\wedge$ (R BENEDITO AGUINALDO)

é uma dedução por derivação a partir de

$$\langle \lambda\text{-}P_o, \mathbb{D} \rangle$$

Propõe-se inicialmente, na seção B.2, o λ-programa $\lambda\text{-}P_o$. Este λ-programa é definido adicionando as funções que representam os passo de derivação por retrocesso ao λ-programa $\lambda\text{-}P$. Em seguida, na seção B.3, considera-se a construção da sequência de derivação "forward". Esta sequência é obtida seguindo em sentido inverso a sequência de derivação por retrocesso apresentada no exemplo 3.1. Finalmente, na seção B.4 é proposto um critério de derivação $\mathbb{D}$. O critério definido é tal que Φ é uma dedução por derivação a partir de $\langle \lambda\text{-}P_o, \mathbb{D} \rangle$.

B.2 O λ-PROGRAMA $\lambda\text{-}P_o$.

O λ-programa $\lambda\text{-}P_o$ é definido como sendo o λ-programa $\lambda\text{-}P$, indicado acima, acrescido de funções que representam as operações determinadas pela definição de "Passo de derivação por retrocesso" proposta na seção 3.6.1.

A definição de passo de derivação por retrocesso é constituida por dois conjuntos de operações: a "redução do objetivo" e o "passo de retrocesso". Na sequência de derivação por retrocesso considerada no exemplo 3.1 são utilizadas tres operações de redução do objetivo. Propõe-se as funções

$$id_\alpha, \wedge_\beta \text{ e } \exists_\gamma,$$

para representar, respectivamente, as operações (a), (b) e (d), indicadas na definição 3.6.1. Observe que somente estas operações são utilizadas na sequência de derivação por retrocesso, indicada no exemplo 3.1. A função que representa a operação "passo de retrocesso" não é definida pois esta operação é a execução das funções do λ-programa propriamente dito. Indica-se a seguir as as operações de redução do objetivo e as definições das funções $id_\alpha, \wedge_\beta, \exists_\gamma$.

OPERACOES DE REDUCAO DO OBJETIVO.

(a) Se existe uma fórmula Φ e uma substituição resposta θ tal que $\Phi \in OB$, $\Psi \in \lambda\text{-}P$, $\Psi = \Phi\theta$, então $OB_2 = OB_1 - \Phi$, $\theta_2 = \theta_1\theta$.

Esta operação é representada pelas funções id_α, definidas a seguir. Para cada $\alpha \in \mathbb{T}$ seja

$$id_\alpha = \lambda X_\alpha\ X_\alpha$$

id_α é a função identidade, definida para cada símbolo para tipo $\alpha \in \mathbb{T}$. Observe que dado uma fórmula Φ_α, se existe uma substituição resposta θ tal que

$$\Psi_\alpha = \Phi_\alpha\theta$$

onde $\Psi_\alpha \in \lambda\text{-}P$, então Φ_α é uma derivação em id_α com resposta θ. Neste caso a lista compatível com id_α, utilizada como argumento é $[\Psi_\alpha]$, isto é,

$$\Psi_\alpha = id_\alpha\ \Psi_\alpha$$

Considera-se a seguir a operação (b).

(b) Se existe uma fórmula $\Phi \in OB_1$ e $\Phi = \lambda X_1 \ldots \lambda X_n\ (\Psi_1 \wedge \Psi_2)$, então $OB_2 = (OB_1 - \Phi) \cup (\lambda X_1 \ldots \lambda X_n \Psi_1\ ,\ \lambda X_1 \ldots \lambda X_n \Psi_2)$, $\theta_2 = \theta_1$.

Representa-se esta operação pela função $\wedge_\beta$,

$$\wedge_\beta = \lambda X_\sigma\ \lambda Y_\sigma\ (f\ X_\sigma\ Y_\sigma)$$

onde f é definida a seguir.

(i) Se

$$X_\sigma = \lambda X_1 \ldots \lambda X_n \Psi_1$$

$$Y_\sigma = \lambda X_1 \ldots \lambda X_n \Psi_2$$

onde

$$tipo[\Psi_1] = tipo[\Psi_2] = \iota$$

então

$$f\ X_\sigma\ Y_\sigma = \lambda X_1 \ldots \lambda X_n\ (\Psi_1 \wedge \Psi_2)$$

(ii) Caso X_σ, Y_σ não satisfazem as condições acima, então

$$f\ X_\sigma\ Y_\sigma = T$$

onde T é a constante que representa "true".

Observe que

$$\lambda X_1 \ldots \lambda X_n\ (\Psi_1 \wedge \Psi_2)$$

e uma derivação em $\wedge_\beta$ conforme a lista compatível

$$[\lambda X_1 \ldots \lambda X_n \Psi_1,\ \lambda X_1 \ldots \lambda X_n \Psi_2]$$

Considera-se agora a operação (d).

(d) Se existe uma fórmula $\Phi \in OB_1$ e

$$\Phi = \lambda X_1 \ldots \lambda X_n\ (\sum \lambda Y\ P),$$

$Y \neq X_i\ \forall\ i$, então para alguma variável Z que não ocorre em θ_1, OB_1 e λ-P, então

$$OB_2 = (OB_1 - \Phi) \cup \{\lambda X_1 \ldots \lambda X_n\ ((\lambda Y\ P)Z)\},\ \theta_2 = \theta_1.$$

Esta operação é representada pela função $\exists_\gamma$.

$$\exists_\gamma = \lambda X_\nu\ (\ g\ X_\nu\ )$$

onde g é definida a seguir.

(i) Se

$$X_\nu = \lambda X_1 \ldots \lambda X_n\ ((\lambda Y\ P)Z)$$

então

$$g\ X_\nu = \lambda X_1 \ldots \lambda X_n\ (\sum \lambda Y\ P)$$

(ii) Caso X_ν não satisfaça a condição acima, então

$$g\ X_\nu = T$$

onde T é a constante que representa "true".

Observe que

$$\lambda X_1 \ldots \lambda X_n\ (\sum \lambda Y\ P)$$

é uma derivação em $\exists_\gamma$ conforme o conjunto compatível

$$\{\lambda X_1 \ldots \lambda X_n\ ((\lambda Y\ P)Z)\}$$

Propõe-se então que

$$\lambda\text{-}P_0 = [\ id_\alpha,\ \wedge_\beta,\ \exists_\gamma\ |\ \lambda\text{-}P\ ]$$

onde $\alpha \in \mathbb{T}$.

B.2 A SEQUENCIA DE DERIVACAO "FORWARD".

Define-se a seguir uma sequência de derivação com memória

$$\langle \lambda\text{-}P_0, \mathbb{D}\rangle, \langle \lambda\text{-}P_1, \mathbb{D}\rangle, \ldots, \langle \lambda\text{-}P_8, \mathbb{D}\rangle$$

tal que

(RELACAO-FAMILIAR R)∧(R BENEDITO AGUINALDO)

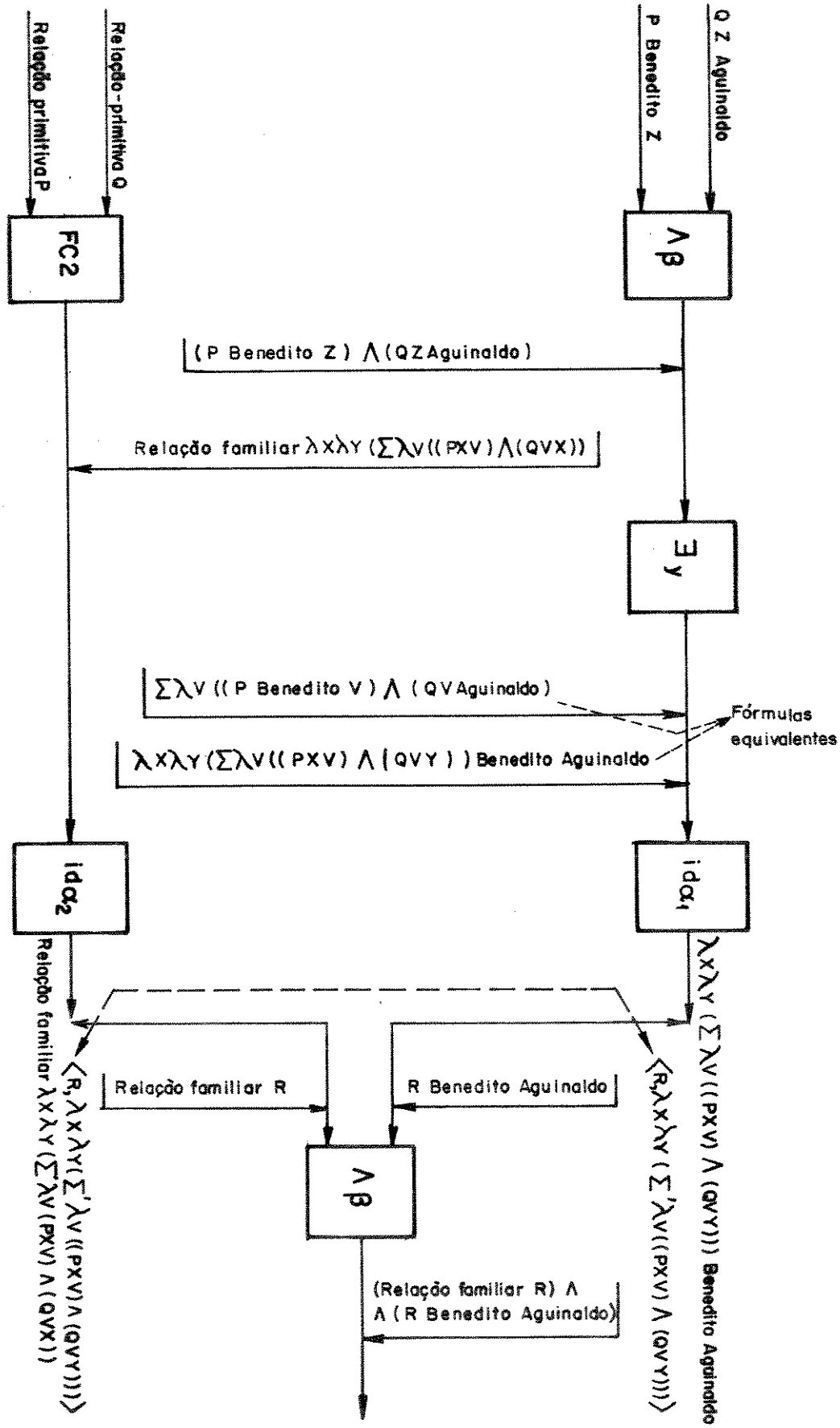
pertence ao λ-programa $\lambda\text{-}P_8$. Isto é, a fórmula acima é uma dedução por derivação a partir do programa com critério $\langle \lambda\text{-}P, \mathbb{D}\rangle$. Para facilitar a definição dos passos da sequência de derivação, considera-se a figura B.1 onde é indicado uma sequência de execuções de funções.

Nesta sequência, todas as funções pertencem a $\lambda\text{-}P_0$ é a fórmula

Φ = **(RELACAO-FAMILIAR R)∧(R BENEDITO AGUINALDO)**

é obtida no final dela. Define-se a seguir os λ-programas $\lambda\text{-}P_i$, $1 \leq i \leq 8$, relacionando-os com as execuções das funções apresentadas na figura B.1. Deve-se observar que é o critério $\mathbb{D}$ que determina a execução, ou não, de cada função. Inicialmente, são apresentadas as funções que são executadas e posteriomente, baseado na sequência de execução de tais funções, define-se o critério $\mathbb{D}$. Define-se a seguir os λ-programas da sequência de derivação.

Sequencia de derivacao.
FIGURA B.1



Sequencia de derivacao.

FIGURA B.1 (CONTINUAÇÃO)

(i)

$$\lambda\text{-}P_1 = [z, \mid \lambda\text{-}P_0]$$

onde "z" é uma derivação em id_α com resposta

$$\Theta_1 = \{\langle z, \text{SONIA}\rangle\}$$

Observe que id_α é a primeira função executada na sequência da figura 3.5. Tem-se a identidade aplicada à constante

SONIA

que pertence a $\lambda\text{-}P_0$.

(ii)

$$\lambda\text{-}P_2 = [\langle\text{ESPOSA z AGUINALDO}\rangle, \langle\text{PAI BENEDITO z}\rangle \mid \lambda\text{-}P_1]$$

onde

(ESPOSA Z AGUINALDO)

e uma derivação em

λXλY (ESPOSA X Y)

conforme a lista compatível

[AGUINALDO, Z]

Analogamente,

(PAI BENEDITO Z)

e uma derivação em

(λXλY PAI X Y)

conforme a lista compatível

[Z, BENEDITO].

Observe que na sequência da figura B.1, esta duas funções são executadas simultaneamente.

(iii)

$$\lambda\text{-}P_3 = [\langle\text{Q Z AGUINALDO}\rangle, \langle\text{RELACAO-PRIMITIVA Q}\rangle \mid \lambda\text{-}P_2]$$

onde

(Q Z AGUINALDO)

e uma derivação em id_α e

(RELACAO-PRIMITIVA Q)

e uma derivação em

λX REALACAO-PRIMITIVA X,

ambas com substituição resposta

{⟨Q, ESPOSA⟩}

Observe também na figura B.1, que as duas funções acima são executadas simultaneamente. O resultado da execução de tais funções determina o λ-programa $\lambda\text{-}P_3$.

(iv)

$$\lambda\text{-}P_4 = [\langle \textbf{P BENEDITO Z} \rangle, \langle \textbf{RELACAO-PRIMITIVA P} \rangle \mid \lambda\text{-}P_3]$$

onde

$$\langle \textbf{P BENEDITO Z} \rangle$$

é uma derivação em id_α e

$$\langle \textbf{RELACAO-PRIMITIVA P} \rangle$$

é uma derivação em

$$\lambda \textbf{X REALACAO-PRIMITIVA X},$$

ambas com susbstituição resposta

$$\{\langle \textbf{P, PAI} \rangle\}$$

Analogamente ao caso anterior, as funções id_α e

$$\lambda \textbf{X} \langle \textbf{RELACAO-PRIMITIVA X} \rangle$$

são executadas simultaneamente. O resultado da execução de tais funções determina o λ-programa $\lambda\text{-}P_4$.

(v)

$$\lambda\text{-}P_5 = [\langle \textbf{P BENEDITO Z} \rangle \curlywedge \langle \textbf{Q Z AGUINALDO} \rangle),$$
$$[\textbf{RELACAO-FAMILIAR } \langle \lambda \textbf{X } \lambda \textbf{Y } [\textstyle\sum \lambda \textbf{V } \langle \textbf{P X V} \rangle \curlywedge \langle \textbf{Q V Y} \rangle)))),$$
$$\mid \lambda\text{-}P_4]$$

onde

$$\langle \textbf{P BENEDITO Z} \rangle \curlywedge \langle \textbf{Q Z AGUINALDO} \rangle)$$

é uma derivação em $\curlywedge_\beta$ (figura B.1) e

$$[\textbf{RELACAO-FAMILIAR } \langle \lambda \textbf{X } \lambda \textbf{Y } [\textstyle\sum \lambda \textbf{V } \langle \textbf{P X V} \rangle \curlywedge \langle \textbf{Q V Y} \rangle)))$$

é uma derivação em FC2 conforme a lista compatível

$$[\langle \textbf{RELACAO-PRIMITIVA P} \rangle]$$

As funções $\curlywedge_\beta$ e FC2 são executadas simultaneamente,

(vi)

$$\lambda\text{-}P_6 = [\langle\Sigma\ \lambda V\ \langle\langle P\ \text{BENEDITO}\ V\rangle\wedge Q\ V\ \text{AGUINALDO}\rangle\rangle \mid \lambda\text{-}P_4]$$

isto é,

$$\lambda\text{-}P_6 = [\langle\langle\lambda X\ \lambda Y\ \langle\Sigma\ \lambda V\ \langle\langle P\ X\ V\rangle\wedge Q\ V\ Y\rangle\rangle\rangle\ \text{BENEDITO AGUINALDO}\rangle \mid \lambda\text{-}P_4]$$

onde

$$\langle\Sigma\ \lambda V\ \langle\langle P\ \text{BENEDITO}\ V\rangle\wedge Q\ V\ \text{AGUINALDO}\rangle\rangle$$

e uma derivação em $\exists_\gamma$.

(vii)

$$\lambda\text{-}P_7 = [\langle\text{RELACAO-FAMILIAR R}\rangle, \langle\text{R BENEDITO AGUINALDO}\rangle \mid \lambda\text{-}P_6]$$

onde

$$\langle\text{RELACAO-FAMILIAR R}\rangle, \langle\text{R BENEDITO AGUINALDO}\rangle$$

são derivações em id_{α_1} e id_{α_2}, respectivamente, com substituição resposta

$$\{\langle R, \langle\lambda X\ \lambda Y\ \langle\Sigma\ \lambda V\ \langle\langle P\ X\ V\rangle\wedge Q\ V\ Y\rangle\rangle\rangle\rangle\}$$

Além disso, tem-se que

$$\alpha_1 = \text{tipo}[\langle\Sigma\ \lambda V\ \langle\langle P\ \text{BENEDITO}\ V\rangle\wedge Q\ V\ \text{AGUINALDO}\rangle\rangle]$$

$$\alpha_2 = \text{tipo}[\langle\text{RELACAO-FAMILIAR}\ \langle\lambda X\ \lambda Y\ \langle\Sigma\ \lambda V\ \langle\langle P\ X\ V\rangle\wedge Q\ V\ Y\rangle\rangle\rangle\rangle]$$

Tais derivações são indicadas na figura B.1. Observe que elas ocorrem simultaneamente.

(viii)

$$\lambda\text{-}P_8 = [\langle\langle\text{RELACAO-FAMILIAR R}\rangle\wedge\langle\text{R BENEDITO AGUINALDO}\rangle\rangle \mid \lambda\text{-}P_7]$$

onde

$$\langle\langle\text{RELACAO-FAMILIAR R}\rangle\wedge\langle\text{R BENEDITO AGUINALDO}\rangle\rangle$$

e uma derivação em $\wedge_\beta$.

B.4 O CRITERIO DE DERIVACAO $\mathbb{D}$.

O critério $\mathbb{D}$ é agora definido "ponto a ponto". Isto é,

$$\mathbb{D}[id_{\alpha},\ \lambda\text{-}P_0] = \text{SONIA}$$

$$\mathbb{D}[\lambda x \lambda y\ (\text{ESPOSA } x\ y),\ \lambda\text{-}P_1] = [z,\ \text{AGUINALDO}]$$

$$\mathbb{D}[\lambda x \lambda y\ (\text{PAI } x\ y),\ \lambda\text{-}P_1] = [\text{BENEDITO},\ z]$$

$$\mathbb{D}[id_{\alpha},\ \lambda\text{-}P_2] = [(\text{ESPOSA } z\ \text{AGUINALDO})]$$

$$\mathbb{D}[\lambda x\ (\text{RELACAO-PRIMITIVA } x),\ \lambda\text{-}P_2] = [\text{ESPOSA}]$$

$$\mathbb{D}[id_{\alpha},\ \lambda\text{-}P_3] = [(\text{PAI BENEDITO } z)]$$

$$\mathbb{D}[\lambda x\ (\text{RELACAO-PRIMITIVA } x),\ \lambda\text{-}P_3] = [\text{PAI}]$$

$$\mathbb{D}[\wedge_{\beta},\ \lambda\text{-}P_4] = [(P\ \text{BENEDITO } z),\ (Q\ z\ \text{AGUINALDO})]$$

$$\mathbb{D}[FC2,\ \lambda\text{-}P_4] = [(\text{RELACAO-PRIMITIVA } Q),\ (\text{RELACAO-PRIMITIVA } P)]$$

$$\mathbb{D}[\exists_{\gamma},\ \lambda\text{-}P_5] = [(P\ \text{BENEDITO } z),\ (Q\ z\ \text{AGUINALDO})]$$

$$\mathbb{D}[id_{\alpha_1},\ \lambda\text{-}P_6] =$$

$$= [(\lambda x\ \lambda y\ [\textstyle\sum \lambda v\ ((P\ x\ v) \wedge (Q\ v\ y))]\ \text{BENEDITO AGUINALDO})]$$

$$\mathbb{D}[id_{\alpha_2},\ \lambda\text{-}P_6] =$$

$$= [(\text{RELACAO-FAMILIAR } (\lambda x\ \lambda y\ [\textstyle\sum \lambda v\ ((P\ x\ v) \wedge (Q\ v\ y))]))]$$

$$\mathbb{D}[\wedge_{\beta},\ \lambda\text{-}P_7] = [(P\ \text{BENEDITO AGUINALDO}),\ (\text{RELACAO-FAMILIAR } R)]$$

e nos outros casos,

$$\mathbb{D}[\lambda x_1, \ldots, \lambda x_n (A\ F_1, \ldots, F_m),\ \lambda\text{-}P_i] = [x_1, \ldots, x_n]$$

onde $i \in \mathbb{Z}$.

APENDICE C

CONDICOES NECESSARIAS E SEQUENCIAS DE DERIVACAO GERAIS.

Dada uma formula Φ e um programa com criterio $\langle \lambda\text{-}P, \mathbb{D} \rangle$, considera-se neste apendice a analise das condicoes necessarias para que Φ seja uma deducao por derivacao a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ quando a sequencia de derivacao e qualquer.

C.1 INTRODUCAO.

Dado uma fórmula Φ e um programa com critério $\langle \lambda\text{-}P, \mathbb{D} \rangle$, os teoremas demonstrados no capítulo 5 estabelecem condições necessárias para que Φ seja uma dedução por derivação a partir de $\langle \lambda\text{-}P, \mathbb{D} \rangle$ quando a sequência de derivação é restrita e neutra. A seguir é apresentado uma análise sobre a dificuldade em se estabelecer resultados análogos para o caso em que a sequência de derivação é geral. Para tanto, considera-se os limitantes superiores e inferiores para o número de β-redex das fórmulas pertencentes a uma sequência

$$\Phi \blacktriangleright \ldots \blacktriangleright \Psi$$

onde cada operação "$\blacktriangleright$" representará uma β-redução ou uma η-redução. A análise é iniciada pela definição do redex de uma fórmula Φ e pela demonstração de um conjunto de resultados básicos que relacionaram o redex das fórmulas Φ e Ψ, onde Ψ é uma β-redução ou η-redução de Φ. É mostrado que a dificuldade em se estabelecer resultados análogos aos do capítulo 5, para o caso geral, deve-se à impossibilidade de se obter um limitante superior para o comprimento da sequência indicada acima.

C.2 RESULTADOS BASICOS.

C.2.1 DEFINICAO. Redex de uma formula.

Dada uma fórmula Φ, então o seu redex, indicado por **redex[Φ]**, é o número de β-redex contidos em Φ.

EXEMPLO C.1

Seja a fórmula

$$F = \lambda X\ (X\ [XA\ (XB)])\ (\lambda Y\ C)$$

tal que, $A,B,C,D \in$ **CONST**, então

$$F \blacktriangleright_{\beta} H$$

onde

$$H = (\lambda Y\ C)\ ([(\lambda Y\ C)\ A]\ [(\lambda Y\ C)\ B])$$

observe então que,

$$\mathbf{redex[}F\mathbf{]} = 1$$

e

$$\mathbf{redex[}H\mathbf{]} = 3.$$

Portanto, após uma β-redução é possível haver um aumento no número de β-redex da fórmula.

***

NOTACAO. No que se segue, dada uma fórmula Φ, então o número de ocorrências livres da variável X em Φ é indicado por

$$|\Phi|_X$$

***

C.2.2 PROPOSICAO.

Sejam as fórmulas F_β, H_α e a variável X_α, então,

$$\text{redex}[S^X_H F] \leq \text{redex}[F] + \text{redex}[H]|F|^2_X$$

DEMONSTRACAO.

A demonstração é feita por indução no comprimento de F.

(i) Se $F \in \text{VAR} \cup \text{CONST}$, então

$$S^X_H F = F$$

ou

$$S^X_H F = H$$

e nos dois casos,

$$\text{redex}[S^X_H F] \leq \text{redex}[F] + \text{redex}[H]|F|^2_X$$

(ii) Se F = AB, então,

$$S^X_H F = (S^X_H A)(S^X_H B)$$

Ha três casos a considerar.

(a)

$$\text{redex}[F] = \text{redex}[A] + \text{redex}[B]$$

e

$$\text{redex}[S^X_H F] = \text{redex}[S^X_H A] + \text{redex}[S^X_H B] + 1$$

Este caso ocorre quando

$$A = X$$

e

$$H = \lambda Y\ C$$

Assim, após a substituição, tem-se que

$$S^X_H F = (S^X_H A)(S^X_H B)$$

$$= (\lambda Y\ C)(S^X_H B)$$

Logo, a própria fórmula $S^X_H F$ é um redex e

$$\text{redex}[S^X_H F] = \text{redex}[S^X_H A] + \text{redex}[S^X_H B] + 1$$

Como

$$A = X$$

então

$$S_H^X A = H$$

e

$$redex[S_H^X A] = redex[H]$$

Aplicando a hipótese de indução,

$$redex[S_H^X B] \leq redex[B] + redex[H]|B|_X^2$$

logo

$$redex[S_H^X F] \leq redex[H] + redex[B] + redex[H]|B|_X^2$$

isto é,

$$redex[S_H^X F] \leq redex[B] + redex[H]\ (|B|_X^2 + 1)$$

Como

$$F = XB$$

então

$$redex[S_H^X F] = redex[B]$$

$$|F|_X = |B|_X + 1$$

isto é,

$$|B|_X^2 + 1 \leq |F|_X^2$$

Logo,

$$redex[S_H^X F] \leq redex[F] + redex[H]\ |F|_X^2$$

(b)

$$redex[F] = redex[A] + redex[B]$$

e

$$redex[S_H^X F] = redex[S_H^X A] + redex[S_H^X B]$$

Este caso ocorre quando a fórmula

$$A$$

NAO é do tipo

$$\lambda Y\ C$$

e a própria fórmula

$$AB$$

NAO é um redex. Além disso, esta mesma condição permanece após a substituição. Isto é, a fórmula

$$S_H^X F$$

não é um redex.

Como

$$redex[S_H^X F] = redex[S_H^X A] + redex[S_H^X B]$$

Aplicando a hipótese de indução,

$$redex[S^X_H A] \leq redex[A] + redex[H]|A|^2_X$$

$$redex[S^X_H B] \leq redex[B] + redex[H]|B|^2_X$$

logo,

$$redex[S^X_H F] \leq redex[A] + redex[B] + redex[H](|A|^2_X + |B|^2_X)$$

Como

$$|F|_X = |A|_X + |B|_X$$

então

$$|A|^2_X + |B|^2_X \leq |F|^2_X$$

logo,

$$redex[S^X_H F] \leq redex[F] + redex[H]|F|^2_X$$

(c)

$$redex[F] = redex[A] + redex[B] + 1$$

e

$$redex[S^X_H F] = redex[S^X_H A] + redex[S^X_H B] + 1$$

Este caso ocorre quando

$$A = \lambda Y\ C$$

Tem-se que as fórmulas

$$AB$$

e

$$S^X_H F$$

são redex.

A demonstração deste caso é análoga a do caso anterior.

(iii) Se $F = \lambda Z\ D$, então

$$S^X_H F = F$$

no caso em que $X = Z$, ou

$$S^X_H F = \lambda Z\ (S^X_H D).$$

no caso em que $X \neq Z$. No primeiro caso, o resultado é óbvio e no segundo tem-se,

$$redex[S^X_H F] = redex[S^X_H D]$$

Aplicando da hipótese de indução,

$$redex[S^X_H D] \leq redex[D] + redex[H]|D|^2_X$$

Como $X \neq Z$, então

$$|F|_X = |D|_X$$

Tem-se também que

$$\text{redex}[F] = \text{redex}[D]$$

logo

$$\text{redex}[S_H^X F] \leq \text{redex}[F] + \text{redex}[H]|F|_X^2$$

CQD.

***

C.2.3 COROLARIO.

Seja Φ o β-redex a seguir,

$$\Phi = (\lambda X\ F)H$$

tal que,

$$\Phi \blacktriangleright_\beta G$$

onde

$$G = S_H^X F$$

então,

$$\text{redex}[G] \leq \text{redex}[F] + \text{redex}[H]|F|_X^2$$

DEMONSTRACAO.

E imediata.

***

C.2.4 PROPOSICAO.

Seja Φ o η-redex a seguir,

$$\Phi = \lambda X\ (FX)$$

onde

$$\Phi \blacktriangleright_\eta F$$

então,

$$\text{redex}[\Phi] \leq \text{redex}[F] + 1$$

DEMONSTRACAO.

Se FX não é um β-redex, então

$$\mathbf{redex}[\Phi] = \mathbf{redex}[F]$$

e caso contrário,

$$\mathbf{redex}[\Phi] = \mathbf{redex}[F] + 1.$$

Portanto,

$$\mathbf{redex}[\Phi] \leq \mathbf{redex}[F] + 1$$

CQD.

***

C.2.5 DEFINICAO. Numero base de uma formula.

Dada uma fórmula Φ, o número base de Φ, indicado por **base**[Φ] é dado por:

$$\mathbf{base}[\Phi] = \max\{\, |F|_X ;\ (\lambda X\ F) \text{ é subfórmula de } \Phi \,\}$$

***

C.2.6 COROLARIO.

Seja Φ o β-redex a seguir,

$$\Phi = (\lambda X\ F)H$$

tal que,

$$\Phi \blacktriangleright_\beta G$$

onde

$$G = S^X_H F$$

então,

$$\mathbf{redex}[G] \leq \mathbf{redex}[\Phi](1 + \mathbf{base}[\Phi]^2)$$

DEMONSTRACAO.

Conforme o corolário A.2.3 tem-se que

$$\mathbf{redex}[G] \leq \mathbf{redex}[F] + \mathbf{redex}[H]|F|^2_X$$

Por definição

$$\mathbf{redex}[F] \leq \mathbf{redex}[\Phi]$$

$$\mathbf{redex}[H] \leq \mathbf{redex}[\Phi]$$

$$|F|_X \leq \mathbf{base}[\Phi]$$

Logo, segue-se o resultado.

***

O próximo corolário reescreve o corolário anterior.

C.2.7 COROLARIO.

Sejam as fórmulas Φ, ξ onde

$$\Phi \blacktriangleright_{\beta} \xi$$

então

$$\mathbf{redex}[\xi] \leq \mathbf{redex}[\Phi](1 + \mathbf{base}[\Phi]^2)$$

DEMONSTRACAO.

Tem-se um raciocínio análogo ao corolário anterior, só que neste caso considera-se também as subfórmulas de Φ.

***

C.2.8 COROLARIO.

Sejam as fórmulas Φ, ξ onde

$$\Phi \blacktriangleright \xi$$

e $\blacktriangleright$ é uma β-redução ou uma η-redução, então

$$\mathbf{redex}[\xi] \leq \mathbf{redex}[\Phi](1 + \mathbf{base}[\Phi]^2)$$

DEMONSTRACAO.

Se $\blacktriangleright$ é uma η-redução, então Φ possui uma subfórmula do tipo

$$\lambda X\ (FX)$$

logo

$$\mathbf{base}[\Phi] \geq 1$$

e a demonstração decorre da proposição A.2.4. Se $\blacktriangleright$ é uma β-redução, a demonstração decorre do corolário A.2.7.

***

C.2.9 PROPOSICAO.

Sejam as fórmulas Φ, $\xi_1,\ldots,\xi_n$ e Ψ onde

$$\Phi \blacktriangleright \xi_1 \blacktriangleright \xi_2 \blacktriangleright \ldots \blacktriangleright \xi_n \blacktriangleright \Psi$$

e $\blacktriangleright$ é uma operação de β-redução ou de η-redução. Então,

$$\mathbf{redex[\Psi] \leq redex[\Phi](1 + base[\Phi]^2)^{n+1}}$$

DEMONSTRACAO.

Pelo corolário A.2.8,

$$\mathbf{redex[\xi_1] \leq redex[\Phi](1 + base[\Phi]^2)}$$

logo,

$$\mathbf{redex[\xi_1](1 + base[\Phi]^2) \leq redex[\Phi](1 + base[\Phi]^2)^2}$$

Novamente, pelo corolário A.2.8,

$$\mathbf{redex[\xi_2] \leq redex[\xi_1](1 + base[\xi_1]^2)}$$

logo,

$$\mathbf{redex[\xi_2] \leq redex[\xi_1](1 + base[\Phi]^2)}$$

pois, por definição,

$$\mathbf{base[\xi_1] \leq base[\Phi].}$$

Portanto,

$$\mathbf{redex[\xi_2] \leq redex[\Phi](1 + base[\Phi]^2)^2}$$

Seguindo este raciocínio,

$$\mathbf{redex[\xi_k] \leq redex[\Phi](1 + base[\Phi]^2)^k}$$

logo,

$$\mathbf{redex[\Psi] \leq redex[\Phi](1 + base[\Phi]^2)^{n+1}}$$

CQD.

***

A proposição anterior não determina um limitante superior para **redex[Ψ]** em função de **redex[Φ]** e **base[Φ]**. Observe que a desilgualdade considera o comprimento da sequência de β-redeuções e η-reduções. Uma conjectura seria encontrar a melhor desigualdade que relaciona **redex[Ψ]**, **redex[Φ]** e **base[Φ]**. A proposição a seguir demonstra que não é possível obter uma relação entre **redex[Ψ]**, **redex[Φ]** e **base[Φ]**, que não dependa do comprimento da sequência de β-reduções e η-reduções.

C.2.10 PROPOSICAO.

Dado $n \geq 0$, então existem fórmulas Φ e Ψ tais que

$$\Phi \blacktriangleright \xi_1 \blacktriangleright \xi_2 \blacktriangleright \ldots \blacktriangleright \xi_n \blacktriangleright \Psi$$

onde $\blacktriangleright$ é uma operação de β-redução ou de η-redução,

$$\mathbf{redex}[\Psi] \geq (\mathbf{base}[\Phi])^{n+1}$$

DEMONSTRACAO.

A demonstração é feita por indução em "n".

(i) Se n = 0, considere

$$\Phi = (\lambda X\ (X\ Y))\ (\lambda Z\ A)$$

e

$$\Psi = (\lambda Z\ A)\ Y$$

onde $A \in \mathbf{CONST}$. Logo,

$$\mathbf{base}[\Phi] = 1$$

$$\Phi \blacktriangleright_{\beta} \Psi$$

e

$$\Psi = (\lambda Z\ A)\ Y$$

$$\mathbf{redex}[\Psi] = 1$$

(ii) Considere o resultado valido para "n-1". Assim, existem fórmulas F e H tais que

$$F \blacktriangleright \xi_1 \blacktriangleright \xi_2 \blacktriangleright \ldots \blacktriangleright \xi_{n-1} \blacktriangleright H$$

onde $\blacktriangleright$ é uma operação de β-redução ou de η-redução,

$$\mathbf{redex}[H] \geq (\mathbf{base}[F])^{n}$$

Considere então

$$\Phi = (\lambda X\ (X \ldots X))\ F$$

onde X ocorre "**base**[F]" vezes na subfórmula

$$(X \ldots X)$$

Aplicando a sequência de β-reduções e η-reduções acima a a partir da fórmula Φ, tem-se

$$\Phi \blacktriangleright \lambda X\ (X \ldots X)\ \xi_1 \blacktriangleright \ldots \blacktriangleright \lambda X\ (X \ldots X)\ \xi_{n-1} \blacktriangleright \lambda X\ (X \ldots X)\ H$$

Aplicando agora uma β-redução na fórmula

$$\lambda X\ (X...X)\ H$$

tem-se

$$\lambda X\ (X...X)\ H \blacktriangleright_{\beta} \Psi$$

onde

$$\Psi = H...H$$

com H ocorrendo "**base**[F]" vezes. Assim, como

$$\mathbf{redex}[H] \geq (\mathbf{base}[F])^{n}$$

logo

$$\mathbf{redex}[\Psi] \geq (\mathbf{base}[F])^{n+1}$$

Por outro lado, como

$$\Phi = \lambda X\ (X...X)\ F$$

onde X ocorre "**base**[F]" vezes em na subfórmula

$$(X...X)$$

então, por definição de **base** de uma fórmula,

$$\mathbf{base}[\Phi] = \mathbf{base}[F]$$

logo,

$$\mathbf{redex}[\Psi] \geq (\mathbf{base}[\Phi])^{n+1}$$

CQD.

***

Na próxima seção é feita a análise das condições necessárias para a dedução por derivação de uma fórmula Φ a partir de um programa com critério. Esta análise se fundamenta nas proposições A.2.8, A.2.9 e no teorema forte da forma normal, que é indicado a seguir.

C.2.11 TEOREMA. Teorema forte da forma normal [Girard, 1987]

Dada uma fórmula Φ, do λ-calculo tipado, toda sequência de β-reduções e η-reduções, a partir de Φ, resulta em uma fórmula normal Ψ. Isto é, toda sequência de β-reduções e η-reduções a partir de Φ é finita.

***

C.3 ANALISE DAS CONDICOES NECESSARIAS.

Dadas duas fórmulas Φ e Ψ tais que Ψ é obtida a partir de Φ por uma série de β-reduções, as proposições 6.2.9 e 6.2.10 estabelecem que não é possível determinar uma desigualdade que relaciona o número de β-redex em Ψ, em função do número de β-redex em Φ. No caso geral, tais números sempre dependem do número de β-reduções e η-reduções utilizadas para se obter Ψ. Isto significa que em geral, dadas as fórmulas Φ e Ψ onde

$$\Phi \blacktriangleright \ldots \blacktriangleright \Psi$$

é uma sequência de β-reduções e η-reduções e Ψ é uma fórmula normal, então nada se conclui sobre o comprimento desta sequência. Pelo teorema forte da forma normal [Girard, 1987], o único resultado que se tem a respeito do comprimento desta sequência é que ela é finita. Observe que Ψ é normal, logo

$$\mathbf{redex}[\Psi] = \emptyset.$$

Assim, o número de β-redex em cada fórmula da sequência de β-reduções a partir de Φ, pode inicialmente aumentar. Entretanto, este número converge para zero quando o comprimento da sequência aumenta.

Dado as fórmulas,

$$F = \lambda X_1 \ldots \lambda X_n (A\ F_1 \ldots F_m)$$

e

$$H = \lambda Y_1 \ldots \lambda Y_r (B\ H_1 \ldots H_s)$$

Se H é uma derivação em F conforme a lista

$$[C_1 \ldots C_p],$$

onde $0 \leq p \leq n$, então H é a forma normal de

$$\Phi = F\ \ C_1 \ldots C_p,$$

Isto é, toda sequência de β-reduções e η-reduções a partir de Φ termina em H. Suponha que exista um limitante superior K para o comprimento de todas as sequências

$$\Phi \blacktriangleright \ldots \blacktriangleright H$$

e que as operações de redução $\blacktriangleright$ são apenas β-redução. Então, pela proposição 5.2.9 teria-se o resultado,

$$\mathbf{ordem}[H] \leq \mathbf{ordem}[F\ C_1 \ldots C_p] + 2K.$$

isto é,

$$H \leq^{dt} F\ C_1 \ldots C_p + 2K$$

ou

$$H \leq^{dp} F\ C_1 \ldots C_p + 2K$$

Para relacionar a complexidade das fórmulas H e F, é estabelecer as condições necessárias, como é feito no capítulo 5, é necessário determinar o limitante superior para o comprimento da sequência

$$\Phi \blacktriangleright \ldots \blacktriangleright H$$

Entretanto, pela proposição A.2.10 isto não é possível no caso geral. Dadas as fórmulas F e H e um número $n > 0$, pode existir uma lista

$$[C_1 \ldots C_p],$$

tal que as fórmulas

$$\Phi = F\ C_1 \ldots C_p,$$

e Ψ satisfazem a proposição A.2.10. Isto é, existe uma sequência

$$\Phi \blacktriangleright \ldots \blacktriangleright \Psi$$

onde $\blacktriangleright$ é uma operação de β-redução ou de η-redução,

$$\mathbf{redex}[\Psi] \geq (\mathbf{base}[\Phi])^{n+1}$$

Neste caso, como

$$\mathbf{redex}[\Psi] \geq (\mathbf{base}[\Phi])^{n+1}$$

então o comprimento da sequência

$$\Phi \blacktriangleright \ldots \blacktriangleright H$$

é maior que n. Isto é, dado um número $n > 0$, no caso geral é possível determinar sequências com comprimento superior a n.

BIBLIOGRAFIA

**Anderson, J., Bower, G., Human Associative Memory, Washington DC, Winston, 1973.

**Andrews, P.B., Resolution on Type Theory, The Journal of Symbolic Logic, Vol. 36, No. 3, Sept. 1971.

**Andrews, P.B., An Introduction to Mathematical Logic and Type Theory: to Truth Through Proof. Academic Press, Inc. 1986.

**Apt, K.R., van Emden, M.H., Contribution to the theory of Logical Programming, JACM, Vol 29, 841-862, 1982.

**Barr, A., Feigenbaum, E., The Handbook of Artificial Inteligence, Vol. 1, Addison-Wesley, 1981.

**Bobrow, D.G., Special Issue on Non-Monotonic Logic, J. Artificial Intelligence, Vol. 13, 1980.

**Bollobás, B., Graph Theory; an Introductory Course, Springer Verlag, 1979.

**Brachman, B.J., Schmalze, J.G., An overview of the KL-ONE Knowledge Representation System, Cognitive Science, 9, 171-216, 1985.

**Buchanan, B.G., Shortliffe E.H., Rule Based Experts Systems - The MYCIN Esperiments of the stanford heuristic Programming Project, Addison-Wesley, 1984.

**Burge, W.H., Recursive Programming Techniques, Addison-Wesley, 1975.

**Casanova, M.A., e outros, Programacao Logica, V Escola de Computacao, 1986.

**Church, A., A Formulation of the Simple Theory of Types, Journal of Symbolic Logic, 5, 1940, pp 56-68.

**Coquand, T., Huet, G., Construction: A Higher-Order Proof System for Mechanization Mathematics, Rapport de Recherche N. 401, INRIA-LABORIA, 1985.

**Dickmann, M.A., Lambda Calculo Tipado, Notas de aula, UNICAMP, 1988.

**Erman, L.D., e outros, The Hearsay II Speach Understanding System: Interactive Knowledge to Resolve Uncertainty, Computing Surveys, 12, 213-253, 1980.

**Feldstein, Controle hierarquico.

**Felty, A., Miller, D., Specifying Theorem Provers in a Higher-Order Logic Programming Language, Thecnical Report MS-CIS-88-12, Dep. of Comp. and Info. Science, Univ. of Pensylvania, 1988.

**Findler, N.V., Associative Networks: The Representation and Use of Knowledge by Computers, Academic-Press, New York, 1979.

**Frege, C.S., Coleção "Os Pensadores", Abril Cultural, 1983.

**Friedman, D.P., The Scheme Programming Language, Prentice-Hall, 1987.

**Forsyth, R., Experts Systems Principles and Cases Studies, Chapman an Hall, 1984. **Frege, G., Os pensadores, Editora Abril, 1983.

**Gallin, D., Intensional and Higher-Order Logic, North-Holland, 1975.

**Girard, J.Y., Lambda Calcul Tipé, Notas de aula, Universidade de Paris, 1987.

**Goldfarb, W.D., The Undecidability of the Second-Order Unification Problem, Theorical Comp. Science 13, 1981, 225-230.

**Hatcher, W.S., Foundations of Mathematics, W.B. Saunders Co, 1968.

**Hayes-Roth, F., Waterman, D.A., Lenat, D.B., Building Experts Systems, Addison-Wesley, 1983.

**Hayes-Roth, B., BB1: An Architeture for Blackboard Systems that Control, Explain, and Learn their Own Behavior, Tech-Report HPP 84-16, University of Stanford, Stanford, 1984.

**Hayes-Roth, B., A Blackboard Architeture for Control, Artificial Intelligence 26, 1985(a), 251-321.

**Hayes-Roth, B., Hewett, M., Learning Control Heuristics in BB1, Tech-Report HPP 85-2, University of Stanford, Stanford, 1985(b).

**Henderson, P., Functional Programming, Application and Implementation, Prentice-Hall, 1980.

**Hilbert, D., Ackermann, W., Gzundzuge der Theoretischen Logick, Springer-Verlag, Berlin, 1928.

**Hindley, J.R., Introduction to Combinators and λ-Calculus, Cambridge University Press, 1986.

**Henkin, L., A Theory of Proposicional Types, Fund. Mathematics, 52, 1963, pp 223-344.

**Hessen, J., Teoria do Conhecimento, Armênio Amado Editora, Portugal, 1987.

**Hogger, C.J., Introduction to Logic Programming, Accademic Press, 1984.

**Huet, G.P., The Undecidability of Unification in Third Order Logic, Information and Control 22, 1973(a), 257-267.

**Huet, G.P., A Mechanization of Type Theory, Proc. of 3rd Intern. Joint Conf. on Artificial Intelligence, 1973(b).

**Huet, G.P., A Unification Algorithm for Typed Lambda-Calculus, Theorical Comp. Science 1, 1975, 27-57.

**Huet, G.P., Lang, B., Proving and Applying Programs Transformation Expressed whit Second-order Patterns, Acta Informactica 11, 31-55, 1978.

**Huet, G., A Mechanization of Type Theory, Rapport de Recherche, INRIA-LABORIA, 1984.

**Huet, G., Formal Structures for Computation and Deduction, Rapport de Recherche, INRIA-LABORIA, 1986.

**Israel, D.J., Beranek, B., The Role of Logic in Knowledge Rrepresentation, IEEE Computer, October 1983.

**Kant, I., Coleção "Os Pensadores", Abril Cultural, 1983.

**Kowalski, R., Logic for Problem Solving, North Holland, 1979.

**Laid, J.E., Newel, A., Rosenbloom, P.S., SOAR: An Architecture for General Inteligence, Artificial Intelligence, 33, pp 1-64, 1987.

**Lippmann, R.P., An Introduction to Computing with Neural Nets, IEEE-ASSP Magazine, Vol. 4, N. 2, April 1987.

**Lipschutz, S., Algebra Linear, MacGraw-Hill, 1972.

**Lloyd, J.W., Foundations of Logic Programming, Springer-Verlag, 1984.

**Lucchesi, C.L., The Indecidability of the Unification Problem for Third Order Languages, Report C S R R 2059, Dept. of Applied Analysis and Computer Science, University of Waterloo, 1972.

**Martins, R.C.B., Moura, A.V., Desenvolvimento Sistemático de Programas Corretos: A abordagem Denotacional, VI Escola de Computacão, Campinas, 1988.

**McCalla, G., Cercone, N., Approaches to Knowledge Representation, IEEE Computer, Vol 16, N. 10, October 1983.

**Meira, S.R.L., Introducão à Programacão funcional, VI Escola de Computacão, Campinas, 1988.

**Miller, D.A., Proofs in Higher-order Logic, PhD dissertation, Carnegie-Mellon University, 1983.

**Miller, D.A., Nadathur, G., A Computational Logic approach to Syntax and Semantics, Techinical Report MS-CIS-85-17, Dep. of Comp. and Info. Science, University of Pennsylvania, 1985(a).

**Miller, D.A., Nadathur, G., Some Uses of Higher-Order Logic in Computational Linguistics, Techinical Report MS-CIS-86-31, Dep. of Comp. and Info. Science, University of Pennsylvania, 1985(b).

**Miller, D.A., Nadathur, G., Higher-Order Logic Programming, Techinical Report MS-CIS-86-17, Dep. of Comp. and Info. Science, University of Pennsylvania, 1986(a).

**Miller, D., A Theory of Modules for Logic Programming, Thecnical Report MS-CIS-86-53, Dep. of Comp. and Info. Science, Univ. of Pensylvania, 1986(b).

**Miller, D.A., Nadathur, G., Higher-order Logic Programming. LNOCS vol. 204, Springer-Verlag, 1986(c).

**Miller, D., Nadathur, G., A Logic Programming Approach to Manipulating Formulas and Programs, Thecnical Report MS-CIS-87-113, Dep. of Comp. and Info. Science, Univ. of Pensylvania, 1987.

**Minsky, M., A Framework for Representing Knowledge: The Psychology of Computer Vision, New York, McGraw-Hill, pp 211-277, 1975.

**Montague, R., Formal Philosophy: Selected Papers of Richard Montague, New Haven, 1974.

**Nadathur, G., A Higher-Order Logic as The Basis for Logic Programming, Ph.D. Dissertation, University of Pennsylvania, 1987.

**Newell, A., Production Systems: Models of Control Structures, Visual Information Processing, W. G. Chase, ed. Academic Press, New York, 1973.

**Nilsson, N.J., Principles of Artificial Inteligence, Tioga Publishing Company, 1980.

**Paulson, L.C., Natural Deduction as Higher-Order Resolution, Journal of Logic Programming, 1986, 3, 237-258.

**Peterson, J.L., Petri Net Theory and The Modeling of Systems, Prentece-Hall Inc, 1981.

**Peterson, J.L., Petri Nets, Computing Surveys, Vol.9, N.3 September, 1977.

**Pietrzykowski, T., A complete Mechanization of Second-Order Type Theory, April 1973, 333-365.

**Quillian, M.R., Semantic Memory: Semantic Information Processing, M. Minsky ed., MIT Press, pp. 27 - 70, 1968.

**Robinson, A., A Machine-Oriented Logic Based on the Resolution Principle, J. ACM, Vol. 12, pp 23-41, 1965.
**Robinson, P.R., Turbo Prolog, Guia do Usuario, McGraw-Hill, 1988.

**Shapiro, S.C., Wookmansee, G.H., A Net Structure Based Relational Question-Answerer, Proc. Int. I Joint Conf. AI, Washington, DC, pp 325-346, 1971.

**Shoenfield, J.R., Mathematical Logic, Addison Wesley, 1967.

**Simis, A., Introducao à Álgebra, Monografias de Matemática 23, IMPA, 1973.

**Simon, D., A Linear Time algorithm for a Subcase of Second-order Instantiations, LNCS, No. 170, Springer Verlag, 1984.

**Souza, J.N., Amaral, W.C., Andrade Neto, M., An Expert System Architecture for Hierarchical Control, 1988 International Conference on System Man and Cybernetics, 1988.

**Souza, J.N., Amaral, W.C., Araújo, E., Complexity Relationship of the Knowledge Representation Based on the Typed λ-calculus, Submetido ao 1989 International Conference on System Man and Cybernetics, 1989.

**Stefik, M., Planning and Meta-Planning, Artificial Intelligence, 16, pp 141-170, 1981.

**Sterling, L., Shapiro, E., The Art of Prolog, MIT PRESS, 1986.

**Taylor, J.H., Frederick, D.K., An Expert System Architeture for Computer-Aided Control Engineering, Proc. of IEEE, vol 72, No 12, December 1984.

**Turbo Prolog, Owner's Handbook, Borland International INC., 1986.

**Walker, M., McCord, M., Sowa, J.F., Wilson, W.G., Knowledge System and Prolog, Addison-Wesley, 1987.

**Winston, P.H., Horn, B.K.P., LISP, Addison-Wesley, 1981.

**Whitehead A.N., Russell, B., Principia Mathematica, Cambridge Univ. Press, Vol.1 1910, Vol.2 1912, Vol.3 1913, Second Edition 1925, 1927.

**Woods, W.A., What's Important About Knowledge Representation?, IEEE Computer, October 1983.

**Zadeh, L., Fuzzy Logic and Aproximate Reasoning, Syntese, Vol. 30, pp 407-428, 1975.