EDSON SATOSHI GOMI

Eng. Eletricista, Escola Politécnica da USP, 1984

INTELIGÊNCIA ARTIFICIAL E PROGRAMAÇÃO EM LÓGICA, E SUAS APLICAÇÕES NOS SISTEMAS DE SUPERVISÃO E CONTROLE DE SISTEMAS DE POTÊNCIA.

Dissertação apresentada à Escola Politécnica da USP para obtenção do título de Mestre em Engenharia Elétrica

Orientador: Prof. Dr. Antonio M. A. Massola,

Departamento de Engenharia de Eletricidade.

São Paulo, 1989

Aos meus pais,

Katsuto e

Satsuki.

Agradecimentos

Ao Prof. Antonio Massola, que, com sua experiência, soube orientar e transmitir sugestões valiosas para a elaboração deste trabalho.

Aos meus pais, que sempre me incentivaram a buscar novos conhecimentos.

À Rosa e à Yoko, que gentilmente ajudaram na edição do texto final.

Ao Antonio Fábio de Lima, pela sua oportuna ajuda na revisão do trabalho e pelos seus comentários.

Ao João Luiz Bosso, pelo seu bom trabalho na realização dos desenhos finais.

À Elebra Controles, pelo apoio concedido.

Aos amigos da Themag Engenharia, que ajudaram na dissipação de dúvidas.

Finalmente, a todos aqueles que colaboraram direta ou indiretamente, para que este trabalho se tornasse realidade.

Edson Satoshi Gomi

RESUMO

Este trabalho apresenta um estudo de aplicações de técnicas de Inteligência Artificial nos Sistemas de Supervisão e Controle de Sistemas de Potência. Em relação à Inteligência Artificial, são abordados métodos de resolução de problemas e os Sistemas Especialistas. Em seguida é feito um estudo de Programação em Lógica, que serve como base teórica para a linguagem PROLOG. Em relação aos Sistemas de Potência são analisadas a finalidade e a importância dos Sistemas de Supervisão e Controle. Por fim, as técnicas de Inteligência Artificial são apresentadas, através de exemplos de aplicação, como novas abordagens que permitem melhorar o desempenho dos Sistemas de Supervisão e Controle dos Sistemas de Potência.

ABSTRACT

This work presents a study of applications of Artificial Intelligence techniques to the Power Systems Supervisory and Control Systems. With respect to Artificial Intelligence, methods of problem solving and Expert Systems are treated. A study of Logic Programming follow, which serves as a theoretical basis for PROLOG language. Concerning to Power Systems, the purpose and importance of Supervisory and Control Systems are analysed. Last, Artificial Intelligence techniques are presented, through application examples, as a new approach that allow performance improvement of Power Systems Supervisory and Control Systems.

Índice

Índice de Figuras

Índice de Tabelas

I - Introdução

Este trabalho tem como objetivo fazer uma análise das aplicações de técnicas de Inteligência Artificial nos Sistemas de Supervisão e Controle de Sistemas de Potência.

O desenvolvimento da automação digital dos Sistemas de Potência tem permitido a criação de funções cada vez mais complexas disponíveis aos operadores desses sistemas. Para a implementação dessas funções, novas tecnologias de hardware e software tem sido aplicadas. Dentre as tecnologias de software pode-se citar as técnicas de Inteligência Artificial.

A implementação de recursos de Inteligência Artificial numa máquina tem como objetivo fazer com que ela execute tarefas que necessitam de inteligência, como raciocínio, visão, fala e audição. Pela sua capacidade de processar informações, um computador é um candidato natural a receber Inteligência Artificial. Um computador é uma máquina que é capaz de encontrar soluções para um problema dado desde que seja escrito um programa adequado para a resolução do problema. De certa forma, a implementação de Inteligência Artificial num computador é equivalente ao problema de escrever um programa que possa construir ou computar soluções de qualquer problema dado a ele.

Mas será que tal problema tem solução? A resposta a tal pergunta está relacionada com alguns conceitos básicos da teoria da computação. Na década de 1930, o matemático inglês A. M. Turing idealizou uma máquina capaz de computar certos tipos de funções, denominadas Turing-computáveis. Nessa mesma década, o matemático norte-americano A. Church enunciou uma tese que

estabelece que qualquer máquina que for construída só computará funções Turing-computáveis. Portanto, aqui tem-se um limite teórico ao que se pode esperar da mecanização da inteligência.

Essa limitação teórica à mecanização da inteligência não impede que se construa sistemas que possuam inteligência limitada, mas que mesmo assim são muito úteis às suas áreas de aplicação. Assim, o estudo da Inteligência Artificial tem avançado com inúmeras contribuições de diversas áreas. Como exemplo pode-se citar os Sistemas Especialistas, que são programas que procuram ter uma performance semelhante à do homem em áreas específicas de aplicação.

A Inteligência Artificial pode ser implementada utilizando-se diversas abordagens, entre elas pode-se fazer uso da Lógica, que é uma ferramenta matemática que fornece estruturas para representação do conhecimento, além de mecanismos de inferência para a resolução de problemas. Das pesquisas relacionadas com a mecanização da Lógica surgiram os Sistemas de Programação em Lógica. Tais sistemas, além de permitirem implementar métodos de resolução de problemas através de técnicas de Inteligência Artificial, fornecem um novo paradigma de programação, denominada de Programação Declarativa.

Um dos resultados dos Sistemas de Programação em Lógica é a linguagem PROLOG ("PROgramming in LOGic"). Esta linguagem permite implementar mecanismos de processamento paralelo. Dentre as dificuldades para se implementar técnicas de Inteligência Artificial pode-se citar o grande número de inferências necessárias para se resolver problemas. O uso de computadores de processamento paralelo pode aumentar a eficiência na resolução de

problemas. A linguagem PROLOG recebeu grande interesse com a divulgação do projeto japonês do computador de 5a. geração, que tem como um dos objetivos desenvolver computadores com arquitetura de processamento paralelo.

Dentre os problemas que se tenta resolver através de técnicas de Inteligência Artificial, alguns estão relacionados com os Sistemas de Supervisão e Controle de Sistemas de Potência. Pode-se citar como exemplos os problemas de Processamento Inteligente de Alarmes e o Diagnóstico de Eventos.

Este trabalho é dividido em itens que abordam temas específicos e que vão sendo relacionados ao longo da dissertação. No capítulo II são analisados algumas técnicas de Inteligência Artificial e os fundamentos de Sistemas Especialistas. No capítulo III é feito um estudo dos conceitos da Lógica e dos Sistemas de Programação em Lógica. Também é abordada a linguagem PROLOG. No capítulo IV são apresentados alguns conceitos relacionados com os Sistemas de Potência, que serão úteis para entender as aplicações de Inteligência Artificial nessa área. Nesse mesmo capítulo é feita uma análise de alguns tipos de aplicações de Inteligência Artificial em Sistemas de Potência. No capítulo V é feita uma análise de como pode ser construído um Sistema Especialista de Diagnóstico de Faltas utilizando a linguagem PROLOG. As considerações finais são apresentadas no capítulo VI.

II - Inteligência Artificial e Sistemas Especialistas.

II.1 - Conceitos iniciais.

O que é Inteligência Artificial? Intuitivamente pode-se definir Inteligência Artificial como a área de estudo que tenta fazer com que máquinas realizem tarefas que envolvem inteligência, como raciocínio, visão, fala e audição. Inteligência requer conhecimento, que geralmente é volumoso, difícil de caracterizar precisamente e está mudando constantemente. A Inteligência Artificial estuda métodos para explorar conhecimento (aquisição e representação do conhecimento) com essas características e técnicas de resolução de problemas. Há interesse de aplicação desses métodos em problemas em que há o fenômeno da explosão combinatória do espaço de pesquisa de uma solução e em problemas que são difíceis de serem estruturados através de técnicas convencionais de programação.

Não houve sucesso em emular a inteligência humana como um todo e então passou-se a estudar sistemas que abrangem áreas específicas como percepção (por exemplo visão, audição e fala), compreensão de linguagem natural e resolução de problemas em campos especializados (por exemplo matemática simbólica, diagnóstico médico, análise química e projeto de engenharia).

As pesquisas na área de Inteligência Artificial resultaram em métodos para representação do conhecimento, especificação de um problema e técnicas de resolução de problemas. Uma das áreas de aplicação desses métodos são os chamados Sistemas Especialistas, que são programas que utilizam conhecimento especializado e regras de inferência para resolver

problemas que são suficientemente difíceis a ponto de requerer um nível de especialização significativo para as suas soluções.

Nos itens seguintes serão analisados alguns métodos de resolução de problemas. Para resolver um problema através de uma computação deve-se representá-lo adequadamente. Uma das abordagens possíveis é através da representação por espaço de estados. Uma vez definida a estrutura da representação do problema, deve-se escolher a direção da busca da solução e os métodos de seleção de possíveis candidatos à resposta. A direção de busca da solução pode ser para frente, também chamada de método de síntese, e para trás, denominada de método de análise. Para controlar a busca da solução devem ser implementados mecanismos de controle. Esses mecanismos elegem os possíveis candidatos à solução, aceitando-os ou descartando-os. Um exemplo de mecanismo de controle são as heurísticas, que são regras geradas pela experiência ou bom senso e que permitem melhorar o desempenho na busca da solução.

Por fim, serão analisados os Sistemas Especialistas, que nada mais são que recursos de programas que implementam alguns métodos desenvolvidos para resolução de problemas.

II.2 - Resolução de problemas.

II.2.1 - Representação de um problema.

A resolução de um problema depende da especificação precisa do mesmo. Esta especificação deve incluir informações sobre as situações iniciais e finais que formam o conjunto de soluções aceitáveis para o problema. Um problema pode ser descrito através de uma **representação por espaço de estados** (figura II.2.1.1). As situações iniciais e finais podem ser descritas por estados iniciais e finais. O movimento de um estado para outro deve ser feito de acordo com um conjunto de regras, que deve fazer parte da especificação do problema. Os dois exemplos seguintes ilustram esse tipo de representação.

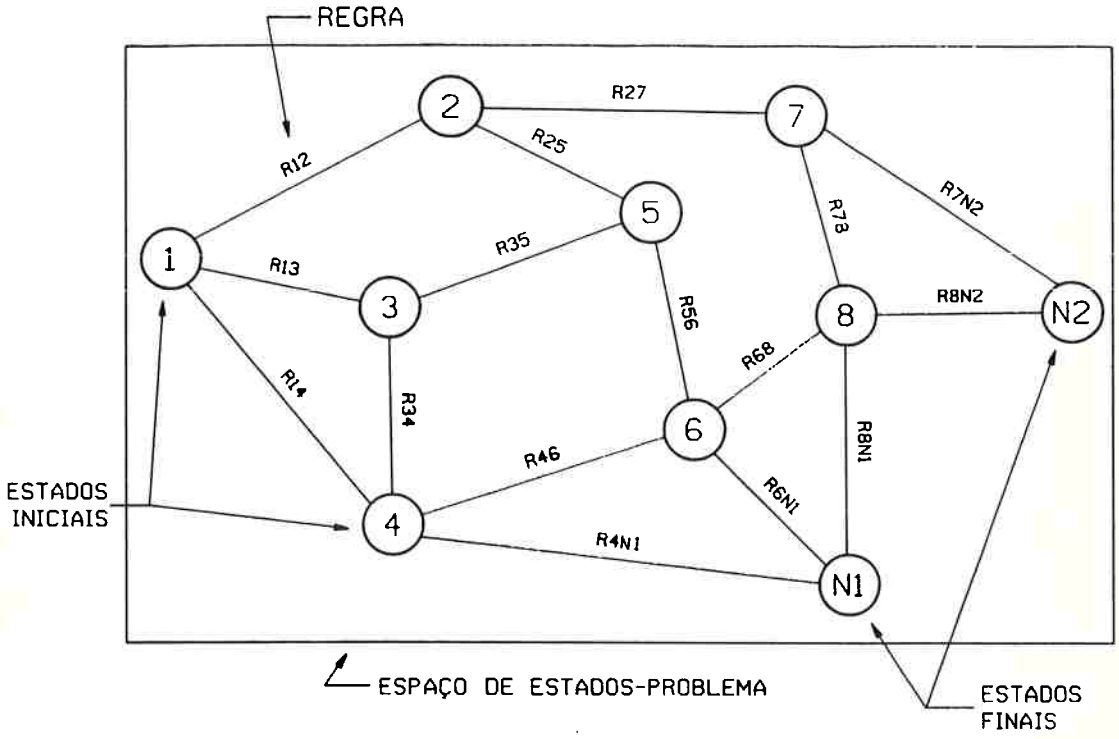


FIGURA II.2.1.1 - REPRESENTAÇÃO DE UM PROBLEMA POR ESPAÇO DE ESTADOS

a) Problema dos jarros de água.

São dados dois jarros: um de 3 litros e outro de 4 litros. Há uma torneira que permite encher os jarros com água. Não há nenhuma medida de volume nos jarros. Como conseguir colocar 2 litros no jarro de 4 litros?

O espaço de estados para este problema pode ser descrito como um conjunto de pares de inteiros (x,y), tal que x = 0,1,2,3 ou 4 e y = 0,1,2 ou 3. O volume do jarro de 4 litros é representado por x e o volume do jarro de 3 litros por y. O estado inicial é (0,0) e o estado que se deseja atingir é (2,n). Observa-se que o problema não especifica qual deve ser o volume final do jarro de 3 litros. Na figura II.2.1.2 tem-se uma representação do problema.

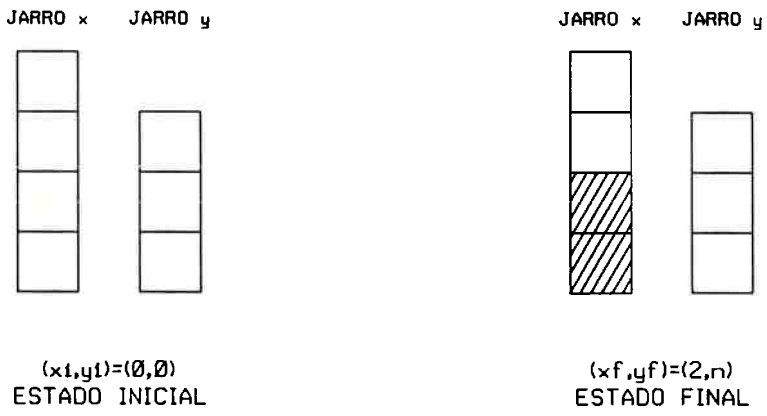


FIGURA II.2.1.2 - ESQUEMA DO PROBLEMA DOS JARROS DE ÁGUA

Os movimentos nesse espaço de estados podem ser descritos pelas seguintes regras:

R1 - se (x,y | x<4) então (4,y).
Encher o jarro de 4 litros.

R2 - se (x,y | y<3) então (x,3).
Encher o jarro de 3 litros.

R3 - se (x,y | x>0) então (x-d,y).
Despejar certa quantidade de água do jarro de 4 litros.

R4 - se (x,y | y>0) então (x,y-d).
Despejar certa quantidade de água do jarro de 3 litros.

R5 - se (x,y | x>0) então (0,y).
Esvaziar o jarro de 4 litros.

R6 - se (x,y | y>0) então (x,0).
Esvaziar o jarro de 3 litros.

R7 - se (x,y | x+y>=4 e y>0) então (4,y-(4-x)).
Despejar água do jarro de 3 litros no jarro de 4 litros até que o jarro de 4 litros esteja cheio.

R8 - se (x,y | x+y>=3 e x>0) então (x-(3-y),3).
Despejar água do jarro de 4 litros no jarro de 3 litros até que o jarro de 3 litros esteja cheio.

R9 - se (x,y | x+y<=4 e y>0) então (x+y,0).
Despejar toda a água do jarro de 3 litros no jarro de 4 litros.

R10 - se (x,y | x+y<=3 e x>0) então (0,x+y).
Despejar toda a água do jarro de 4 litros no jarro de 3 litros.

Para resolver o problema necessita-se, além das informações acima, de uma estrutura de controle que faça aplicações das regras R1 a R10 até que seja atingido o estado

objetivo. Uma das soluções do problema é apresentada a seguir:

Volume do jarro de 4 litros	Volume do jarro de 3 litros	Regra aplicada
0	0	
0	3	R2
3	0	R9
3	3	R2
4	2	R7
0	2	R5
2	0	R9

b) Jogo de xadrez.

O jogo de xadrez pode ser representado por uma matriz 8x8, onde cada posição contém um símbolo que representa uma peça do jogo ou um símbolo indicando que a posição está vazia. A representação do estado inicial é imediata e o estado objetivo pode ser definido como qualquer estado em que o oponente esteja com o seu rei sob ataque e esta peça não tenha qualquer movimento permitido para sair desta situação.

O conjunto de regras para descrever os movimentos do jogo de xadrez é bem mais complexo que o problema dos jarros de água. Uma das regras poderia ser:

se [peão branco na posição(linha 2,coluna j) e
posição(linha 3,coluna j) está livre e
posição(linha 4,coluna j) está livre]
então
[mover peão da posição(linha 2,coluna j)
para posição(linha 4,coluna j)].

A representação de problemas por espaço de estados permite uma definição formal do problema, possibilitando converter uma situação arbitrária (estado inicial) numa situação desejada (estado final) através de um conjunto de operações permitidas (regras). Cada regra representa um movimento no espaço de estados e o método de exploração desse espaço chama-se pesquisa. A direção de uma pesquisa pode ser controlada por diversos mecanismos de estratégia, que se encarregam de selecionar as regras que serão utilizadas.

Essas estratégias de controle devem obedecer a alguns requisitos importantes. Por exemplo, a estratégia deve causar movimento no espaço de estados. Se no problema dos jarros de água adotar-se um mecanismo de controle que utilize sempre a primeira regra aplicável da lista de regras, nunca chegar-se-á a uma solução do problema. Primeiro deve-se encher o jarro de 4 litros, depois o jarro de 3 litros, então despeja-se o jarro de 4 litros, depois o jarro de 3 litros, e continua-se a aplicar essas regras indefinidamente sem se chegar a uma solução.

Além disso a estratégia deve ser sistemática. Por exemplo, pode-se escolher aleatoriamente uma regra de uma lista de regras aplicáveis. Este mecanismo é melhor que o apresentado no item anterior, pois além de causar movimento, pode eventualmente chegar a uma solução. Mas não é sistemático porque num dado momento pode-se utilizar várias regras que não contribuem para que o movimento caminhe em direção a uma solução.

II.2.2 - Métodos básicos para controle de busca da solução.

Os métodos básicos para solução de problemas baseiam-se em uma técnica denominada pesquisa heurística, que pode ser implementada através de várias maneiras. Antes de se apresentar alguns desses métodos, deve-se colocar algumas considerações. Uma pesquisa pode ser feita em duas direções: para frente ("forward"), que parte do estado inicial e tenta atingir o estado objetivo, e para trás ("backward"), que parte do estado objetivo e tenta chegar ao estado inicial. Uma maneira de se representar uma estratégia de pesquisa é através de uma representação em árvore, onde cada nó gera um conjunto de candidatos à solução através de aplicação de regras válidas. Esse processo continua até que surja uma solução. Em outros problemas será mais produtivo representar o processo de busca de solução de um problema através de grafos, que representam os caminhos para se chegar a uma solução.

Uma classe de mecanismos de controle é composta pelas heurísticas. Uma heurística permite selecionar uma região de pesquisa no espaço de estados, aumentando a eficiência do processo de pesquisa. Usando heurísticas, pode-se obter boas (nem sempre ótimas) soluções para problemas difíceis. Quando, para se resolver um problema, não se tem um algoritmo sistemático ou direto pode-se utilizar heurísticas de uso geral, também chamados de métodos fracos. Pode-se citar como heurísticas de uso geral os seguintes métodos: "generate-and-test", "hill climbing", pesquisa por expansão em amplitude ou "breadth-first search", "best-first search". A escolha do método a ser utilizado depende das características do problema e, para aumentar a eficiência, as heurísticas de uso geral devem ser usadas em conjunto com

heurísticas específicas do domínio do problema. A seguir serão analisadas algumas heurísticas de uso geral.

a) Método "Generate-and-Test".

Este método é o mais simples de todos e consiste dos seguintes passos:

1 - Gera-se uma possível solução. Em alguns problemas isso significa alcançar um ponto particular do espaço de estados, em outros escolher um caminho a partir do ponto de partida.
2 - Verifica-se se o ponto alcançado ou o ponto final do caminho escolhido faz parte do conjunto de soluções aceitáveis.
3 - Pára-se se a solução foi encontrada, senão retorna-se ao passo inicial.

Se a geração de possíveis soluções é feita sistematicamente, então esse método encontrará uma solução eventualmente. Infelizmente se o espaço de estados é muito grande, a busca da solução poderá levar muito tempo. A maneira mais direta de implementar esse método é através de pesquisa por expansão em profundidade ("depth-first search") com um mecanismo denominado "backtracking", que permite retornar ao ponto anterior antes de desviarmos para pontos em que não há solução ou em que a pesquisa mostra-se improdutiva.

b) Método "Hill Climbing".

O "hill climbing" é uma variante do método "generate-

and-test", na qual a realimentação proveniente do procedimento de teste é utilizada para decidir qual a direção do movimento no espaço de busca. No "generate-and-test" a função de teste respondia apenas sim ou não, isto é, se a solução havia ou não sido alcançada. No "hill climbing" a função de teste possui uma função heurística que fornece uma estimativa da proximidade do estado gerado com a solução. O "hill climbing" é descrito pelas seguintes etapas:

1 - Gera-se o primeiro candidato da mesma maneira que no método "generate-and-test". Se for a solução, pára-se. Senão continua-se.
2 - Aplica-se um certo número de regras para gerar um novo conjunto de possíveis soluções.
3 - Para cada elemento desse conjunto faz-se o seguinte:
    - Verifica-se se é a solução. Se for, pára-se.
    - Verifica-se se esse elemento está próximo de qualquer dos elementos já testados. Se estiver próximo guarda-se o elemento, senão deve-se esquecê-lo.
4 - Pega-se o melhor elemento encontrado acima e o utiliza-se para gerar o próximo conjunto de possíveis soluções. Esse passo corresponde a mover através do espaço de busca na direção que parecer caminhar mais rapidamente para a solução.
5 - Volta-se para o passo 2.

c) Pesquisa por expansão em amplitude ("Breadth-first search").

Os métodos "generate-and-test" e o "hill climbing" escolhem a primeira solução que encontram ("depth-first search"), são fáceis de se implementar e podem chegar a uma boa solução

rapidamente. Mas esses métodos também podem perder um bom tempo explorando caminhos infrutíferos. Uma alternativa estratégica simples é o método de pesquisa por expansão em amplitude, descrito a seguir.

1 - Constrói-se uma árvore cujo estado inicial é a sua raiz.
2 - Gera-se todos os descendentes através das regras aplicáveis ao estado inicial.
3 - Para cada nó gerado, gera-se todos os seus descendentes, aplicando as regras apropriadas a cada nó.
4 - Continua-se o processo descrito no passo 3 até que alguma regra gere a solução.

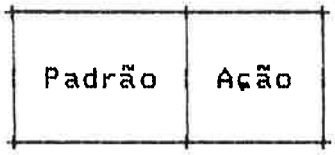
d) "Best-first search".

O método "best-first" é uma maneira de combinar as vantagens dos métodos "depth-first" e "breadth-first". A cada passo desse método seleciona-se os nós mais promissores que foram gerados através da aplicação de uma função heurística a cada um deles. O processo de pesquisa é descrito a seguir.

1 - A partir do nó inicial deve-se gerar todos os seus sucessores.
2 - Uma função heurística estima o custo de se obter uma solução a partir de cada nó.
3 - Gera-se os sucessores do nó mais promissor e retorna-se ao passo 2 até que uma solução seja encontrada.

Como a pesquisa é o metodo básico para a busca de uma solução do problema, é interessante criar uma estrutura que facilite o processo de pesquisa. Essa estrutura pode ser

fornecida, por exemplo, pelos Sistemas de Produção, que consistem de um conjunto de regras descritas através de dois componentes, um que determina a aplicabilidade da regra e outro que determina a ação a ser executada (ver figura II.2.2.1)



O campo Padrão determina a aplicabilidade da regra.

O campo Ação determina a ação a ser executada.

Figura II.2.2.1 - Descrição de uma regra em um Sistema de Produção

Além disso, contém uma ou mais bases de dados que fornecem toda informação necessária para a execução de uma determinada tarefa, estratégias de controle que especificam a ordem na qual as regras serão comparadas com a base de dados e os meios de resolver os conflitos que surgem quando várias regras se aplicam ao mesmo tempo. Essas estratégias tem como requisitos causar movimento no espaço de estados e organizar uma pesquisa sistemática. Estruturas como os Sistemas de Produção são formas de representação do conhecimento, representações que serão analisadas com mais detalhe no próximo item.

II.3 - Sistemas Especialistas.

Uma das áreas da Inteligência Artificial é o raciocínio a partir de um conjunto de conhecimento. Utilizando-se um domínio limitado de conhecimento, é possível construir programas que podem ter uma perfomance humana de raciocínio. Sistemas Especialistas são programas que executam tarefas que podem ser feitas apenas por especialistas que tenham acumulado o conhecimento necessário para a resolução dos problemas. Os Sistemas Especialistas tentam emular algumas partes da inteligência humana. Dessa maneira, os melhores modelos para essa inteligência são pessoas especialistas na área de aplicação do problema. A idéia de um Sistema Especialista é produzir um programa de computador que, ao receber um problema, fornece a mesma solução que seria dada por um especialista, em cujo conhecimento o programa foi baseado. Exemplos de áreas de aplicação são: diagnóstico médico, projeto eletrônico, análise científica e exploração mineral.

Os Sistemas Especialistas diferem substancialmente de programas convencionais porque suas tarefas não tem soluções algoritmicas e frequentemente devem tirar conclusões baseados em informações incompletas ou incertas. Isso sugere que, além de uma estrutura adequada para armazenar e manipular conhecimento, em alguns casos o Sistema Especialista deve possuir mecanismos para manipular probabilidades e fatores de incerteza. Como outras características dos sistemas especialistas pode-se citar que são sistemas altamente interativos, são voltados ao processamento simbólico, utilizam-se de heurísticas para a resolução de problemas e necessitam de profissionais especializados para aquisição do conhecimento necessário. A tabela II.3.1 apresenta

alguns Sistemas Especialistas existentes.

Sistema Especialista	Área de Aplicação	Desenvol-vimento	Linguagem utilizada
MYCIN	Medicina: tratamento de doenças infecciosas	Stanford University	LISP
PROSPECTOR	Geologia: exploração mineral	SRI International	INTERLISP
R1	Computadores: configuração de computadores	Carnegie-Mellon University	OPS5
DENDRAL	Química: pesquisa de estruturas moleculares	Stanford University	INTERLISP

Tabela II.3.1 - Exemplos de Sistemas Especialistas.

Os Sistemas Especialistas estão ligados à Engenharia de Conhecimento, ou seja, à atividade de transferência do conhecimento de um especialista para o programa. O conhecimento do especialista corresponde a estratégias, procedimentos e regras para a resolução dos problemas. Uma das dificuldades da Engenharia do Conhecimento é selecionar um conjunto de estratégias que permita construir uma solução de maneira seletiva

e eficiente num espaço de alternativas. Existem, então, três personagens envolvidos com um Sistema Especialista: o especialista, o engenheiro do conhecimento e o usuário final. O especialista é uma pessoa que por causa do treinamento e experiência é capaz de executar uma atividade de maneira eficiente. Os especialistas sabem reconhecer problemas e propor soluções. O engenheiro do conhecimento é a pessoa capaz de construir um Sistema Especialista, possui conhecimentos de ciência da computação e de Inteligência Artificial, e utiliza-se de ferramentas de construção de Sistemas Especialistas. O usuário é a pessoa que utiliza o sistema construído. Como exemplos de usuários pode-se citar geólogos que querem ajuda para descobrir novos depósitos minerais ou técnicos que desejam especificar uma configuração de um computador de grande porte para um cliente. Pode-se observar na figura II.3.1 as atividades relacionadas com a construção e uso de um Sistema Especialista.

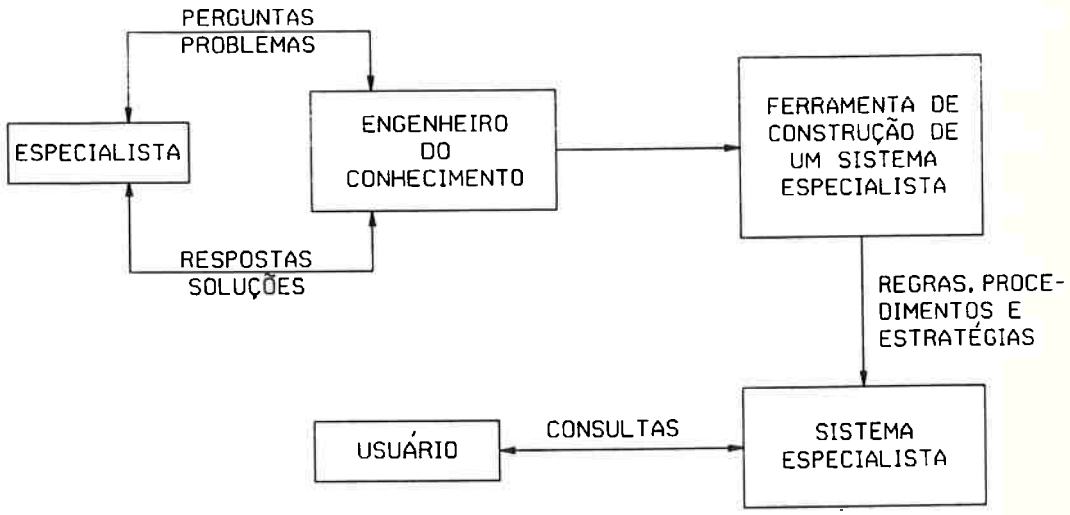


FIGURA II.3.1 - PROCESSOS ENVOLVIDOS COM UM SISTEMA ESPECIALISTA

O engenheiro do conhecimento interage com o especialista através de entrevistas, onde são apresentadas perguntas e problemas, além de obter respostas e soluções. Através de ferramentas adequadas, o conhecimento do especialista é transformado em estratégias, procedimentos e regras, que serão armazenados na base de conhecimento do sistema especialista. O usuário interage com o Sistema Especialista através de uma interface que permite fazer consultas.

O conceito de ferramenta refere-se tanto à linguagem de programação utilizada como ao ambiente de suporte disponível para o desenvolvimento do Sistema Especialista. O ambiente de suporte pode incluir interfaces gráficas sofisticadas, editores fáceis de manusear e ferramentas de depuração. As linguagens de desenvolvimento podem ser baseadas em programação em lógica (por exemplo, PROLOG), programação funcional (LISP), programação baseada em regras (OPS5) ou mesmo linguagens algoritmicas (C, PASCAL, FORTRAN).

Os Sistemas Especialistas apresentam inúmeras vantagens que tornam atrativo o seu desenvolvimento: o conhecimento torna-se permanente, isto é, não se perde quando não se puder contar com os serviços do especialista, torna-se mais fácil de ser documentado e transferido e o custo final torna-se baixo, pois o sistema pode ser duplicado e utilizado por muitos usuários ao mesmo tempo. Além disso, o Sistema Especialista pode apresentar alta performance, dado que o processo de resolução de um problema é feito manipulando-se conhecimento numa velocidade muito maior que a do ser humano. Pode-se citar também o fato de um Sistema Especialista poder manipular e memorizar uma grande quantidade de informações sem erros, a menos que elas não sejam corretas, entre

outras vantagens.

II.3.1 - Estrutura de um Sistema Especialista.

Um Sistema Especialista genérico consiste de vários módulos, que pode-se observar na figura II.3.1.1. O núcleo do Sistema Especialista consiste da máquina de inferência e da base de conhecimento. Ao núcleo está ligado a interface com o usuário e o conjunto todo está apoiado no ambiente de programação e execução, fornecido pelo sistema operacional e o computador. O Sistema Especialista pode contar com o apoio de outros módulos do ambiente de operação como bases de dados, redes de comunicação e sensores. Para a construção e manutenção do sistema especialista existe o ambiente de desenvolvimento, que consiste de ferramentas de aquisição, validação e manutenção do conhecimento, além de ferramentas de projeto de interfaces com o usuário.

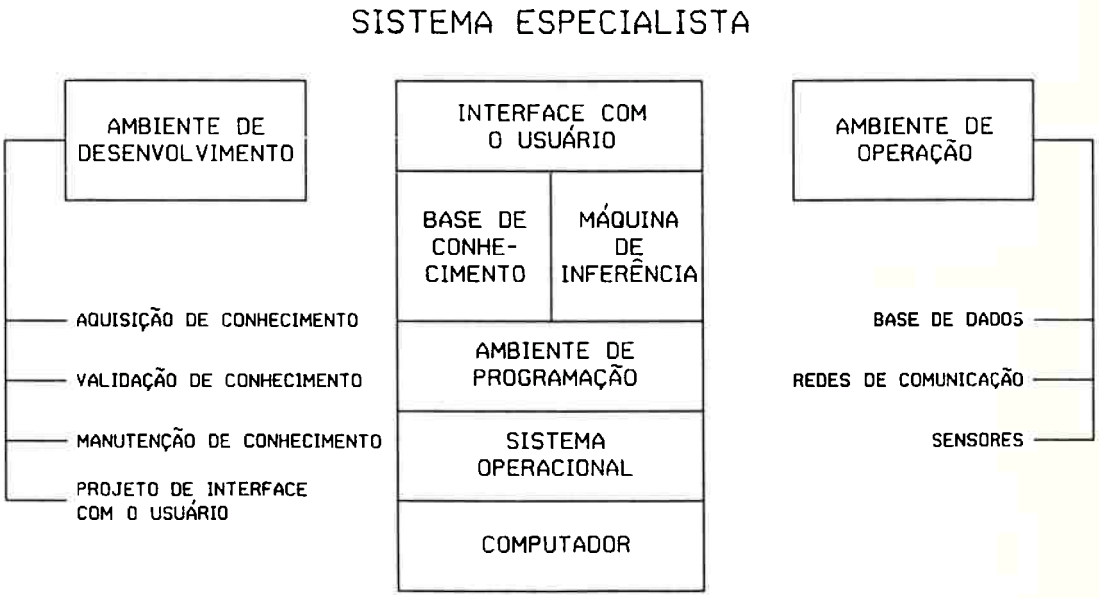


FIGURA II.3.1.1 - ARQUITETURA DE UM SISTEMA ESPECIALISTA

A máquina de inferência e a base de conhecimento estão intimamente relacionados. A máquina de inferência procura a solução de um problema interpretando o conteúdo da base de conhecimento, ou seja, ela é um processador dos símbolos que representam o conhecimento e que procura chegar à solução aplicando mecanismos de inferência aos símbolos. A escolha da estrutura de dados para a representação dos símbolos na base de conhecimento depende do método de pesquisa implementado. A arquitetura do Sistema Especialista e o seu comportamento dependem, então, do método de representação do conhecimento, do mecanismo de inferência e da técnica de pesquisa no espaço de busca da solução que forem implementados.

Na figura II.3.1.2 pode-se observar um esquema mais detalhado da base de conhecimento e da máquina de inferência.

A **base de conhecimento** armazena o conhecimento necessário para a resolução do problema através de vários tipos de representações, como regras de produção, redes semânticas, "frames" e lógica de primeira ordem.

A máquina de inferência é a parte do Sistema Especialista que contém os métodos de resolução do problema. Consiste de um interpretador, que decide como usar os fatos e as regras da base de conhecimento, e de um escalador, que decide quais fatos e regras e em que ordem serão utilizados. Dois processos de inferência podem ser usados em máquinas de inferência, isoladamente ou de maneira combinada. O primeiro processo é denominado **inferência para frente** ("forward reasoning"), onde o sistema parte de um conjunto de fatos e tenta alcançar uma solução mediante o uso das regras. Também é chamado

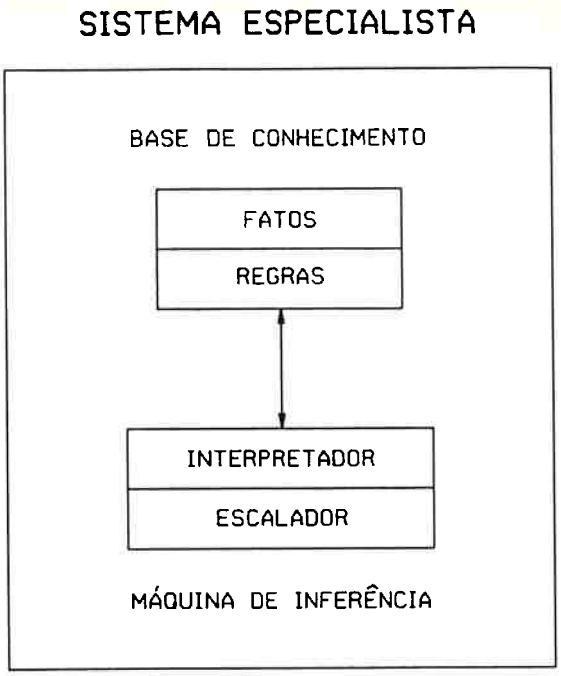


FIGURA II.3.1.2 - ESTRUTURA DE BASE DE CONHECIMENTO E DA MÁQUINA DE INFERÊNCIA

de processo dirigido por dados ou método de síntese. O segundo processo é denominado inferência para trás ("backward reasoning"), onde o sistema parte de uma solução hipotética e tenta encontrar evidências que confirmem a solução. É também chamado de processo dirigido por objetivos ou método de análise. Um sistema baseado em inferência para frente pode nunca chegar a uma conclusão ou mesmo gastar muito tempo gerando alternativas que não contribuem para se chegar à solução. Por esta razão, muitos Sistemas Especialistas funcionam primariamente por inferência para trás, já que com a utilização de heurísticas pode-se iniciar a pesquisa a partir de um conjunto de soluções prováveis.

II.3.2 - Métodos de representação do conhecimento utilizado em Sistemas Especialistas.

Os Sistemas Especialistas podem utilizar vários métodos

de representação do conhecimento, que serão analisados a seguir. A escolha desses métodos influencia a arquitetura da base de conhecimento e da máquina de inferência, e por consequência, o comportamento do Sistema Especialista. Dentre o vários métodos existentes pode-se citar regras de produção, redes semânticas, "frames" e lógica de primeira de ordem.

a) Regras de Produção.

Regras fornecem um meio formal de representar recomendações, diretivas ou estratégias. São representadas por declarações do tipo:

SE (lista de condições) ENTÃO (lista de conclusões).

Tem-se como exemplos de regras:

SE (o pH do líquido é menor que 6) ENTÃO (o líquido é um ácido)

SE (um líquido inflamável é derrubado) ENTÃO (chame o departamento de incêndio)

SE (o carro não parte e o indicador de combustível apresenta leitura zero) ENTÃO (o carro não tem combustível e deve ser abastecido)

Em um Sistema Especialista baseado em regras, o conjunto de regras é comparado com uma coleção de fatos ou conhecimentos sobre a situação corrente. O interpretador de regras compara a porção SE das regras com os fatos e executa aquelas regras cuja porção SE concorda com os fatos. Na figura II.3.2.1 pode-se observar o mecanismo de funcionamento do interpretador de regras. A ação das regras modifica o conjunto de fatos na base de

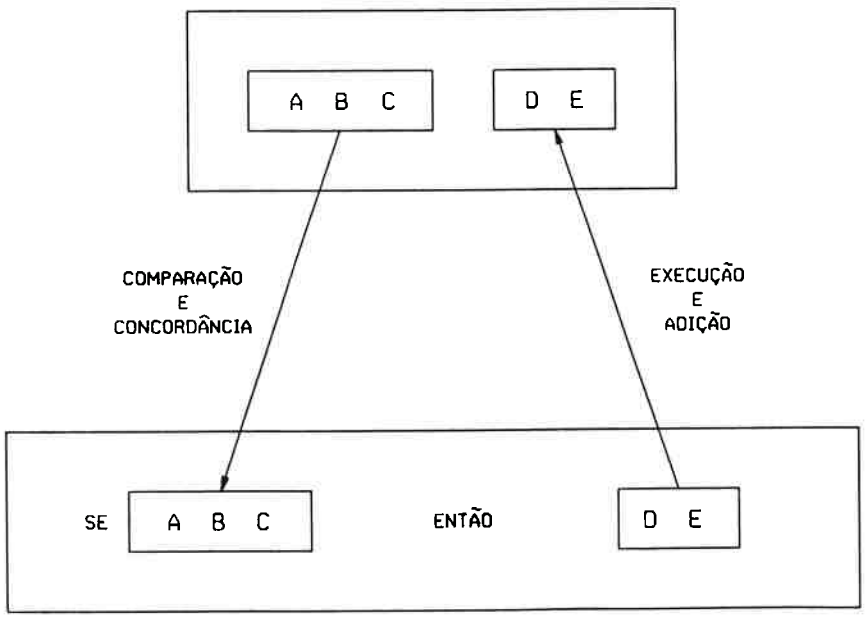


FIGURA II.3.2.1 - MECANISMO DE FUNCIONAMENTO DO INTERPRETADOR DE REGRAS

conhecimento adicionando novos fatos e estes por sua vez podem ser usados para selecionar outras regras. Existem duas maneiras em que as regras podem ser usadas: um método é denominado encadeamento para frente e outro é chamado de encadeamento para trás.

Na figura II.3.2.2 tem-se a ilustração do funcionamento do processo de encadeamento para frente. Inicialmente tem-se um conjunto de fatos A, B, C, E, G, H, que representam o estado inicial e um conjunto de regras:

SE F e B ENTÃO Z

SE C e D ENTÃO F

SE A ENTÃO D.

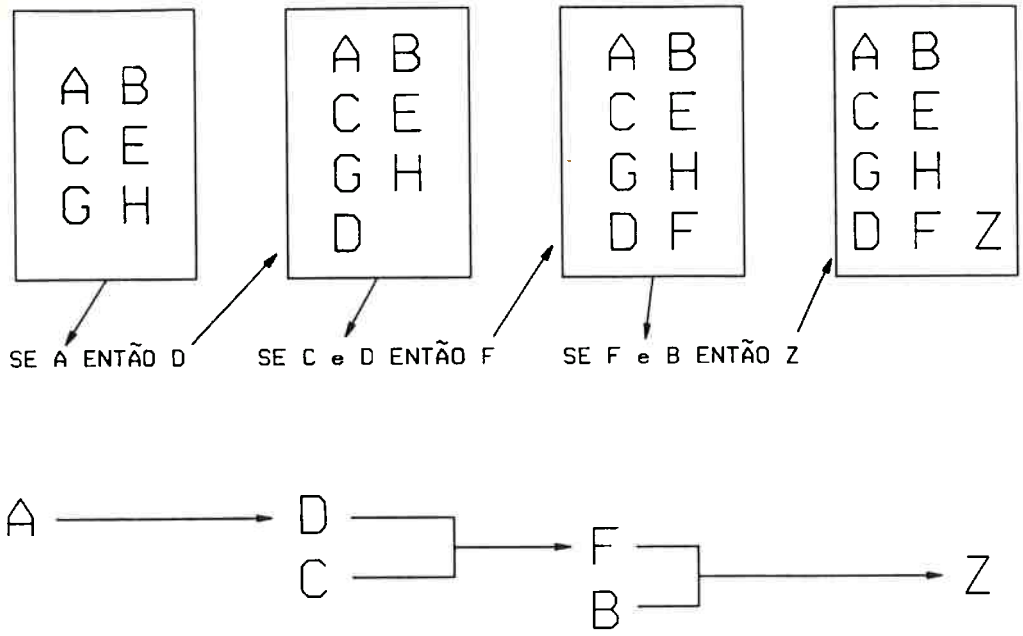


FIGURA II.3.2.2 - ENCADEAMENTO PARA FRENTE

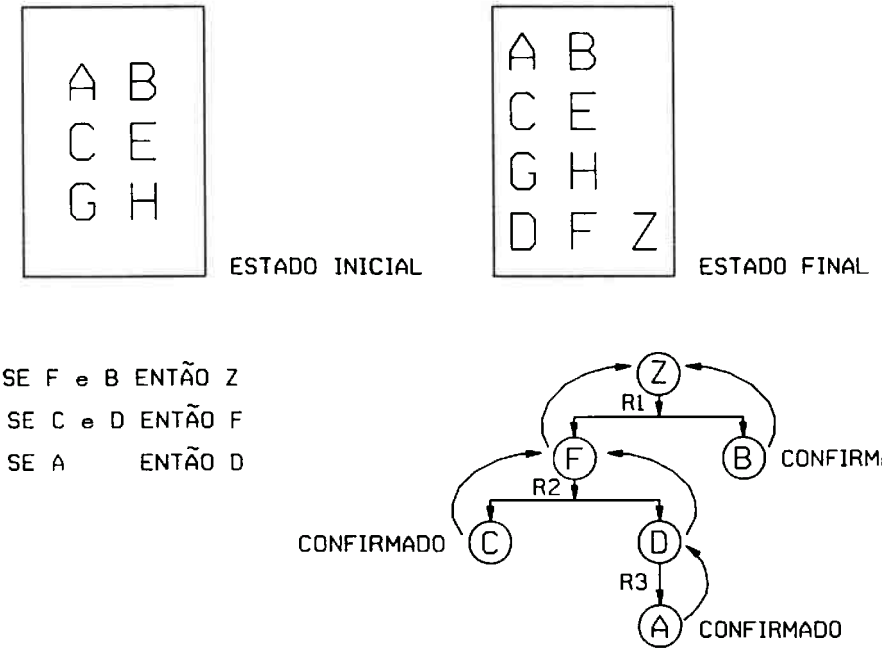


FIGURA II.3.2.3 - ENCADEAMENTO PARA TRÁS

Depois uma regra cuja porção SE concorda com o conjunto de fatos é selecionada pela máquina de inferência e executa-se a porção ENTÃO, que irá adicionar novos fatos. O processo é repetido até que o objetivo seja alcançado.

Na figura II.3.2.3 podem-se observar o funcionamento do método de encadeamento para trás. Nesse processo, se o objetivo for Z, por exemplo, procura-se gerar fatos que confirmem o objetivo. O processo é iniciado comparando-se o objetivo com a porção ENTÃO das regras. Se alguma regra for selecionada, então deve-se verificar se os fatos da sua porção SE já estão no conjunto de fatos. Se estiverem, o objetivo estará confirmado. Os fatos que não estiverem presentes passarão a ser novos objetivos a serem confirmados. O processo se repete até que haja evidências que confirmem ou não os fatos procurados.

b) Redes Semânticas.

Redes semânticas permitem representar relacionamentos entre objetos. Uma rede semântica consiste de pontos denominados nós, que representam objetos, conceitos ou eventos, ligados através de arcos que descrevem as relações entre os nós. Na figura II.3.2.4 podem-se observar a estrutura de uma rede semântica.

Como exemplo simples, a figura II.3.2.5 apresenta uma rede semântica para o conceito de um navio.

c) "Frames".

"Frames" permitem descrever propriedades de objetos e

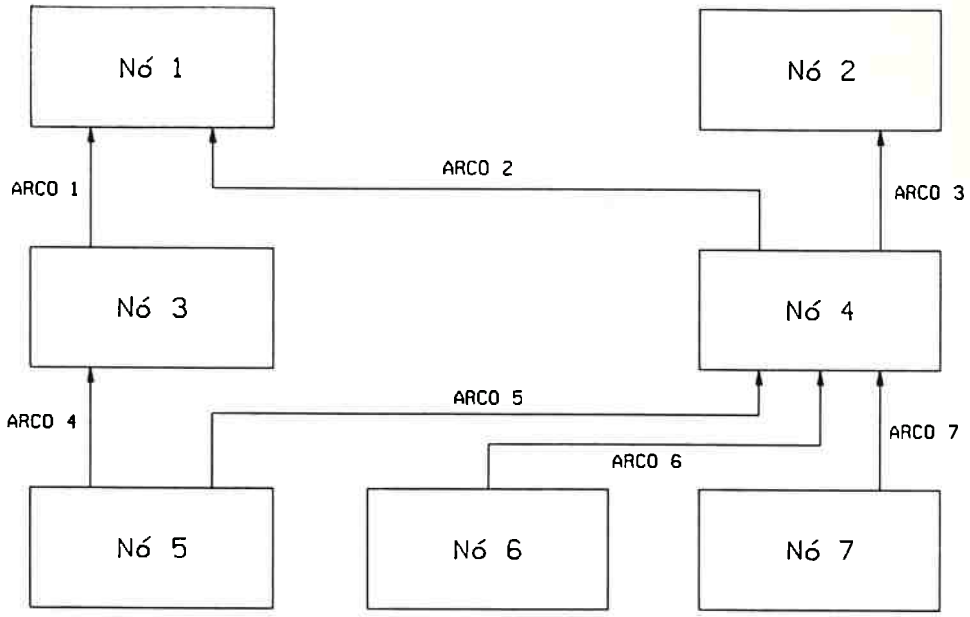


FIGURA II.3.2.4 - ESTRUTURA DE UMA REDE SEMÂNTICA

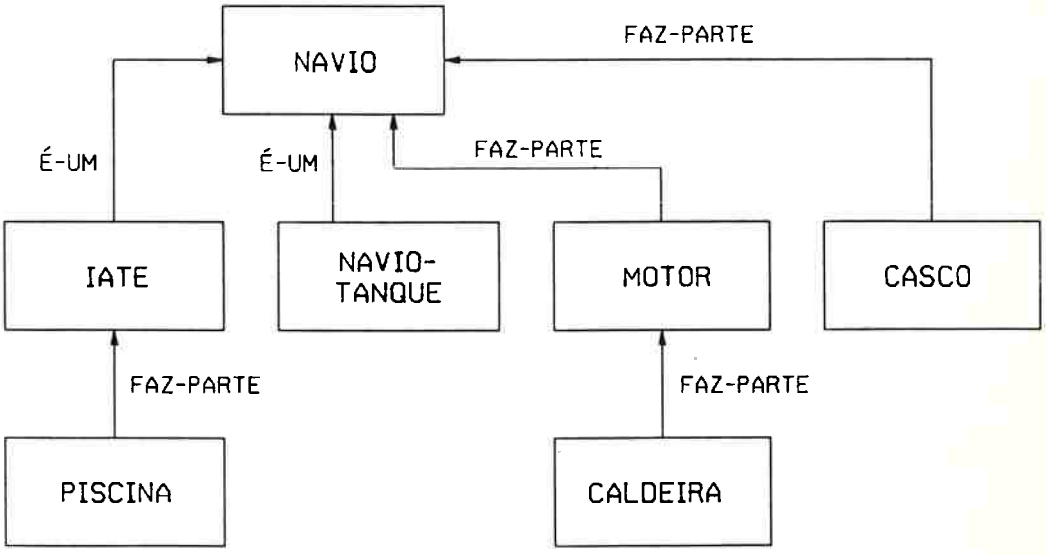


FIGURA II.3.2.5 - REDE SEMÂNTICA PARA O CONCEITO DE UM NAVIO

relações com outros objetos. "Frames" consistem de uma coleção de "slots" que contém descrições de aspectos do objeto, atributos e procedimentos associados a cada "slot". A representação por "frames" é semelhante à da rede semântica, só que os objetos são representados por uma estrutura mais complexa, ao invés de símbolos atômicos. Os nós mais superiores de uma rede de "frames" representam os conceitos mais gerais e os nós inferiores as instâncias específicas desses conceitos.

A figura II.3.2.6 apresenta um exemplo de uma rede de "frames". A cada slot de um "frame" são associados um atributo e procedimentos que são executados quando ocorrem mudanças nos atributos. Existem três tipos de procedimentos: "IF-added procedure", que é executada quando uma informação nova é colocada no "slot", "IF-removed procedure", que é executada quando uma informação é retirada do "slot" e "IF-needed procedure", que é executada quando uma informação é necessária no "slot", mas ele está vazio. A figura II.3.2.7 apresenta a estrutura de um frame.

Como exemplo de operação desses procedimentos, suponha-se que no exemplo da figura II.3.2.6, o gerente geral solicite um relatório de avaliação, ao qual é atribuído o número 15, do "Projeto Secreto 1". Então, o atributo "Projeto Secreto 1" é colocado no "slot Tópico" e um procedimento que procura o nome do líder do projeto é executado. O nome é retirado de uma base de dados e colocado no "slot Autor". Um procedimento é executado para emitir uma mensagem ao líder do projeto que está sendo solicitado um relatório de avaliação referente ao dia DATA e de tamanho COMPRIMENTO. Como falta a informação sobre a data do relatório de avaliação, o sistema solicita-a ao gerente geral e a informação é colocada no "slot DATA". Fica faltando a informação

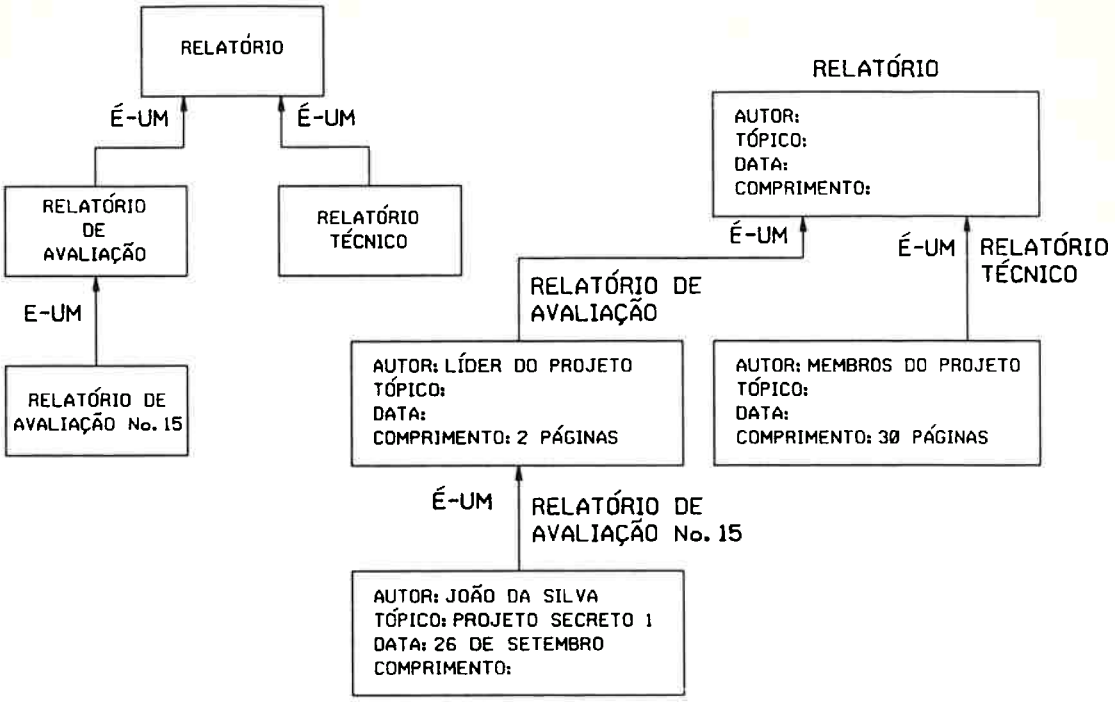


FIGURA II.3.2.6 - EXEMPLO DE UMA REDE DE "FRAMES"

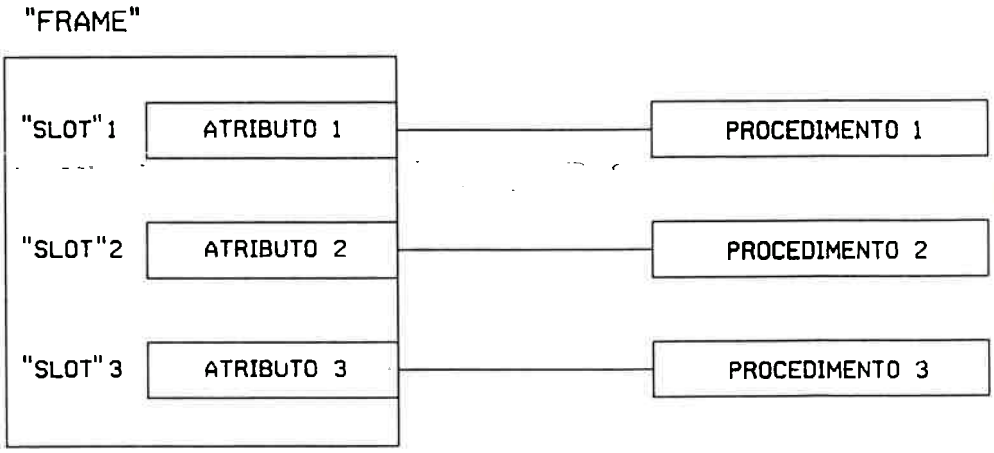


FIGURA II.3.2.7 - ESTRUTURA DE UM "FRAME"

do tamanho do relatório, mas como o relatório de avaliação número 15 é um relatório de avaliação e no "frame" superior existe um valor padrão de duas páginas, que é assumido. O sistema está pronto, então, para emitir a mensagem de solicitação do relatório.

d) Lógica de Primeira Ordem.

A lógica é um sistema formal para representar assertivas e relações entre as mesmas. Por exemplo, "José é estudante." é uma assertiva. Em lógica utiliza-se a notação prefixa, de forma que a sentença anterior é representada por:

estudante(José).

Pode-se utilizar variáveis para representar conceitos como todos, qualquer, algum, nada. Por exemplo:

a) Todos os homens são mortais.
   mortal(X) se homem(X)
b) José gosta de qualquer um que goste de esporte.
   gosta(José,X) se gosta(X,esporte)

Os conceitos da Lógica de Primeira Ordem serão apresentados em mais detalhe no capítulo III.

II.3.3 - Tratamento de incertezas.

Os Sistemas Especialistas podem manipular conhecimento incerto ou resolver problemas em que não hajam informações suficientes. Para isso se utilizam dos chamados fatores de

confiança, que são números que representam o grau de certeza numa dada informação. Os fatores de confiança são aplicadas às informações necessárias para se gerar uma conclusão e ,então, combinadas para gerar um fator de confiança para a conclusão. Através desse mecanismo, os fatores de confiança são propagadas a cada etapa de inferência.

Para ilustração dos fatores de confiança será apresentada a abordagem utilizada pelo MYCIN. No MYCIN as assertivas são associadas a dois números: MB, que é a medida da confiança, e MD, que é a medida da desconfiança.

As medidas de MB e MD, de uma hipótese "h" dado a evidência "e", podem ser calculadas através das seguintes expressões:

$$MB(h,e) = \begin{cases} 1 & \text{se } P(h)=1 \\ \dfrac{\max[P(h|e),P(h)]-P(h)}{\max[1,0]-P(h)} & \text{em caso contrário} \end{cases}$$

$$MD(h,e) = \begin{cases} 1 & \text{se } P(h)=1 \\ \dfrac{\min[P(h|e),P(h)]-P(h)}{\min[1,0]-P(h)} & \text{em caso contrário} \end{cases}$$

onde P(h) = probabilidade da hipótese "h"

P(h|e) = probabilidade de "h", dado a evidência "e"

Dessas duas medidas pode-se calcular o fator de confiança CF:

$$CF(h,e) = MB(h,e) - MD(h,e)$$

Separando a medida de CF em dois componentes, o problema de uma evidência confirmatória "leve" ser interpretada como desconfirmação é evitada. Medidas de CF para uma hipótese "h", dadas duas observações "e1" e "e2" podem ser calculadas por:

$$MB(h,e1 \text{ e } e2) = \begin{cases} 0, \text{ se } MD(h,e1 \text{ e } e2) = 1 \\ MB(h,e1) + MB(h,e2) * (1 - MB(h,e1)), \text{ em caso contrário} \end{cases}$$

$$MD(h,e1 \text{ e } e2) = \begin{cases} 0, \text{ se } MB(h,e1 \text{ e } e2) = 1 \\ MD(h,e1) + MD(h,e2) * (1 - MD(h,e1)), \text{ em caso contrário} \end{cases}$$

$$CF = MB(h,e1 \text{ e } e2) - MD(h,e1 \text{ e } e2)$$

Medidas de CF para uma combinação de hipóteses h1 e h2 podem ser calculadas através das seguintes fórmulas:

$$MB(h1 \text{ e } h2,e) = \min(MB(h1,e), MB(h2,e))$$

$$MB(h1 \text{ ou } h2,e) = \max(MB(h1,e), MB(h2,e))$$

MD é calculado de maneira análoga.

$$CF = MB - MD$$

Como exemplo, se for feita uma observação inicial que confirme a confiança em "h" com MB(h,e1)=0.3, então MD(h,e1)=0 e CF(h,e1)=0.3. Com uma segunda observação também confirma-se "h" com MB(h,e2)=0.2. Daí tem-se que:

MB(h,e1 e e2) = 0.3 + 0.2*(1-0.3) = 0.44

MD(h,e1 e e2) = 0

CF(h,e1 e e2) = 0.44

II.3.4 - Linguagens para Construção de Sistemas Especialistas.

Para a construção de Sistemas Especialistas podem ser usadas linguagens algoritmicas, como PASCAL e FORTRAN, linguagens mais voltadas ao processamento simbólico, como LISP e PROLOG, ou linguagens especializadas para a construção de sistemas especialistas, como o EMYCIN. Cada escolha possui, obviamente, suas vantagens e desvantagens. A utilização de linguagens como LISP ou PROLOG permite maior flexibilidade na escolha da arquitetura da base de conhecimento e da máquina de inferência. Por outro lado demanda-se um esforço maior na construção desses dois módulos, além da interface com o usuário. Uma linguagem especializada para o desenvolvimento de Sistemas Especialistas permite reduzir o tempo de projeto, mas à custa de perda de flexibilidade. A seguir tem-se uma breve descrição de algumas linguagens utilizadas no desenvolvimento de Sistemas Especialistas.

LISP é uma linguagem que tem mecanismos para manipular estruturas de dados do tipo lista. Daí a razão do seu nome: "LISt Processing language". Sua estrutura de controle mais natural é a recursividade, que é apropriada para muitas tarefas de resolução de problemas. Já existem computadores voltados ao processamento da linguagem LISP, as máquinas denominadas de "LISP MACHINES". O programa seguinte é uma definição, em LISP, da operação "append" (união) de duas listas.

```
(DE APPEND (L1 L2)
    (COND ((NULL L1) L2)
          (( ATOM L1) (CONS L1 L2))
          (TRUE (CONS (CAR L1) (APPEND (CDR L1) L2)))))
```

PROLOG é uma linguagem baseada na Lógica de Primeira Ordem. Um programa PROLOG consiste de assertivas e regras para mostrar relações entre objetos. Uma consulta a um programa PROLOG é tentar provar se a sentença que exprime a consulta é consequência lógica das sentenças que fazem parte do programa. Um programa PROLOG utiliza-se do processo de inferência para trás, com auxílio de dois mecanismos denominados de unificação e "backtracking". A linguagem PROLOG será analisada com mais detalhes no item III.

EMYCIN é uma linguagem baseada em representação do conhecimento por regras. Essencialmente o EMYCIN é o Sistema Especialista MYCIN destituído da sua base de conhecimento. A linguagem utiliza inferência para trás, fornece mecanismos para manipular fatores de confiança e possui interfaces para explicar os passos de raciocínio e para aquisição de conhecimento.

OPS5 é uma linguagem que também se baseia em representação do conhecimento por regras e trabalha com o processo de encadeamento para frente. A linguagem suporta várias representações de dados e estruturas de controle, além de possuir um eficiente interpretador de regras.

Notas bibliográficas: o texto do capítulo II é baseado nas referências [3], [5], [6], [7], [10], [14], [16] e [23]. As referência [5] e [7] apresentam uma boa introdução aos conceitos de Inteligência Artificial, abordando métodos de representação do conhecimento e de resolução de problemas. A referência [6] enfoca o uso das técnicas de Inteligência Artificial na Engenharia de Software. As referências [3], [10], [14], [16] e [23] analisam os Sistemas Especialistas.

III - Lógica e Programação em Lógica.

Um raciocínio lógico pode ser definido como um processo de gerar conclusões resultantes de um conjunto de assertivas. A lógica estuda a relação de implicação entre assertivas e conclusões, que são expressas através de sentenças. Ela não trata da verdade ou falsidade de sentenças individuais, mas da relação entre as mesmas. Como exemplo, analise-se as sentenças:

S1 - João é pai de Henrique.

S2 - Henrique é pai de Maria.

S3 - Se a pessoa X é pai da pessoa Z e Z é pai de Y então X é avô de Y.

As sentenças S1, S2 e S3 representam um conjunto de conhecimentos. As ferramentas de lógica não analisam a aceitabilidade dessas sentenças de maneira isolada, mas fornecem meios de examinar as relações entre elas. Então, das três sentenças dadas pode-se concluir que João é avô de Maria. Em outras palavras, diz-se que o fato "João é avô de Maria" é consequência lógica das sentenças S1, S2 e S3.

A lógica fornece um formalismo para a representação de conhecimento e mecanismos para manipulá-la, por isso tem sido aplicada em vários problemas de Inteligência Artificial. Dois exemplos de lógica são a Lógica Proposicional (ou Sentencial) e Lógica de Primeira Ordem (ou de Predicados). As pesquisas sobre a mecanização da lógica levaram à criação dos Sistemas de Programação em Lógica. A seguir, será visto maneira introdutória o que é Programação em Lógica.

Um programa escrito em uma linguagem tradicional, como PASCAL ou FORTRAN, descreve um algoritmo, ou seja, um procedimento que instrui uma máquina a realizar uma determinada computação. Por exemplo, um procedimento escrito em PASCAL para calcular o fatorial de um número "n" seria o seguinte programa:

```
var
     k,n,fatorial: integer;
begin
     for k:=0 to n do
     begin
          if k=0 then
                    fatorial:=1
          else
                    fatorial:=k*fatorial
     end;
end.
```

Um programa em lógica consiste de um conjunto finito de sentenças lógicas que representam o modelo de um determinado problema. Não há a descrição de um procedimento para obter soluções de um problema, mas das informações que permitem obter a solução do problema. Para o exemplo do cálculo do fatorial de um número "n" tem-se as seguintes assertivas:

A1: O fatorial de 0 é 1.

A2: O fatorial de k+1 é (k+1)*(fatorial de k).

Um programa em lógica procura determinar o fatorial de um número "n" aplicando regras de inferência às assertivas A1 e A2. Uma estratégia de aplicação dessas regras de inferência, que

é implícita ao programa, encarrega-se de efetuar o controle da pesquisa da solução.

Um algoritmo pode ser expressado simbolicamente pela equação:

Algoritmo = Lógica + Controle.

O componente lógico do algoritmo representa as informações necessárias para resolver um problema e no componente de controle encontramos a maneira na qual essas informações serão utilizadas. Um programa em lógica representa somente o componente lógico de um algoritmo. O interpretador ou compilador utilizado para processar os programas em lógica fica inteiramente responsável pelo procedimento adotado para a pesquisa da solução. A separação conceitual de um algoritmo em dois componentes permite que ele seja construído e refinado sem que haja a preocupação com o seu mecanismo de controle. Uma vez determinado o componente lógico de um algoritmo podemos melhorar a sua performance através de mudanças no seu componente de controle. O componente lógico expressa o significado de um algoritmo e influencia o seu comportamento, enquanto que o componente de controle determina uma estratégia de resolução do problema e afeta apenas a eficiência do algoritmo, sem afetar o seu significado. Assim, dois algoritmos diferentes, Alg_1 e Alg_2, podem ser obtidos unindo dois mecanismos de controles diferentes, C1 e C2, à mesma lógica L. Os dois algoritmos são equivalentes, pois resolverão o mesmo problema com os mesmos resultados. Simbolicamente temos que Alg_1 e Alg_2 são equivalentes se:

Alg_1 = L + C1 e

Alg_2 = L + C2

A separação de um algoritmo nos dois componentes descritos simplifica o problema de relacionar uma especificação

com o programa. Ignorando o componente de controle, o componente lógico é implicado pela especificação. Com isso temos um método que simplifica a construção de um algoritmo a partir das especificações para a resolução de um problema.

A programação em lógica que será examinada neste trabalho baseia-se nos resultados sobre a mecanização da Lógica de Primeira Ordem ou Cálculo de Predicados. A Lógica de Primeira Ordem fornece uma linguagem formal para descrever um problema e mecanismos para decidir quando uma assertiva é consequência lógica de outras.

Uma das principais características da Programação em Lógica são as noções de consulta e resposta. Uma consulta a um programa em lógica é uma assertiva que exprime as condições a serem satisfeitas por uma resposta correta em função das outras informações contidas no programa.

A maioria dos sistemas para Programação em Lógica reduzem a busca de respostas corretas à pesquisa de refutações a partir das sentenças do programa e da negação da consulta. Uma refutação é a dedução de uma contradição.

Todos esses conceitos serão abordados com mais detalhes a seguir. A idéia de usar Lógica como um formalismo executável em computador levou à criação da linguagem PROLOG ("PROgramming in LOGic"). PROLOG é uma linguagem que permite resolver problemas a partir de sentenças que exprimem a representação simbólica de objetos e seus relacionamentos.

Vamos passar, então, a uma descrição dos fundamentos da

Lógica, abordando Lógica Proposicional e Lógica de Primeira Ordem, e depois à Programação em Lógica e à linguagem PROLOG.

III.1 - Lógica Proposicional

Neste item serão apresentado de forma breve alguns conceitos da Lógica Proposicional. Primeiro será apresentado a construção de fórmulas da Lógica Proposicional através de sua sintaxe e depois as noções de interpretação de fórmulas, e de consequência lógica.

Pode-se escrever fatos do mundo real através de fórmulas da lógica proposicional. Essas fórmulas formam uma linguagem proposicional. Analise-se os seguintes exemplos:

Fatos do mundo real	Fórmulas de lógica proposicional
Está chovendo.	CHOVENDO
Está ensolarado.	ENSOLARADO
Se está chovendo então não está ensolarado.	CHOVENDO $\rightarrow \neg$ ENSOLARADO

Com uma linguagem proposicional pode-se exprimir a estrutura lógica de fatos do mundo real, ou seja, pode-se expressá-los através de sentenças lógicas, concatenadas por conectivos (e, ou, se...então). Uma linguagem proposicional pode ser definida sintaticamente como um conjunto de fórmulas proposicionais, que são escritas a partir de um alfabeto proposicional.

Um alfabeto proposicional consiste de símbolos lógicos e não-lógicos.

Os símbolos lógicos são representados por:

pontuação: (

)

conectivos: $\neg$ (negação)

$\wedge$ (conjunção)

$\vee$ (disjunção)

$\rightarrow$ (implicação)

$\leftrightarrow$ (bi-implicação ou bi-condicional),

Os símbolos não-lógicos são representados por um conjunto de símbolos diferentes dos símbolos lógicos. Por exemplo, CHOVENDO e ENSOLARADO são símbolos não-lógicos.

As fórmulas proposicionais são formadas por símbolos não-lógicos do alfabeto proposicional, isolados ou ligados através dos conectivos lógicos. Assim, se P e Q são fórmulas proposicionais, então $(\neg P)$, $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$ e $(P \leftrightarrow Q)$ também são fórmulas proposicionais.

As regras sintáticas da linguagem que foram apresentadas definem a construção das fórmulas. As regras semânticas capturam o significado pretendido dos conectivos e associam a cada fórmula valores-verdade "falso" (F) ou "verdadeiro" (V). O significado de uma fórmula não é fixado a princípio, mas dependerá de uma atribuição de valores-verdade, especificada em separado. Por exemplo, pode-se associar ao fato CHOVENDO um valor "verdadeiro", ao ENSOLARADO um valor "falso" e utilizando a semântica dos conectivos concluir que o fato CHOVENDO $\rightarrow \neg$ENSOLARADO é verdadeiro. A tabela III.1.1 apresenta a semântica dos conectivos, ou seja, a atribuição de valores-verdade para os conectivos.

P	Q	¬ P	P∧Q	P∨Q	P→Q	P←→Q
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	F	F	V	V	F
F	F	V	F	F	V	V

TABELA III.1.1 - Semântica dos conectivos

Cada conjunto de atribuições de valores-verdade faz parte de uma interpretação. Para as fórmulas CHOVENDO e ENSOLARADO podemos atribuir 4 interpretações, que são apresentadas na tabela III.1.2. Uma interpretação torna consistente um conjunto de fórmulas se todas elas tiverem uma atribuição de valores-verdade "verdadeiro" (V), caso contrário a interpretação torna-o inconsistente. Observando a tabela III.1.2, nota-se que apenas a primeira interpretação torna consistente o conjunto de fórmulas CHOVENDO e ENSOLARADO, e que apenas a segunda interpretação faz consistentes CHOVENDO e ¬ENSOLARADO. Com a noção de consistência e inconsistência pode-se definir informalmente o conceito de consequência lógica. Uma fórmula P é consequência lógica de um conjunto de fórmulas S, ou S implica logicamente P, se e somente se, para toda interpretação em que S for consistente, então P será consistente com S. Voltando à tabela III.1.2, observa-se que se CHOVENDO e ¬ENSOLARADO são verdadeiros, o fato CHOVENDO → ¬ENSOLARADO também é verdadeiro. Logo, para essa interpretação, o fato CHOVENDO → ¬ENSOLARADO é consequência lógica de CHOVENDO e ¬ENSOLARADO.

CHOVENDO	ENSOLARADO	¬ ENSOLARADO	CHOVENDO→¬ENSOLARADO
V	V	F	F
V	F	V	V
F	V	F	V
F	F	V	V

TABELA III.1.2 - ATRIBUIÇÕES DE VALORES-VERDADE

Através da noção de consequência lógica pode-se determinar a validade de uma hipótese num universo expresso por um conjunto de fórmulas S. De uma maneira intuitiva, pode-se dizer que uma fórmula P será consequência lógica de S se e somente se, para qualquer estado deste universo, se todas as fórmulas em S forem verdadeiras neste estado, então P também será verdadeira neste estado.

Para verificar se uma fórmula P é consequência lógica de um conjunto de fórmulas S pode-se usar o método da tabela-verdade. O método da tabela-verdade é um método sistemático para verificar implicação lógica. A idéia é gerar todas as atribuições de valores-verdade de um conjunto finito de fórmulas S e para cada uma que tornar consistente todas as fórmulas de S, verificar se P é consistente com S. Será visto uma aplicação do método da tabela-verdade num exemplo posterior.

Outro conceito importante é o de tautologia. Uma fórmula P1 é tautologicamente equivalente a uma fórmula P2 se e somente se a fórmula P2 for consequência lógica de P1 e vice-versa. A tabela III.1.3 apresenta fórmulas que são tautologicamente

equivalentes. Essas tautologias podem ser facilmente verificadas através do método da tabela-verdade. As tautologias são as verdades lógicas da lógica proposicional.

P1	P2	Nome da tautologia
$A*(B*C)$	$(A*B)*C$	Associatividade
$A*B$	$B*A$	Comutatividade
$A\wedge(B\vee C)$	$(A\wedge B)\vee(A\wedge C)$	Distributividade
$A\vee(B\wedge C)$	$(A\vee B)\wedge(A\vee C)$	Distributividade
$A\rightarrow(B\rightarrow C)$	$(A\rightarrow B)\rightarrow(A\rightarrow C)$	Distributividade
$A\rightarrow B$	$\neg A\vee B$	
$A\longleftrightarrow B$	$(\neg A\vee B)\wedge(\neg B\vee A)$	
$\neg(A\wedge B)$	$\neg A\vee\neg B$	De Morgan
$\neg(A\vee B)$	$\neg A\wedge\neg B$	De Morgan
$\neg(A\rightarrow B)$	$A\wedge\neg B$	
$\neg(A\longleftrightarrow B)$	$(A\wedge\neg B)\vee(\neg A\wedge B)$	
$\neg\neg A$	A	

* = conectivo "$\wedge$" ou "$\vee$".

TABELA III.1.3 - Tautologias da lógica proposicional

Para ilustrar uma aplicação da lógica proposicional será analisado um exemplo (retirado da referência [37], p.18) a seguir:

"Os sofistas, uma espécie de professores viajantes ("sofista" significava mais ou menos o mesmo que "professor"), tornaram-se famosos na Grécia clássica por visitarem cidades ensinando, por um soldo, a arte de argumentar. Eles alcançaram

grande fama e grande habilidade em argumentar a favor ou contra qualquer afirmação, não importando a sua veracidade. A arte que ensinavam, refletida hoje em dia no termo "sofismático", que tem um sentido quase pejorativo, naquela época poderia ser vital pois, se um cidadão fosse acusado de um delito e tivesse que se apresentar perante um tribunal, caberia a ele se defender. Outra pessoa poderia naturalmente preparar a sua defesa, mas não o substituir na defesa em si. Além disto, os juízes não eram profissionais treinados, mas simples cidadãos escolhidos aleatoriamente e, portanto, bastante influenciáveis por defesas bem arquitetadas.

O mais famoso dos sofistas, Protágoras, nasceu em Abdera por volta de 480 AC e morreu por volta de 420 AC. Consta que Protágoras teve um discípulo brilhante, chamado Euathlus. Protágoras o ensinou a arte de argumentar por uma certa quantia, metade da qual seria paga imediatamente e metade após Euathlus ganhar o seu primeiro caso. Euathlus, porém, demorou a pagar a Protágoras, que o processou então.

Protágoras argumentou que se Euathlus ganhasse o caso, ganharia então o seu primeiro caso, logo deveria pagá-lo. Mas se Euathlus não ganhasse o caso, deveria pagá-lo também pois era esta a questão em jogo.

Euathlus, que foi um bom aluno, argumentou da seguinte forma. Se ele ganhasse o caso, não deveria pagar, pois Protágoras perderia a causa. Mas, se ele não ganhasse o caso, não estaria ganhando o seu primeiro caso e, portanto, também não deveria pagar a Protágoras.

Quem está com a razão então?"

Para representar os argumentos de Protágoras e Euathlus pode-se usar as seguintes fórmulas:

A (Euathlus ganha o caso)

B (Euathlus ganha o seu primeiro caso)

C (Euathlus deve pagar a Protágoras)

$\neg C$ (Euathlus não deve pagar a Protágoras)

O argumento de Protágoras pode ser representado pela seguinte fórmula:

P: $((A \rightarrow B) \wedge (B \rightarrow C) \wedge (\neg A \rightarrow C)) \rightarrow C$

O argumento de Euathlus pode ser representado por:

E: $((A \rightarrow \neg C) \wedge (\neg A \rightarrow \neg B) \wedge (\neg B \rightarrow \neg C)) \rightarrow \neg C$

Usando o método da tabela-verdade gera-se todas as atribuições de valores-verdade do problema, que são apresentadas na tabela III.1.4

Para Protágoras receber a quantia devida deve-se mostrar que P e E implicam C, ou seja, quando C for verdadeira P e E também devem ser verdadeiras.

Para Euathlus não pagar a quantia devemos mostrar que P e E implicam $\neg C$, ou seja, quando $\neg C$ for verdadeira P e E também devem ser verdadeiras.

A análise da tabela III.1.4 mostra que ambos teriam razão, mesmo se Euathlus ganhasse o caso (atribuições 1 e 2) ou

se Euathlus não ganhasse o caso. Portanto a análise por lógica é inconclusiva. Uma solução proposta para que Protágoras ganhasse o caso seria: Protágoras deveria deixar Euathlus ganhar o caso e em seguida deveria processá-lo novamente. Desta vez não haveria dúvidas que Protágoras ganharia o caso, pois Euathlus já haveria ganho o seu primeiro caso.

	A	B	C	¬C	P	E
1	V	V	V	F	V	V
2	V	V	F	V	V	V
3	V	F	V	F	V	V
4	V	F	F	V	V	V
5	F	V	V	F	V	V
6	F	V	F	V	V	V
7	F	F	V	F	V	V
8	F	F	F	V	V	V

Tabela III.1.4 - Atribuição de valores-verdade do problema

A lógica proposicional tem algumas limitações. Vamos supor que seja necessário representar os seguintes fatos:

Sócrates é um homem.

Platão é um homem.

Então, poder-se-ía utilizar as seguintes fórmulas:

Sócrates_Homem

Platão_Homem

Uma característica que relaciona Sócrates e Platão é que ambos são homens. Através dos mecanismos da lógica sentencial não é possível estabelecer facilmente a característica em comum entre Sócrates e Platão através das duas fórmulas dadas. A lógica proposicional também não permite representar facilmente o conceito de quantificador universal, que surge, por exemplo, no fato:

Todos os homens são mortais.

Não é possível construir uma fórmula proposicional que represente o conceito dado por esta sentença, a menos que se enumere todos os homens indicando que eles são mortais. Essas dificuldades são contornadas se for utilizada uma lógica com recursos mais poderosos para a representação do conhecimento: a Lógica de Primeira Ordem ou Cálculo de Predicados.

III.2 - Lógica de Primeira Ordem.

A Lógica de Primeira Ordem ou Cálculo de Predicados é um sistema formal que permite descrever um conjunto de conhecimento através de assertivas utilizando uma linguagem formal e resolver problemas através de princípios lógicos.

Para descrever o conhecimento utiliza-se dois conceitos básicos: a existência de objetos, referidos também como indivíduos, e de relações entre eles. Os indivíduos constituem o domínio de um problema. Por exemplo, para o fato "João é pai de Henrique", "João" e "Henrique" são indivíduos e "pai" representa uma relação entre esses dois objetos.

O conhecimento é expresso através de sentenças baseadas nas linguagens de Primeira Ordem. Uma linguagem de Primeira Ordem consiste de termos e fórmulas, que são construídos a partir de símbolos que fazem parte de um alfabeto de Primeira Ordem.

Uma vez construída a descrição do conhecimento passa-se a ter condições de gerar conclusões usando princípios lógicos. Assim como Lógica Proposicional, no Cálculo de Predicados tem-se os conceitos de interpretação e de consequência lógica. É possível verificar se uma sentença P é consequência lógica de um conjunto de sentenças S utilizando-se o método da tabela-verdade, mas naturalmente essa não é a melhor abordagem, principalmente quando se tem um grande número de interpretações. Tem-se um melhor desempenho se forem utilizadas regras de inferência lógica em um processo denominado dedução. Intuitivamente uma dedução é a geração de novas sentenças por meio de regras de inferência lógica. Umas dessas regras é denominada de Resolução.

Os estudos para mecanização dos processos de dedução levaram aos métodos de refutação.

Nos itens III.2.1 a III.2.3 serão apresentados conceitos a respeito da sintaxe das linguagens de Primeira Ordem. Nos itens III.2.4 a III.2.8 serão analisadas as noções de interpretação e consequência lógica, e metodologias para a determinação da consequência lógica.

III.2.1 - Sintaxe das Linguagens de Primeira Ordem.

Um alfabeto de primeira ordem consiste de símbolos lógicos e não-lógicos.

Os símbolos lógicos consistem de símbolos de pontuação, conectivos (negação, conjunção, disjunção, implicação e bi-implicação), quantificadores, variáveis e opcionalmente o símbolo de igualdade. Os símbolos lógicos são apresentados na tabela III.2.1 e estão representados numa forma denominada padrão.

Os símbolos não-lógicos consistem de constantes, símbolos funcionais e símbolos predicativos constituídos por uma sequência de caracteres, de tal forma que todos os símbolos sejam distintos dos demais.

Os termos são variáveis, constantes ou expressões da forma f(t1,t2,...,tn), onde f é um símbolo funcional e t1,t2,...,tn são termos. Como exemplos tem-se:

- variáveis: X, a, B, nome, cont
- constantes: y, A, c, João, casa

- termos f(t1,t2,...,tn): candidato(João,idade(20))

f1(t1,t2(t3,t4),t5)

Pode-se observar que não há uma convenção definida para a escolha dos símbolos que irão representar os termos. Essa escolha é totalmente livre, a princípio, desde que obedeça ao alfabeto das linguagens de primeira ordem.

Símbolo	Representação
Pontuação	(
	)
Conectivos	¬ negação
	∧ conjunção
	v disjunção
	→ implicação
	↔ bi-implicação
quantificadores	∃ quantificador existencial (existe)
	∀ quantificador universal (para todo)
variáveis	um conjunto de símbolos distinto dos demais
igualdade	= (opcional)

Tabela III.2.1 - Representação de símbolos lógicos de um alfabeto de primeira ordem.

As fórmulas consistem de:

a) sequências do tipo p(t1,t2,...,tn), onde p é um símbolo

predicativo e t1,t2,...,tn são termos. As fórmulas desse tipo são chamadas de fórmulas atômicas. Exemplos:

pai(João,Henrique), país(Brasil)

b) sequências do tipo t1=t2, onde t1 e t2 são termos. As fórmulas desse tipo também são chamadas de fórmulas atômicas. Exemplos:

X=1, nome=João

c) sequências do tipo $(\neg P)$, $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, $(P \leftrightarrow Q)$, onde P e Q são fórmulas. Exemplos:

doente(João) $\vee$ sadio(João)

ancestral(x,y) $\wedge$ ancestral(y,z) $\rightarrow$ ancestral(x,z)

d) sequências do tipo $\forall x(P)$ e $\exists x(P)$, onde P é uma fórmula e x uma variável. Exemplos

$\forall$ x(homem(x) $\rightarrow$ mortal(x))

$\exists$ y(homem(y) $\rightarrow$ bom(y))

Através de uma linguagem de primeira ordem podemos exprimir sentenças como:

Sócrates é um homem.

Platão é um homem.

Todo homem é mortal.

As sentenças acima podem ser representadas pelas fórmulas a seguir:

homem(Socrates)

homem(Platão)

$\forall$ x(homem(x) $\rightarrow$ mortal(x))

Sócrates e Platão são constantes, x é uma variável e homem e mortal são símbolos predicativos. Note-se que através do símbolo predicativo "homem" pode-se exprimir uma característica em comum entre Sócrates e Platão. Na terceira fórmula tem-se representado o conceito de quantificação universal contida na sentença "Todo homem é mortal". Portanto, a linguagem de primeira ordem permite superar as limitações encontradas quando se utiliza uma linguagem proposicional.

Um alfabeto de primeira ordem define a sintaxe dos termos e fórmulas de uma linguagem de primeira ordem. A semântica, isto é, o significado dos termos e fórmulas deve ser definido à parte. O significado dos termos e fórmulas depende de uma interpretação para o alfabeto da linguagem, mais precisamente, de um mapeamento associando cada símbolo não-lógico do alfabeto com um elemento, uma função ou uma relação sobre o conhecimento que se quer representar, dependendo do tipo símbolo. Por exemplo, considere-se a fórmula:

estuda(José,Matemática)

O significado pretendido da fórmula poderia ser "José estuda matemática". O símbolo predicativo "estuda" estabelece uma relação entre "José" e "Matemática", que por sua vez são constantes que representam o nome de uma pessoa e de uma disciplina. Da mesma maneira, a fórmula "p(n,d)" poderia ter o mesmo significado ou outro completamente diferente. O fato de

termos e fórmulas admitirem diferentes significados, ao invés de ser um incoveniente, permite extrair propriedades genéricas e comuns a diversos conjuntos de conhecimentos.

Para exemplificar o uso de termos e fórmulas de Primeira Ordem será analisado o exemplo a seguir, que será bastante utilizado nos próximos itens. Cinco pessoas, João, Henrique, Catarina, Maria e José pertencem a uma família. João é pai de Henrique e Catarina. Henrique, por sua vez, é pai de Maria. Por fim, Catarina é mãe de José. Sabe-se também que qualquer pessoa x é avô de y se x é pai de z e z é pai ou mãe de y. Todos esses fatos podem ser representados pelas fórmulas seguintes:

pai(João,Henrique)

pai(João,Catarina)

pai(Henrique,Maria)

mãe(Catarina,José)

$\forall x \forall y \forall z(pai(x,z) \wedge pais(z,y) \rightarrow avô(x,y))$

$\forall x \forall y(pai(x,y) \rightarrow pais(x,y))$

$\forall x \forall y(mãe(x,y) \rightarrow pais(x,y))$

Nestas fórmulas João, Henrique, Catarina, Maria e José são constantes; pai, mãe, avô e pais são símbolos predicativos; e x, y e z são variáveis. Deve ser observado que as variáveis estão universalmente quantificadas.

III.2.2 - Notação Clausal.

A notação clausal é uma representação simplificada da linguagem de primeira ordem na forma padrão. Uma cláusula é uma sequência de fórmulas atômicas, também denominadas literais,

cujas variáveis estão sempre universalmente quantificadas. Os literais da cláusula formam uma disjunção, cuja representação é implícita. Analise-se o seguinte exemplo:

$$\forall x \forall y \forall z (pai(x,z) \wedge pais(z,y) \rightarrow avô(x,y))$$

Considerando-se que as fórmulas da Lógica de Primeira são tautologias se puderem ser mapeadas em tautologias da Lógica Proposicional e considerando que x, y e z estão universalmente quantificadas e utilizando a tautologia que transforma $A \rightarrow B$ em $\neg A \vee B$ resulta que sua representação na forma clausal é:

$$\neg\, pai(x,z)\ \neg\, pais(z,y)\ avô(x,y)$$

Nas referências [1] e [37] são apresentados métodos para transformar fórmulas na representação padrão para a notação clausal.

As fórmulas atômicas ou literais podem ser positivos ou negativos. Exemplos de literais positivos são:

estudante(José)	mortal(x)
pai(João,Henrique)	homem(y)

Exemplos de literais negativos são:

$\neg$ estudante(José)	$\neg$ mortal(x)
$\neg$ pai(João,Henrique)	$\neg$ homem(y)

Os conceitos de símbolos predicativos, símbolos

funcionais, constantes, variáveis, termos e fórmulas atômicas seguem diretamente das definições da forma padrão.

As fórmulas que representam o conhecimento sobre uma família, que foi apresentado no ítem III.2.1, tem a seguinte forma na notação clausal:

pai(João,Henrique)

pai(João,Catarina)

pai(Henrique,Maria)

mãe(Catarina,José)

¬ pai(x,z) ¬ pais(z,y) avô(x,y)

¬ pai(x,y) pais(x,y)

¬ mãe(x,y) pais(x,y)

III.2.3 - Cláusulas de Horn.

Cláusulas de Horn são cláusulas que possuem no máximo um literal positivo. Essa restrição, apesar de limitar os tipos de cláusulas admissíveis, simplifica a aplicação de métodos de inferência lógica. As cláusulas de Horn não diminuem muito o poder de expressão das cláusulas, já que a maioria pode ser transformada num conjunto equivalente de cláusulas de Horn.

As cláusulas de Horn são representadas através de uma sintaxe diferente da notação de cláusulas, conforme veremos a seguir. Uma cláusula de Horn é uma cláusula definida ou uma cláusula-objetivo. Uma cláusula definida é uma expressão da forma:

L←M1,M2,...,Mn

Esta representação é equivalente à cláusula:

L ¬ M1 ¬ M2 ... ¬ Mn

O literal L é chamado de "cabeça" ou "conclusão" da cláusula e os literais M1,M2,...,Mn formam o "corpo" ou "condição". A sequência M1,M2,...,Mn forma uma conjunção e a cláusula definida deve ser interpretada como:

para toda variável xi
L se M1 e M2 e ... e Mn

Uma cláusula-objetivo é uma expressão da forma:

←M1,M2,...,Mn

Esta representação é equivalente à cláusula:

¬ M1 ¬ M2 ... ¬ Mn

O conjunto de cláusulas que expressa o conhecimento da família, apresentada no ítem III.2.1, tem a seguinte forma em cláusulas de Horn:

pai(João,Henrique) ←
pai(João,Catarina) ←
pai(Henrique,José) ←
mãe(Catarina,José) ←
avô(x,y) ← pai(x,z), pais(z,y)
pais(x,y) ← pai(x,y)
pais(x,y) ← mãe(x,y)

III.2.4 - Semântica da forma clausal.

Sintaxe trata da gramática das sentenças, por outro lado, semântica analisa do significado das mesmas. Em lógica as

sentenças podem ser representadas por cláusulas. A cada cláusula pode ser associado um valor "verdadeiro" (V) ou "falso" (F). Essa associação é relacionada com a noção de interpretação e consistência de um conjunto de cláusulas. Se todos os indivíduos de um conjunto de cláusulas forem representados por termos diferentes entre si, então é chamado de universo de discurso a coleção de todos os indivíduos descrito por um conjunto de cláusulas. Por exemplo, considere-se o seguinte conjunto de cláusulas:

A1 humano(João) <-
(João é humano)
A2 mortal(x) <- humano(x)
(se x é humano então x é mortal)
A3 brasileiro(João) <-
(João é brasileiro)

Neste caso o universo de discurso contém apenas um indivíduo, que é "João". Uma interpretação é relacionar cada símbolo predicativo das cláusulas com um elemento do universo de discurso associando a cada literal resultante um valor verdadeiro ou falso. Nas cláusulas A1 a A3 tem-se 3 símbolos predicativos (humano, mortal e brasileiro) que podem ser relacionados com "João", assumindo valores "verdadeiro" (V) ou "falso" (F), num total de 8 interpretações, conforme a tabela III.2.4.1

Apenas a primeira interpretação torna os literais humano(João), brasileiro(João) e mortal(João) verdadeiros. Neste caso diz-se que o conjunto de cláusulas é consistente para essa interpretação. As demais interpretações tornam o conjunto de cláusulas inconsistente.

	humano(João)	brasileiro(João)	mortal(João)
1	V	V	V
2	V	V	F
3	V	F	V
4	V	F	F
5	F	V	V
6	F	V	F
7	F	F	V
8	F	F	F

Tabela III.2.4.1 - Interpretações da cláusulas A1 a A3

É importante ressaltar que a noção de consistência depende do conjunto de cláusulas que representa o problema. Por exemplo, dado o conjunto de cláusulas:

B1 mulher(x) ← mãe(x,y)
(se x é mãe de y então x é mulher)

B2 homem(x) ← pai(x,y)
(se x é pai de y então x é homem)

B3 pai(João,Henrique) ←
(João é pai de Henrique)

O conjunto de cláusulas B1 a B3 expressa todas as assertivas do problema e é consistente. Se for incluída a cláusula B4 ela também será consistente com as cláusulas B1 a B3, apesar de B4 representar uma impossibilidade da natureza.

B4 mãe(João,Henrique) ←
(João é mãe de Henrique)

Para que o fato representado pela cláusula B4 se torne inconsistente com B1 a B3 pode-se, por exemplo, acrescentar a cláusula B5, que elimina a possibilidade de alguém ser pai e mãe ao mesmo tempo.

B5 ← mulher(x), homem(x)

(Se x é mulher então x não é homem)

Um conjunto de cláusulas S é consistente se e somente se todas as suas cláusulas são verdadeiras em uma interpretação de S, caso contrário ele é inconsistente. Uma cláusula é verdadeira em uma interpretação de um conjunto de cláusulas S se e somente se todos os seus literais são verdadeiros. Se houver variáveis na cláusula elas devem ser substituídas com termos do universo de discurso de S antes de se determinar a semântica da cláusula.

A definição de inconsistência permite determinar a semântica da cláusula vazia. Como a cláusula vazia não tem literais, ela não pode ser verdadeira em nenhuma interpretação. Logo a cláusula vazia é sempre falsa.

Para se resolver um problema através de lógica deve-se entender o problema de implicação lógica. Um conjunto de cláusulas S implica uma clásula P, ou P é consequência lógica de S, se o conjunto formado pela união de S com P (S U (P)) é consistente, isto é, se uma interpretação é torna verdadeiras todas as cláusulas de S então ela também torna P verdadeira. Entretanto, será adotado um outro método. Para provar que P é consequência lógica de S nega-se P e mostra-se que o conjunto de cláusulas resultante (S U (¬ P)) é inconsistente. Isso pode não parecer natural mas há razões para isso, que serão apresentadas

adiante, no ítem III.2.5.

Por exemplo, para provar que "João é mortal" deve-se mostrar que A4 é inconsistente com A1 a A3.

A4 ← mortal(João)

De fato, não há nenhuma interpretação que torna o conjunto A1 a A4 consistente, já que se A4 for verdadeira, A2 será falsa e vice-versa.

A inconsistência de um conjunto de cláusulas pode ser demonstrada semanticamente mostrando que nenhuma interpretação do conjunto de cláusulas torna todas verdadeiras, utilizando-se, por exemplo, o método da tabela-verdade; ou então pode ser demonstrada sintaticamente através de uma prova consistindo de passos de inferência. Essa última abordagem é utilizada nos estudos para a mecanização da lógica. Os procedimentos para demonstrar a inconsistência de um conjunto de cláusulas são chamados de procedimentos de refutação. Uma refutação procura deduzir sistematicamente a cláusula vazia, utilizando regras de inferência. Por fim, é importante ressaltar que a semântica da lógica simbólica, baseada na noção de interpretação é independente das regras de inferência utilizadas para manipular expressões da linguagem.

III.2.5 - Conceitos de dedução e refutação.

Dado um conjunto de cláusulas S e uma cláusula C, uma dedução de C a partir de S consiste de uma sequência de cláusulas terminando em C e gerada a partir de S usando regras de

inferência.

A sequência (D1,D2,...,Dn) é uma dedução de C a partir de S, se Dn=C e, para todo i<=n, um dos casos abaixo é satisfeito:

a) Di ocorre em S.
b) Existem j,k<i tais que Di pode ser obtida por alguma regra de inferência a partir de Dj e Dk.

Um exemplo de regra de inferência é a regra Modus Ponens que afirma o seguinte:

MP: a partir de A e A→B deduza B, ou de maneira equivalente, a partir de A e ¬ A B deduza B.

Outro exemplo de regra de inferência é a regra Modus Tollens, que afirma o seguinte:

MT: a partir de ¬ B e A → B deduza ¬ A, ou de maneira equivalente, a partir de ¬ B e ¬ A B deduza ¬ A.

Para descrever um exemplo de aplicação da regra Modus Ponens será utilizado o conjunto de cláusulas S a seguir:

S1. pai(João,Henrique)
S2. pai(João,Catarina)
S3. pai(Henrique,Maria)
S4. mãe(Catarina,José)
S5. ¬ pai(João,Henrique) ¬ pais(Henrique,Maria) avô(João,Maria)
S6. ¬ pai(Henrique,Maria) pais(Henrique,Maria)

S7. ¬ mãe(Catarina,José) pais(Catarina,José)

Para demonstrar que "João é avô de Maria" é verdadeiro deve-se tentar deduzir a cláusula:

C. avô(João,Maria)

A sequência seguinte é uma dedução de C a partir de S:

D1. pai(João,Henrique) de S1

D2. ¬ pai(João,Henrique) ¬ pais(Henrique,Maria) avô(João,Maria) de S5

D3. ¬ pais(Henrique,Maria) avô(João,Maria) de D1 e D2 por MP

D4. pai(Henrique,Maria) de S3

D5. ¬ pai(Henrique,Maria) pais(Henrique,Maria) de S6

D6. pais(Henrique,Maria) de D4 e D5 por MP

D7. avô(João,Maria) de D3 e D6 por MP

Através do método da tabela-verdade pode-se verificar que o conjunto de cláusulas formado por S U {C} é consistente. Portanto C é consequência lógica de S.

Outra maneira de se provar que S implica logicamente C é provar que SA = S U {¬ C} é inconsistente, deduzindo a cláusula vazia (□) a partir de S1. A dedução da cláusula vazia a partir de um conjunto de cláusulas é chamado de refutação. Para o exemplo anterior a dedução da cláusula vazia é imediata:

D8. ¬avô(João,Maria) de ¬ C

D9. □ de D7 e D8 por MP

Um fato importante que deve ser observado é que até aqui não se conhece a ordem em que devem ser escolhidas as sentenças para aplicar uma regra de inferência num processo de dedução. No exemplo que foi analisado, a escolha das cláusulas foi arbitrária, ou seja, elas poderiam ter sido utilizadas em outra ordem. Portanto, a construção de procedimentos sistemáticos de dedução não é simples. Em geral há o problema da explosão combinatória na geração de novas cláusulas, devido à liberdade de escolha de cláusulas para a aplicação de regras de inferência. Procedimentos que procuram gerar sistematicamente a cláusula vazia são denominados de procedimentos de refutação.

Mas por quê esses procedimentos de refutação são importantes? Isso se deve a certas limitações que existem e que estão relacionados com os chamados problemas de decisão para a Lógica de Primeira Ordem. Um problema de decisão consiste de um par PR=(C,P), onde C é um conjunto, cujos elementos são chamados de instâncias de PR, e P é um subconjunto de C, cujos elementos são chamados de instâncias solúveis de PR. Um problema de decisão será decidível se existir um algoritmo que receba como entrada qualquer instância p de PR e pare com SIM se p for uma instância solúvel; e pare com NÃO em caso contrário. Se não existir tal algoritmo, o problema de decisão será indecidível. O problema de decisão será parcialmente decidível se existir um algoritmo que receba como entrada qualquer instância p de PR e pare com SIM se p for uma instância solúvel; e não pare ou pare com NÃO em caso contrário. Para o problema da implicação lógica tem-se que o conjunto de instâncias é um conjunto de fórmulas S U {C} e o conjunto de instâncias solúveis é um conjunto tal que S implica logicamente C. Demonstra-se que o problema da implicação lógica é

indecidível, ou seja, não existe um algoritmo que seja capaz de decidir sempre se uma fórmula C é consequência lógica de um conjunto de fórmulas S. Entretanto, o problema de demonstrar que um conjunto de fórmulas é inconsistente é parcialmente decidível. Isso significa que é possível construir um algoritmo que sempre pára se um conjunto de fórmulas admitir uma refutação e que pode terminar ou não em caso contrário.

Nos ítens seguintes serão analisados alguns procedimentos de refutação baseados em uma regra de inferência denominada de resolução.

III.2.6 - Unificação de cláusulas.

Neste ítem será analisado um mecanismo de substituição de variáveis de cláusulas por termos, denominado de unificação. Esse mecanismo é importante para um método de inferência lógica denominado resolução. Considere-se o seguinte conjunto de cláusulas:

gosta(João,lógica) (João gosta de lógica)

gosta(João,x) ¬ gosta(x,lógica) (João gosta de x se x gosta de lógica)

Se for substituída a variável x da segunda cláusula pelo termo "João" da primeira cláusula pode-se chegar à conclusão de que João gosta de si mesmo. O que foi feito foi tornar "gosta(João,lógica)" e "gosta(x,lógica)" idênticos através da substituição de "x" por "João". O processo de tornar idênticos fórmulas atômicas de um conjunto de cláusulas é chamado de unificação. Para que a unificação de duas fórmulas atômicas seja

possível é necessário que as duas fórmulas tenham a mesma sequência de símbolos, exceto no que se refere às variáveis. Por exemplo "fala(João,inglês)" e "fala(y,inglês)" podem ser unificados (substituição da variável y pela constante João), mas "fala(João,inglês)" e "fala(y,francês)" não podem ser unificados, já que as duas cláusulas apresentam uma estrutura diferente. Pela mesma razão "fala(João,inglês)" e "escreve(y,inglês)" não podem ser unificados.

Então, uma unificação é obtida através da substituição das variáveis por termos (constantes, termos funcionais ou variáveis). Uma substituição é um conjunto do tipo (x1/t1, x2/t2, ..., xn/tn), onde cada elemento xk/tk da substituição instancia um termo tk com uma variável xk.

Uma unificação obedece às seguintes regras:

- Um termo pode substituir várias variáveis.
- Uma variável só pode ser substituída por um único termo.

Como exemplos considere-se o seguinte conjunto de cláusulas

pai(João,Catarina)

mãe(Catarina,José)

avô(x,y) ← pai(x,z) ← mãe(z,y)

Neste caso uma substituição possível é:

(x/João, y/José, z/Catarina)

No capítulo 4 da referência [37] é apresentada uma definição mais precisa do conceito de unificação.

III.2.7 - Resolução.

Resolução é um método de dedução que usa apenas uma regra de inferência, chamada de Regra da Resolução. Uma refutação pelo método da resolução procura gerar a cláusula vazia usando a regra da resolução. Se houver uma refutação a partir de um conjunto de cláusulas S utilizando resolução, então S é inconsistente.

A regra da resolução afirma o seguinte:

RE: se uma cláusula A1 possui um literal L1 e uma cláusula A2 possui um literal ¬L2 e L1 e L2 podem ser unificados então derive A = (A1 - L1) v (A2 - ¬ L2). Neste caso a cláusula A chama-se resolvente de A1 e A2.

Considere-se o seguinte exemplo:

De A1. pai(João,Henrique)

A2. ¬ pai(x,z) ¬ pai(z,y) avô(x,y)

Pode-se derivar

A. ¬ pai(Henrique,y) avô(João,y)

Neste caso ocorreu a unificação de "pai(João,Henrique)" e "¬ pai(x,z)" através da substituição (x/João, z/Henrique). Como esses dois literais tem sinais opostos (um é positivo e outro é negativo), eles não aparecem no resolvente resultante.

Uma refutação a partir de um conjunto de cláusulas S, utilizando resolução, é uma sequência (D1,D2,...,Dn) de cláusulas tal que Dn é a cláusula vazia e, para todo i<=n, um dos casos abaixo é satisfeito:

a) Di ocorre em S.

b) Di é um resolvente de Dj e Dk, para j,k<i.

Por exemplo, seja S o seguinte conjunto de cláusulas:

S1. pai(João,Henrique)

S2. pai(João,Catarina)

S3. pai(Henrique,Maria)

S4. mãe(Catarina,José)

S5. ¬ pai(x,z) ¬ pais(z,y) avô(x,y)

S6. ¬ pai(x,y) pais(x,y)

S7. ¬ mãe(x,y) pais(x,y)

Se o objetivo for demonstrar que "João é avô de José" é verdadeiro, deve-se acrescentar a cláusula:

S8. ¬ avô(João,José)

e gerar a seguinte refutação:

D1. pai(João,Catarina)	de S2
D2. ¬ pai(x,z) ¬ pais(z,y) avô(x,y)	de S5
D3. ¬ pais(Catarina,y) avô(João,y)	de D1 e D2 por RE
D4. mãe(Catarina,José)	de S4
D5. ¬ mãe(x,y) pais(x,y)	de S7
D6. pais(Catarina,José)	de D4 e D5 por RE
D7. avô(João,José)	de D3 e D6 por RE
D8. ¬ avô(João,José)	de S8
D9. □	de D7 e D8 por RE

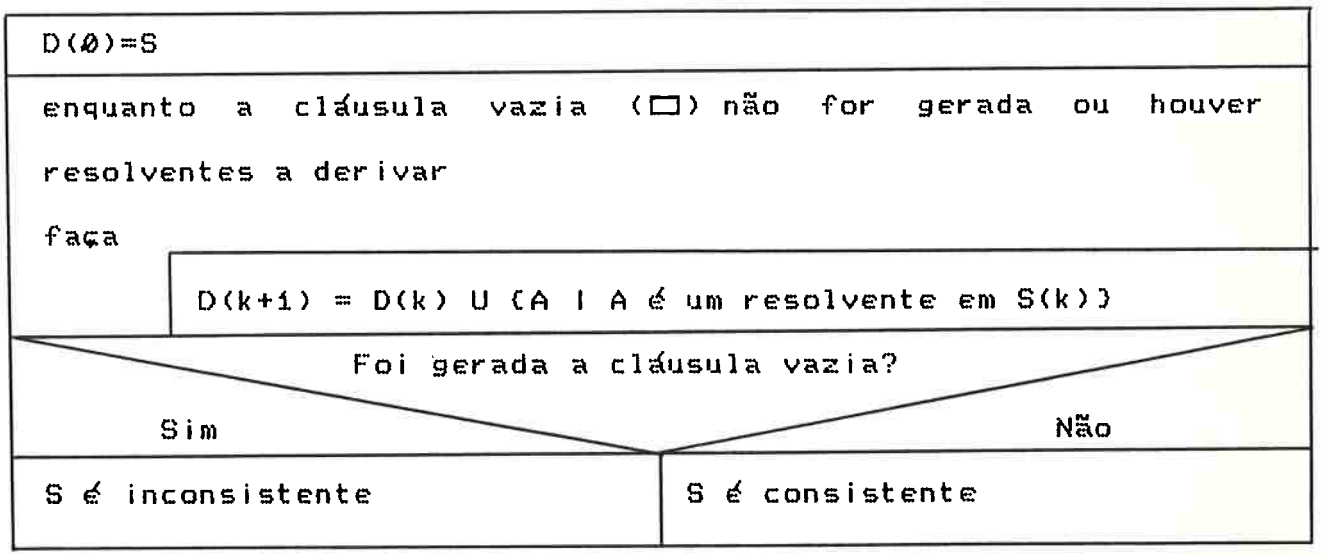
III.2.8 - Procedimentos de refutação por resolução.

Este item descreve alguns procedimentos de refutação por resolução. Serão analisados os procedimentos de Resolução por

Saturação, Resolução Linear e Resolução LSD.

a) Procedimento de Resolução por Saturação.

Seja S um conjunto finito de cláusulas. O procedimento de resolução por saturação é descrito através do diagrama de Nassi-Shneiderman a seguir:

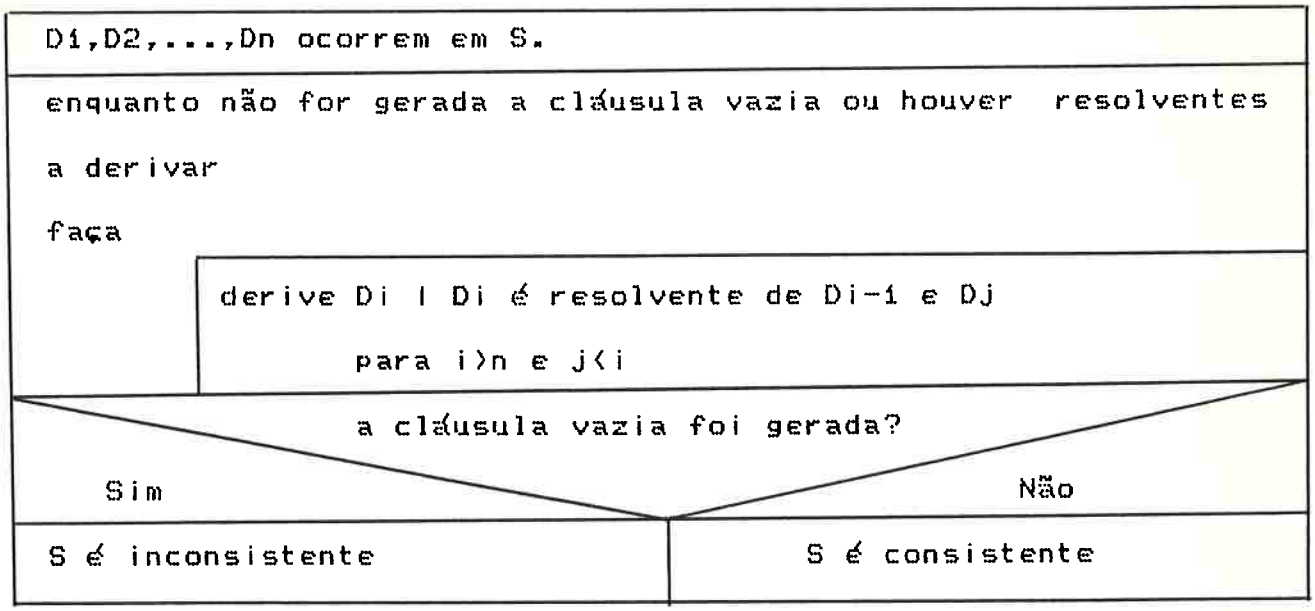


Deve-se observar que se S for consistente e houver um número infinito de resolventes a gerar, o procedimento descrito nunca pára (procedimento de decisão parcial). Este método não tem interesse prático, pois a liberdade de escolha de cláusulas e literais permitida pela regra da resolução pode resultar numa explosão combinatória de resolventes. Isso pode ser evitado através de uma escolha sistemática de cláusulas e literais. Um procedimento mais eficiente é implementado pela resolução linear.

b) Resolução linear.

Seja S um conjunto finito de cláusulas (D1,D2,...,Dn). O procedimento de resolução linear gera uma sequência de cláusulas (D1,D2,...,Dn,Dn+1,...,Di), onde Di é a cláusula vazia, se S for

inconsistente. O procedimento de resolução linear é descrito pelo diagrama a seguir.



Observe-se que este procedimento já limita a escolha de cláusulas para derivar o resolvente. Todo resolvente Di é gerado da cláusula imediatamente anterior Di-1 e de uma cláusula anterior Dj, tal que j<n.

Seja um exemplo de aplicação com o seguinte conjunto de cláusulas:

S1. pai(João,Henrique)

S2. pai(João,Catarina)

S3. pai(Henrique,Maria)

S4. mãe(Catarina,José)

S5. ¬ pai(x,z) ¬ pais(z,y) avô(x,y)

S6. ¬ pai(x,y) pais(x,y)

S7. ¬ mãe(x,y) pais(x,y)

Deseja-se demonstrar que "João é avô de José." Então tem-se que:

S8. ¬ avô(João,José)

A seguinte sequência de cláusulas pode ser gerada pelo procedimento de resolução linear:

D1.	pai(João,Henrique)	de S1
D2.	pai(João,Catarina)	de S2
D3.	pai(Henrique,Maria)	de S3
D4.	mãe(Catarina,José)	de S4
D5.	¬ pai(x,z) ¬ pais(z,y) avô(x,y)	de S5
D6.	¬ pai(x,y) pais(x,y)	de S6
D7.	¬ mãe(x,y) pais(x,y)	de S7
D8.	¬ avô(João,José)	de S8
D9.	¬ pai(João,z) ¬ pais(z,José)	de D8 e D5
D10.	¬ pais(Catarina,José)	de D9 e D2
D11.	¬ mãe(Catarina,José)	de D10 e D7
D12.	□	de D11 e D4

c) Resolução LSD.

A resolução-LSD trabalha com cláusulas de Horn e utiliza um procedimento de resolução linear com função de seleção f. Uma função de seleção é uma função que seleciona um literal de uma cláusula-objetivo. Por exemplo, seja C uma cláusula-objetivo da seguinte forma:

← N1,N2,...,Nm

Então f(← N1,N2,...,Nm) = Ni. Se f mapeia a cláusula-objetivo iniciando pelo literal mais à esquerda então f é denominada função de seleção padrão.

A regra da resolução com função de seleção f pode ser

descrita da maneira a seguir.

Seja uma cláusula de Horn da forma

L←M1,M2,...,Mn

Seja uma cláusula-objetivo da forma

←N1,...,Ni,...,Nm

Se f(←N1,...,Nm) = Ni e L e Ni podem ser unificados, derive a cláusula

←N1,...,Ni-1,M1,...,Mn,Ni+1,...,Nm

Considere-se um exemplo com o seguinte conjunto de cláusulas:

S1. pai(João,Henrique) ←

S2. pai(João,Catarina) ←

S3. pai(Henrique,Maria) ←

S4. mãe(Catarina,José) ←

S5. avô(x,y) ← pai(x,z), pais(z,y)

S6. pais(x,y) ← pai(x,y)

S7. pais(x,y) ← mãe(x,y)

Para demonstrar que "João é avô de José" deve-se acrescentar a seguinte cláusula de Horn:

S8. ← avô(João,José)

A seguinte sequência de cláusulas é uma refutação utilizando-se resolução LSD com função de seleção padrão.

D1. pai(João,Henrique) ← de S1

D2. pai(João,Catarina) ← de S2

D3.	pai(Henrique,Maria) ←	de S3
D4.	mãe(Catarina,José) ←	de S4
D5.	avô(x,y) ← pai(x,z), pais(z,y)	de S5
D6.	pais(x,y) ← pai(x,y)	de S6
D7.	pais(x,y) ← mãe(x,y)	de S7
D8.	← avô(João,José)	de S8
D9.	← pai(João,z), pais(z,José)	de D8 e D5
D10.	← pais(Catarina,José)	de D9 e D2
D11.	← mãe(Catarina,José)	de D10 e D7
D12.	□	de D11 e D4

Se f selecionar o literal mais à direita ao invés do literal mais à esquerda tem-se a seguinte refutação:

D1.	pai(João,Henrique) ←	de S1
D2.	pai(João,Catarina) ←	de S2
D3.	pai(Henrique,Maria) ←	de S3
D4.	mãe(Catarina,José) ←	de S4
D5.	avô(x,y) ← pai(x,z), pais(z,y)	de S5
D6.	pais(x,y) ← pai(x,y)	de S6
D7.	pais(x,y) ← mãe(x,y)	de S7
D8.	← avô(João,José)	de S8
D9.	← pai(João,z), pais(z,José)	de D8 e D5
D10.	← pai(João,z), mãe(z,José)	de D9 e D7
D11.	← pai(João,Catarina)	de D10 e D4
D12.	□	de D11 e D2

Aqui cabem alguns comentários. Pode-se observar que nos exemplos anteriores foi demonstrado o fato "João é avô de José" através de diferentes procedimentos. Isso exemplifica que a lógica dessas sentenças é independente dos mecanismos de

inferência utilizados para se gerar uma conclusão, indo em encontro com as idéias de separação de um algoritmo em dois componentes (lógico e de controle). Os procedimentos descritos são corretos e completos. São corretos no sentido em que se a cláusula vazia é gerada a partir de S U ($\neg$ C), então C é consequência lógica de S. São completos no sentido em que se C é consequência lógica de S então a cláusula vazia será gerada. Finalmente, deve-se lembrar que esses procedimentos serão limitados pelo fato dos procedimentos de refutação serem procedimentos de decisão parcial. Isso significa que se um conjunto de cláusulas não admitir uma refutação, então a execução do procedimento poderá não terminar nunca, numa tentativa inútil de gerar a cláusula vazia.

III.3 - Programação em Lógica.

Nos itens III.1 e III.2 foram analisados alguns conceitos de lógica que formam a base para a compreensão dos Sistemas de Programação em Lógica. Neste trabalho será considerado a variante da Programação em Lógica em que os programas são conjuntos finitos de cláusulas de Horn.

Um programa em cláusulas de Horn consiste de um conjunto de regras e assertivas. Por exemplo, considere-se o seguinte programa:

pai(João,Henrique) ←

pai(João,Catarina) ←

pai(Henrique,Maria) ←

mãe(Catarina,José) ←

avô(x,y) ← pai(x,z), pais(z,y)

pais(x,y) ← pai(x,y)

pais(x,y) ← mãe(x,y)

Cada assertiva expressa um relacionamento atômico entre indivíduos. Exemplos de assertivas são as três primeiras sentenças do programa apresentado. Cada regra tem a forma "A se B e C e...e D", onde A é a conclusão e B, C, ..., D são condições a serem satisfeitas para que A seja verdadeira. As três últimas sentenças são regras e X, Y e Z representam variáveis. Neste caso, variáveis correspondem a indivíduos arbitrários. As regras e assertivas são formadas por cláusulas de Horn.

Além da noção de programa, tem-se os conceitos de consulta e de resposta de um programa. O objetivo de um programa

em lógica é tentar provar que a consulta desejada é consequência lógica do programa. Logo, uma consulta é representada por uma cláusula-objetivo. As noções de consulta e resposta representam algumas das principais características da Programação em Lógica.

As consultas mais elementares são aquelas que não contém variáveis livres e podem ser interpretadas da seguinte maneira: dada uma sentença S, testar se S é consequência lógica do conjunto de sentenças contidas no programa. Tais consultas tem apenas uma resposta correta, que é representada pela cláusula vazia, ou não tem respostas corretas. Por exemplo, considere-se as consultas

← pai(João,Henrique) e

← avô(João,Maria)

A resposta dessas consultas é a claúsula vazia, já que elas são implicadas logicamente pelo programa e não tem variáveis. As consultas com variáveis podem ter respostas com uma ou várias substituições. Uma resposta com apenas uma substituição representa uma inferência simples a partir do programa. Por exemplo, seja a consulta:

← avô(x,Maria)

A resposta é a substituição (x/João). Quando há várias substituições, elas representam respostas simples alternativas para a consulta em questão. A consulta "← pai(x,y)" apresenta uma resposta correta que contém as substituições ((x/João, y/Henrique), (x/João, y/Catarina), (x/Henrique, y/Maria)). As substituições de variáveis por termos do programa resultam dos processos de unificação. Já as consultas "← pai(Henrique,João)" e "← filho(João,Catarina)" não tem respostas corretas.

Um programa em lógica é executado quando submete-se uma consulta como entrada de um interpretador lógico. O interpretador é um programa capaz de realizar inferência por Resolução de acordo com o procedimento de refutação implementado. Se a cláusula vazia for gerada, o interpretador apresenta algum tipo de confirmação de que ela foi gerada, juntamente com as substituições das variáveis, se houver, da cláusula-objetivo que representa a consulta. Se a cláusula vazia não for gerada o interpretador também deve emitir algum tipo de mensagem informando que ela não foi gerada. Por exemplo, seja a seguinte consulta:

← avô(x,y)

O interpretador deverá indicar que há uma resposta correta com as seguintes substituições ((x/João, y/Maria), (x/João, y/José)).

As regras de um programa em lógica possuem uma interpretação procedimental. De fato, regras da forma

A ← B,C,...,D

podem ser interpretadas como procedimentos que reduzem o problema de reduzir A aos subproblemas de resolver B,C,...,D. Assim A é o nome do procedimento A (- B,C,...,D, que por sua vez contém chamadas para os procedimentos B,C,..., e D.

III.4 - A linguagem PROLOG.

A linguagem PROLOG ("PROgramming in LOGic") é uma implementação de alguns conceitos da Programação em Lógica. O PROLOG é baseado nas cláusulas de Horn e opera com um procedimento de refutação baseado em Resolução.

Em linguagens convencionais, um programa é uma coleção de procedimentos que são executados em ordem específica. Um programa PROLOG é uma coleção de fatos e regras, que são representadas através de cláusulas de Horn. A execução de um programa PROLOG começa a partir de uma consulta, representada por uma cláusula-objetivo, que é entregue a uma máquina de inferência. Pode-se denominar essa máquina de máquina PROLOG. Uma máquina PROLOG pode trabalhar, por exemplo, com um procedimento de resolução tal que as cláusulas do programa são selecionadas na ordem em que ocorrem e o literal selecionado de cada cláusula é sempre o mais à esquerda. Normalmente, os interpretadores ou compiladores PROLOG seguem estas duas estratégias de seleção de cláusulas e literais.

A sintaxe das cláusulas PROLOG varia conforme a sua implementação. Por exemplo, em algumas versões do PROLOG, o símbolo "←" é substituído por ":-", a conjunção é representada por vírgula (",") e todas as cláusulas terminam com ponto (".").

A linguagem PROLOG pode suportar vários tipos de dados, tais como variáveis, constantes e estruturas.

Variáveis são objetos que podem ser ligadas a quaisquer valores, ou mesmo, outras variáveis, nos processos de unificação.

O primeiro caracter do nome de uma variável é uma letra maiúscula. O símbolo "_" ("undescore") representa a variável anônima. Variáveis anônimas são variáveis que são sempre unificadas com sucesso, mas não são ligadas a nenhum valor. Exemplos de variáveis são:

X, Y, Pai, Var1

Constantes podem ser átomos ou números. Átomos são sequências de caracteres que representam objetos simples. O primeiro caracter de um átomo é uma letra minúscula, ou então os caracteres estão entre apóstrofes. Os números podem ter notação inteira e ponto flutuante, conforme a implementação da linguagem. Exemplos de constantes são:

joão, c5, "Catarina", 123, 999.99

Estruturas permitem manipular objetos compostos como se fossem objetos simples. Um tipo de estrutura é denominada de lista. Uma lista é formada por duas partes, denominadas de cabeça e cauda, sendo que esta última pode ser uma outra lista. As estruturas seguintes são exemplos de listas:

["João","Henrique","Catarina","Maria","José"]

["João"|Cauda]

[], que representa a lista vazia.

O PROLOG responde a consultas feitas executando unificação. O processo de unificação seguem os mesmos princípios já abordados no ítem III.2.6. Descrevendo esse processo resumidamente, uma cláusula é selecionada para ser unificada com

uma cláusula-objetivo se elas tem o mesmo símbolo predicativo e o mesmo número de argumentos. Variáveis unificam com constantes, estruturas ou outras variáveis, constantes unificam com constantes que sejam iguais, e estruturas unificam com outras estruturas desde que elas tenham o mesmo símbolo funcional, o mesmo número de argumentos e estes possam ser unificados.

Um processamento de um programa em PROLOG inicia-se com uma consulta. A máquina PROLOG vai consultando as cláusulas do programa na ordem em que ocorrem e seleciona a primeira cláusula que puder ser unificada com a cláusula-objetivo que representa a consulta. Se a cláusula selecionada possuir apenas a parte direita, denominada de "conclusão" da cláusula, e a unificação for realizada com sucesso, então a máquina PROLOG apresenta uma mensagem de SUCESSO da consulta e mostra os valores que foram ligados às variáveis que eventualmente existirem na consulta. Se a cláusula selecionada possuir a parte esquerda, denominada de "condições" da cláusula, então, cada literal da parte esquerda deve ser unificada com outra cláusula do programa. Quando todos os literais forem unificados, a máquina PROLOG apresentará uma mensagem de SUCESSO, juntamente com os valores que foram ligados às variáveis. Caso não houver sucesso em realizar uma unificação e não houver mais cláusulas para selecionar, então a máquina PROLOG apresentará uma mensagem de FALHA.

Quando ocorre uma falha na unificação, a máquina PROLOG retorna para o ponto onde se iniciou a seleção da cláusula que unificaria com a cláusula-objetivo corrente e procura a próxima alternativa. Esse processo é denominado de "backtracking".

Para ilustrar o funcionamento da máquina PROLOG tem-se o

seguinte programa exemplo, desenvolvido para o Turbo PROLOG:

```
pai("Joao","Henrique").
pai("Joao","Catarina").
pai("Henrique","Maria").
mae("Catarina","Jose").

avo(X,Y):- pai(X,Z), pais(Z,Y).
pais(X,Y):- pai(X,Y).
pais(X,Y):- mae(X,Y).
```

Se for feita a consulta "pai("Henrique",X)" o programa irá executar os seguintes passos:

```
Goal: pai("Henrique",X)
REDO:pai("Henrique",_)
REDO:pai("Henrique",_)
RETURN:pai("Henrique","Maria")
X=Maria
1 Solution
```

Para a consulta "pai(X,"Maria") tem-se os seguintes passos de execução:

```
Goal: pai(X,"Maria")
REDO:pai(_,"Maria")
REDO:pai(_,"Maria")
RETURN:pai("Henrique","Maria")
X=Henrique
1 Solution
```

Já a pesquisa para a consulta "avo("Joao",Y) é mais complexa:

```
Goal: avo("Joao",Y)
CALL:pai("Joao",_)
RETURN:*pai("Joao","Henrique")
CALL:pais("Henrique",_)
CALL:pai("Henrique",_)
REDO:pai("Henrique",_)
REDO:pai("Henrique",_)
RETURN:pai("Henrique","Maria")
RETURN:*pais("Henrique","Maria")
RETURN:avo("Joao","Maria")
Y=Maria
REDO:pais("Henrique",_)
CALL:mae("Henrique",_)
FAIL:mae("Henrique",_)
REDO:pai("Joao",_)
RETURN:pai("Joao","Catarina")
CALL:pais("Catarina",_)
CALL:pai("Catarina",_)
REDO:pai("Catarina",_)
REDO:pai("Catarina",_)
FAIL:pai("Catarina",_)
REDO:pais("Catarina",_)
CALL:mae("Catarina",_)
RETURN:mae("Catarina","Jose")
RETURN:pais("Catarina","Jose")
RETURN:avo("Joao","Jose")
Y=Jose
2 Solutions
```

Nesta última consulta percebe-se que a máquina PROLOG encontrou uma solução para o problema (Y=Maria) e depois apresentou uma solução alternativa (Y=Jose). Essa é uma das características da Programação em Lógica: a possibilidade de apresentar soluções alternativas a um problema dado.

Além disso, a linguagem PROLOG permite a definição de programas invertíveis, isto é, que não distinguem entre argumentos de entrada e saída (isso foi ilustrado acima pelas consultas "pai("Henrique",X)" e "pai(X,"Maria"), e permite a representação de programas e dados através do mesmo formalismo (cláusulas).

Entre as dificuldades atuais apresentadas pela linguagem pode-se citar a existência de vários dialetos e baixa performance quando executada em computadores convencionais.

Diversas técnicas tem sido propostas para aumentar a performance do PROLOG. Pode-se citar entre elas os mecanismos de "AND-parallelism" e "OR-parallelism".

O mecanismo de "AND-parallelism" é a solução simultânea de mais de um literal no corpo de uma cláusula. Por exemplo, na cláusula "avo(X,Y):- pai(X,Z), pais(Z,Y)" a unificação simultânea de "pai(X,Z)" e "pais(Z,Y)" poderia ser um exemplo de "AND-parallelism". A dificuldade nessa abordagem é manter a consistência do valor que será ligado à variável Z nos processos de unificação das duas cláusulas.

O mecanismo "OR-parallelism" representa a unificação simultânea de múltiplas cláusulas com a cláusula-objetivo

corrente. Por exemplo, a cláusula-objetivo "pai(X,Y)" poderia ser unificada simultaneamente com as cláusulas:

pai("Joao","Henrique"),
pai("João","Catarina") e
pai("Henrique","Maria").

Para implementar esses mecanismos de paralelismo, diversas máquinas, denominadas máquinas PROLOG, tem sido projetadas e analisadas. Pode-se citar entre elas o "Parallel Inference Machine (PIM)", desenvolvimento realizado pelo projeto do computador de 5a. geração (ICOT-Japão), o "Parallel Inference Engine (PIE)", desenvolvido pela Universidade de Tóquio (Tóqui-Japão), e o "Programmed Logic Machine (PLM)", construído pela Universidade da Califórnia (Berkeley-USA).

Notas bibliográficas: o texto deste capítulo é baseado nas referências [1], [2], [4], [11], [12], [13], [15], [36], [37], [40] e [41]. A parte referente à Lógica foi baseada nas referências [1], [15] e [37]. Em [37] tem-se uma apresentação formal da Lógica, ao contrário de [1], que é mais informal. A Programação em Lógica é abordada nas referências [1], [15], [36] e [37]. As referências [2], [4], [11], [12] e [13] tratam da linguagem PROLOG, em particular [4] e [11] apresentam bons exemplos para quem deseja iniciar-se no PROLOG. A referência [40] trata de arquiteturas de computadores para processamento de Inteligência Artificial e em [41] tem-se uma apresentação do projeto japonês do computador de 5a. geração.

IV - Aplicações de Métodos de Inteligência Artificial e de Sistemas Especialistas aos Problemas de Sistemas de Potência.

Neste item serão analisados alguns métodos de Inteligência Artificial e Sistemas Especialistas aplicados aos problemas de Sistemas de Potência. Mas antes de ser feita essa análise, será apresentada uma descrição funcional de um Sistema Elétrico de Potência e de seus problemas relacionados. Também serão descritos os sistemas SCADA e EMS, que tem a função de efetuar a Supervisão e Controle do Sistema Elétrico, e que são candidatos naturais a receberem Sistemas Especialistas e técnicas de Inteligência Artificial.

IV.1 - O Sistema Elétrico de Potência.

Os Sistemas Elétricos de Potência tem a função de fornecer energia elétrica aos usuários, no instante em que esta for solicitada. Então, cada Sistema de Potência é um produtor que transforma energia hidráulica, térmica ou de outra natureza em energia elétrica, e um distribuidor que fornece a quantidade de energia solicitada, instante a instante. Na figura IV.1.1 pode-se observar um esquema geral de um Sistema de Potência.

É interessante ressaltar alguns aspectos dos Sistemas de Potência. Não é possível o armazenamento da energia produzida, sendo a produção vinculada à demanda, instante a instante. A produção em geral é distante dos centros de consumo e devido à grande distância de transmissão utilizam-se tensões elevadas para diminuir as perdas de energia. Os sistemas de transmissão entregam a energia a estações distribuidoras, que tem a função

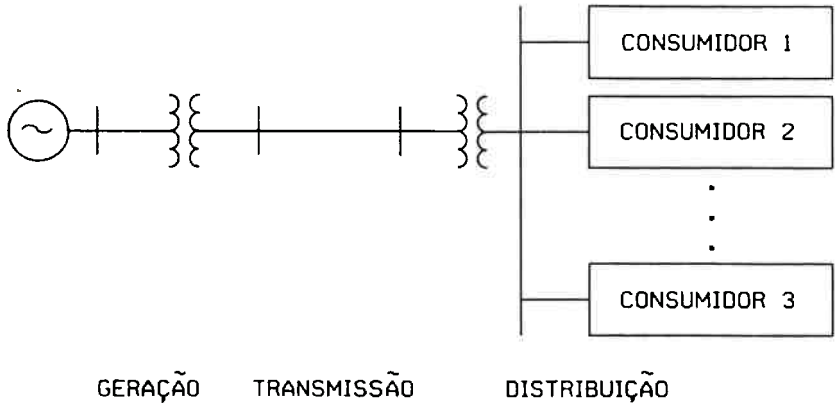


FIGURA IV.1.1 - ESQUEMA GERAL DO SISTEMA DE POTÊNCIA

de entregá-la aos consumidores em níveis de tensão mais adequados.

Um fator importante reside na confiabilidade dos Sistemas de Potência, isto é, o índice de disponibilidade deve ser alto. A confiabilidade de um Sistema de Potência depende de um planejamento, tanto de projeto como de operação e expansão. Os estudos de planejamento exigem estudos de fluxo de carga, cálculo de faltas e estudos de estabilidade, entre outros.

Os estudos de fluxo de carga tem a função de determinar a tensão, a corrente, a potência e o fator de potência nos diversos pontos do sistema elétrico. Estes dados são essenciais para o conhecimento prévio dos efeitos da interligação com outros sistemas, da ligação de novas cargas, de novas usinas e de linhas de transmissão.

O cálculo de faltas tem como objetivo determinar a corrente que circula nas diferentes partes do sistema elétrico durante a ocorrência de uma falta. A falta em um fio ou um cabo pode ser definida como uma falha total ou parcial na sua isolação ou continuidade. Os dados obtidos do cálculo de faltas servem para especificar os disjuntores, que deverão interromper a corrente e isolar a falta, e também para ajustar os relés que controlam os disjuntores.

Os estudos de estabilidade, por sua vez, procuram determinar as condições de estabilidade de um sistema elétrico. A estabilidade de um sistema elétrico consiste em manter o funcionamento síncrono dos geradores e motores do sistema. Quando varia a carga de um gerador ou de um sistema, ocorre uma variação na corrente, do gerador ou do sistema. Se a variação da corrente não resultar em variação dos módulos das tensões dos geradores, forçosamente as fases das tensões deverão variar. Então, variações momentâneas de velocidade são necessárias para o ajuste das fases das tensões, uma vez que essas fases são determinadas pelas posições relativas dos rotores. A instabilidade ocorre quando se pretende aumentar a energia mecânica fornecida a um gerador acima de um limite definido, chamado limite de estabilidade.

A interligação dos Sistemas de Potência oferece um aumento da confiabilidade, pois aumenta a oferta de potência, além de proporcionar vantagens econômicas. Pode-se citar, entre elas, a exigência de um menor número de máquinas reservas destinadas a operar em condições de pico (capacidade de reserva) e de um menor número de máquinas operando em vazio para atender repentinos e inesperados aumentos de consumo (reserva

operante). Entretanto, a interligação dos Sistemas de Potência traz alguns problemas. Por exemplo, a corrente de curto-circuito é aumentada, obrigando a instalação de disjuntores de maior capacidade, e as perturbações causadas por um curto-circuito podem se extender aos sistemas interligados, a menos que sejam instalados nos pontos de interligação relés e disjuntores adequados.

É importante lembrar que os sistemas devem operar na mesma frequência e fase.

IV.2 - Sistemas SCADA

A complexidade dos Sistemas de Potência exige a instalação de Sistemas de Supervisão e Controle para a operação e garantia da confiabilidade.

"Controle e Supervisão" pode ser definido como a capacidade de exercer controle sobre o sistema e verificar a sua performance de acordo com a ação desejada.

Os Sistemas de Supervisão e Controle são conhecidos como sistemas SCADA ("Supervisory Control And Data Acquisition"). Os sistemas SCADA permitem aos operadores executar suas tarefas com informação e controle suficiente, de maneira cuidadosa, segura e econômica. Os operadores têm as funções de comandar a operação normal do sistema, verificar os problemas que ocorrerem, responder de maneira rápida e efetiva aos mesmos, além de gerar e emitir relatórios. Como outros usuários do sistema SCADA pode-se citar os departamentos de proteção, de manutenção e de produção. O departamento de proteção utiliza as informações do

sistema SCADA para determinar se os relés operaram adequadamente, no caso de uma falta, e para determinar porque eles não operaram, em caso de falha. O departamento de manutenção faz uso das informações para programar a manutenção de disjuntores em função do número de operações desses dispositivos. O departamento de produção utiliza o SCADA, para coletar dados sobre a geração e o consumo.

Um sistema SCADA consiste genericamente de uma estação principal e de um certo número de UTRs (Unidades Terminais Remotas) dispersos geograficamente, todos interconectados à estação principal através de canais de comunicação. Na figura IV.2.1 pode-se observar esquemas genéricos de sistemas SCADA, onde as UTRs são ligadas de maneiras diferentes à estação principal. As UTRs são instaladas em locais de geração e subestações, para permitir à estação principal obter dados e executar comandos de controle. Cada UTR possui interface com os equipamentos de controle, através de relés de comando, e com circuitos de medição, através de transdutores. A estação principal deve fornecer funções relacionadas com a operação, monitoração e parada das plantas de potência, com a operação do sistema de transmissão, com a execução da secionalização da rede elétrica e operações de chaveamento. Para isso deve dispor de bases de dados com várias informações, como por exemplo: tensões de barramento, fluxos de linhas (MW, MVAr, A), fluxos de transformador (MW, MVAr), posições de "tap" de transformadores, estados de chaves e disjuntores, estados de relés, alarme de subestações, alarmes de transformadores (por exemplo: pressão), leituras de MW e MVAr-hora, estado do esquema de proteção e informações de sequências de eventos.

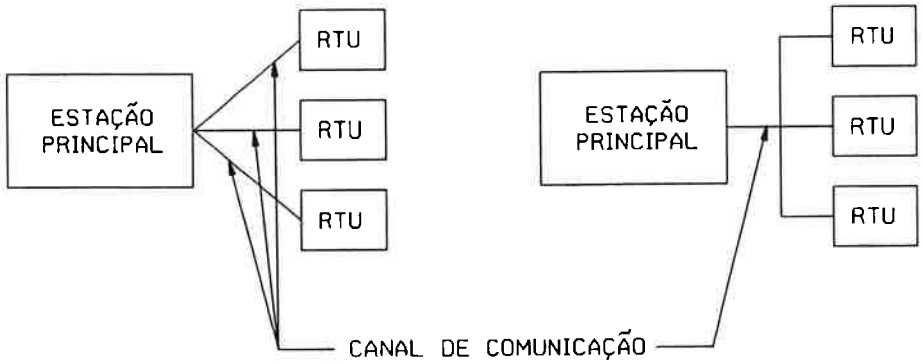


FIGURA IV.2.1 - ESQUEMAS GENÉRICOS DE SISTEMAS SCADA

IV.3 - Sistemas de Gerenciamento de Energia (EMS).

Sistemas de Gerenciamento de Energia ("EMS, Energy Management System") são sistemas de tempo real baseados em "mainframes" ou sistemas distribuídos de computadores para executar funções computacionalmente intensivas, relacionadas com o sistema elétrico.

Os sistemas EMS executam programas de aplicação, são responsáveis por uma interface homem-máquina sofisticada e interagem com o sistema elétrico através de sistemas SCADA.

Na figura IV.3.1 pode-se observar um diagrama de blocos do EMS. Pode-se citar como programas aplicativos os que fazem o controle automático em tempo real (por exemplo controle automático de geração), monitoração da rede em tempo real, e a

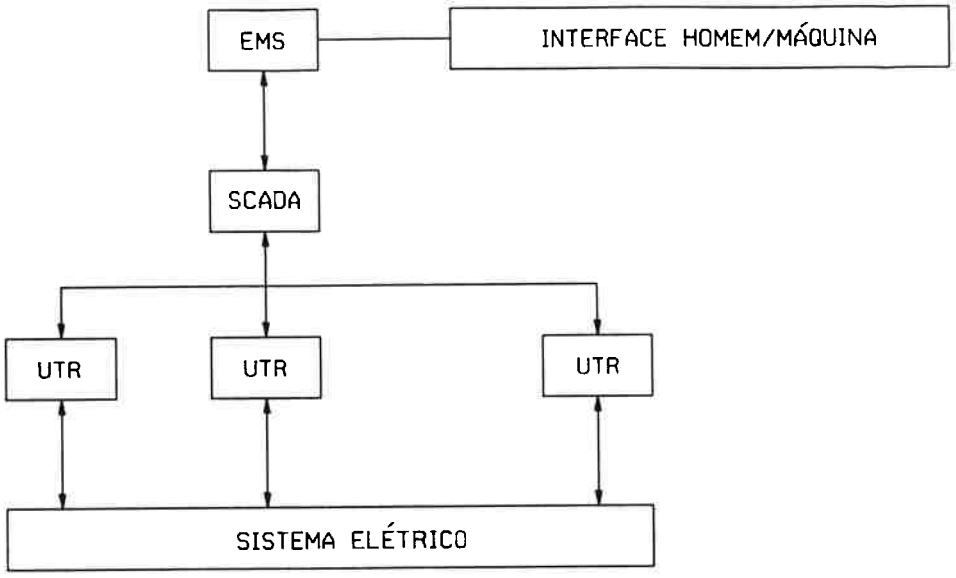


FIGURA IV.3.1 - DIAGRAMA DO EMS

análise de segurança "on-line" (representada por aplicativos que determinam o fluxo de carga, seleção de contingências e alocação de unidades geradoras). Os sistemas de gerenciamento de energia também oferecem interfaces homem-máquina mais elaboradas que as tradicionais. As primeiras interfaces homem-máquina eram compostas por conjuntos de teclas, displays numéricos e registradores gráficos. Com o tempo foram surgindo terminais monocromáticos e coloridos, alfanuméricos e gráficos, que podem mostrar rapidamente dados de maneira dinâmica e atualizados.

Sistemas EMS podem ter milhares de pontos de medição sendo monitorados. Naturalmente é difícil gerenciar de maneira adequada todos esses dados. Assim tornou-se necessário o desenvolvimento de sistemas de gerenciamento de bases de dados e novas estratégias no projeto de interfaces homem-máquina. O objetivo é fornecer aos operadores uma menor quantidade de dados e dados mais inteligentes, livrar os operadores da tarefa mais

árdua permitindo-se assim tempo para pensar, além de fazê-los se preocuparem com o sistema elétrico e não com o sistema de controle.

O desenvolvimento futuro de sistemas EMS pode ser dividido em diversas áreas como Base de Dados Relacionais, Interfaces Gráficas, Reconhecimento de Padrões, Inteligência Artificial e Redes de Comunicação.

A introdução de gerenciadores de bases de dados relacionais irá permitir que os dados sejam armazenados em grupos funcionais chamados relações. Essas relações podem acessar qualquer subconjunto de dados e fornecê-los à aplicação requisitante em um formato conveniente. Dessa maneira, a interface homem-máquina, a aquisição de dados e as várias aplicações podem ser verdadeiramente independentes do formato da base de dados e podem efetivamente compartilhar os mesmos dados. O uso de bases relacionais implicará em tarefas computacionalmente mais pesadas, mas a performance e preço de novos computadores deverá eliminar esse problema.

Em relação às Interfaces Gráficas o custo e a performance do hardware gráfico tornam possível emular as características de apresentação de medidores analógicos e registradores gráficos. Os terminais gráficos coloridos podem apresentar desenhos de medidores analógicos, histogramas, termômetros e gráficos em tempos de resposta admissíveis. Outras aplicações também são possíveis: mover uma janela através de um diagrama do Sistema de Potência, mostrando a configuração do sistema e das cargas, e "zoom" para variar a magnitude de visão da janela de acordo com a necessidade.

Os estudos relacionados com Reconhecimento de Padrões e Inteligência Artificial destinam-se a desenvolver aplicações de diagnóstico e análise. O volume de dados necessários para determinar de maneira precisa o estado do Sistema de Potência é muito grande e os sistemas atuais simplesmente apresentam esses dados aos operadores. Técnicas tem sido desenvolvidas para reconhecer e diagnosticar eventos do sistema, tornando automáticas uma parte ou a totalidade das tarefas de supervisão e controle.

Os sistemas EMS utilizam vários computadores, que devem trocar dados entre si através de redes de comunicação. Esses dados devem ser confiáveis e, portanto, um grande esforço tem sido feito em aplicar tecnologias emergentes de comunicação de dados. Tornou-se vital estabelecer padrões e protocolos de comunicação de dados de modo que um computador ligado a uma rede possa se comunicar com outro com grande confiabilidade. Um dos padrões desenvolvidos, por exemplo, é o modelo OSI ("Open System Interconnection") da ISO ("International Standards Organization").

IV.4 - Análise de algumas Aplicações de Técnicas de Inteligência Artificial e de Sistemas Especialistas aos Problemas de Sistemas de Potência.

A análise e diagnóstico dos problemas de Sistemas de Potência é importante tanto para planejamento como para monitoração e controle.

As tarefas de análise são hoje executadas com auxílio de computadores, que permitem maior rapidez e sofisticação nos estudos de sistemas.

O desenvolvimento do hardware resulta em máquinas mais velozes e de maior capacidade de armazenamento de dados.

O desenvolvimento das técnicas de software permite o surgimento de aplicações cada vez mais sofisticadas.

Estes dois fatos geraram reflexos na área de Sistemas de Potência que passou a dispor de ferramentas mais poderosas. Exemplos de ferramentas que tornaram-se disponíveis são as de aplicações de métodos de Inteligência Artificial, representadas principalmente pelos Sistemas Especialistas. As aplicações de Inteligência Artificial em Sistemas de Potência podem ser divididas em duas áreas: problemas de tempo real e de planejamento.

Os problemas de tempo real são relacionados com tarefas de processamento de alarmes, operações de chaveamento, monitoração e controle da rede elétrica, e restauração de sistema elétrico.

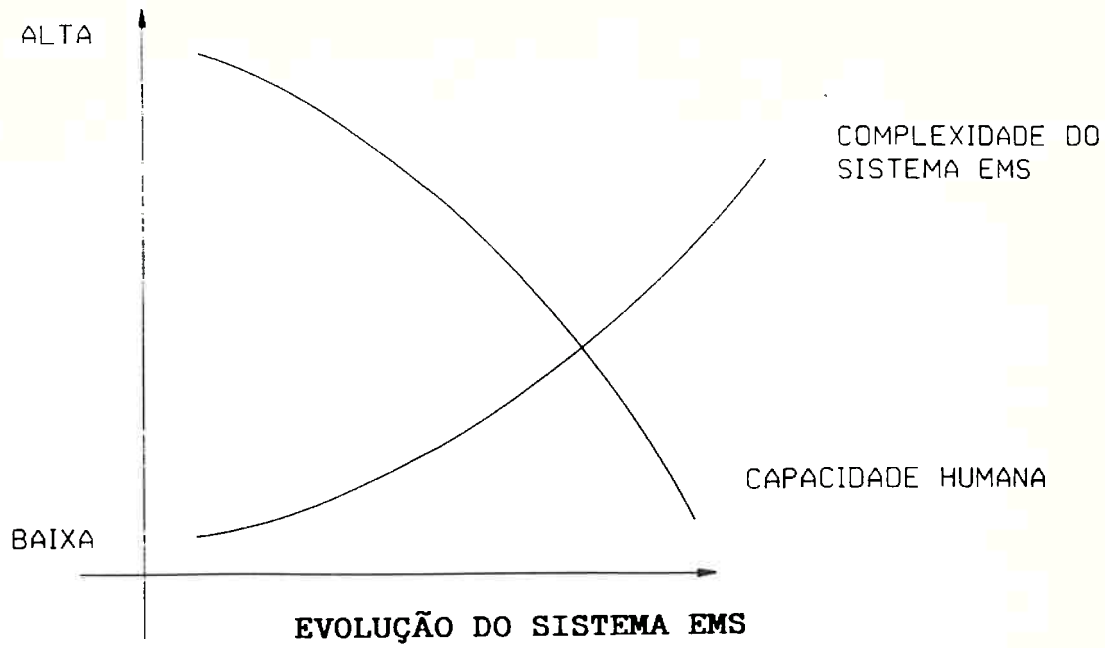


FIGURA IV.4.1 - BARREIRA COGNITIVA

Os problemas de planejamento referem-se ao planejamento do fluxo de potência, "unit commitment", e treinamento de operadores.

As aplicações de tempo real são hoje executadas pelos Sistemas de Gerenciamento de Energia (EMS). Sistemas típicos tem milhares de pontos de medição e por consequência, uma grande quantidade de dados para serem tratados e analisados. Surge, então, o problema da barreira cognitiva, que pode ser visualizada na figura IV.4.1. À medida que se aumenta a complexidade do EMS, diminui-se a capacidade do operador absorver toda a informação disponível num tempo hábil. A solução é automatizar cada vez mais as funções de monitoração e controle, com técnicas confiáveis e inteligentes.

Os Sistemas Especialistas surgem como um método novo para a automação de funções que hoje são de responsabilidade dos operadores.

A introdução de Sistemas Especialistas nos EMS procurar dar suporte ao operador em seu trabalho diário e livrá-lo de atividades rotineiras, além fornecer a ele funções de auxílio inteligentes em situações de emergência.

Ao serem adicionados aos Sistemas de Gerenciamento de Energia, os Sistemas Especialistas devem atender alguns requisitos. Por exemplo, devem ser integrados ao ambiente de hardware e software do centro de controle com capacidade para manipular dados de tempo real do processo. A comunicação entre o operador e o Sistema Especialista seria feita pelos mesmos consoles utilizados para o controle. Além disso poderiam usar a mesma base de dados e a mesma descrição da topologia da rede disponível no centro de controle. As funções existentes no EMS como fluxo de carga ou despacho econômico poderiam ser usados como subsistemas para produzir resultados a serem usados durante inferências realizadas pelo Sistema Especialista.

A implementação de Inteligência Artificial em EMS pode ser feita de várias maneiras. Poder-se-ía utilizar linguagens algoritmicas como C e Pascal (figura IV.4.2). Um inconveniente é a maior dificuldade de se implementar métodos de Inteligência Artificial com linguagens algorítmicas. Outra possibilidade seria utilizar programas escritos em PROLOG ou LISP no ambiente padrão EMS (figura IV.4.3). Entretanto o hardware tradicional frequentemente não é eficiente quando se executa programas escritos nessas linguagens. Isso poderia ser resolvido

FIGURA IV.4.2 - IMPLEMENTAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM EMS UTILIZANDO LINGUAGENS ALGORÍTMICAS

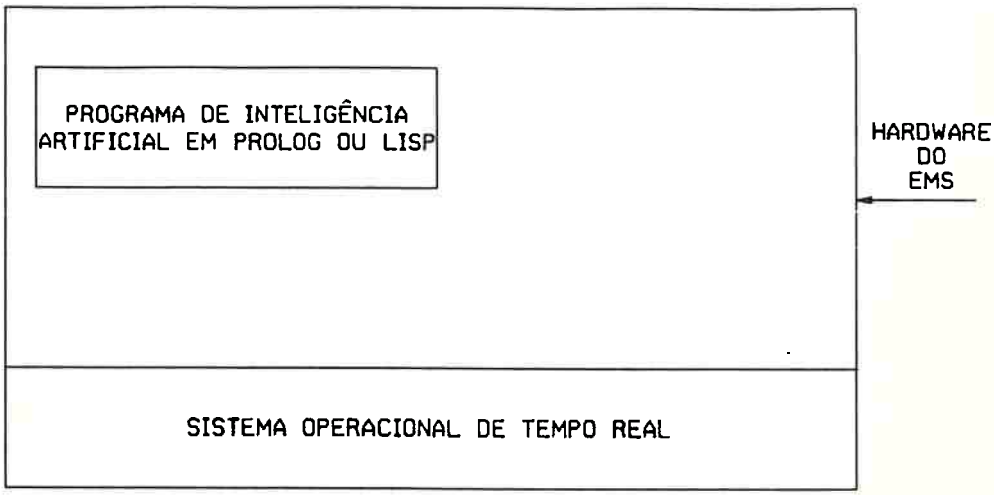


FIGURA IV.4.3 - IMPLEMENTAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM EMS UTILIZANDO PROLOG OU LISP

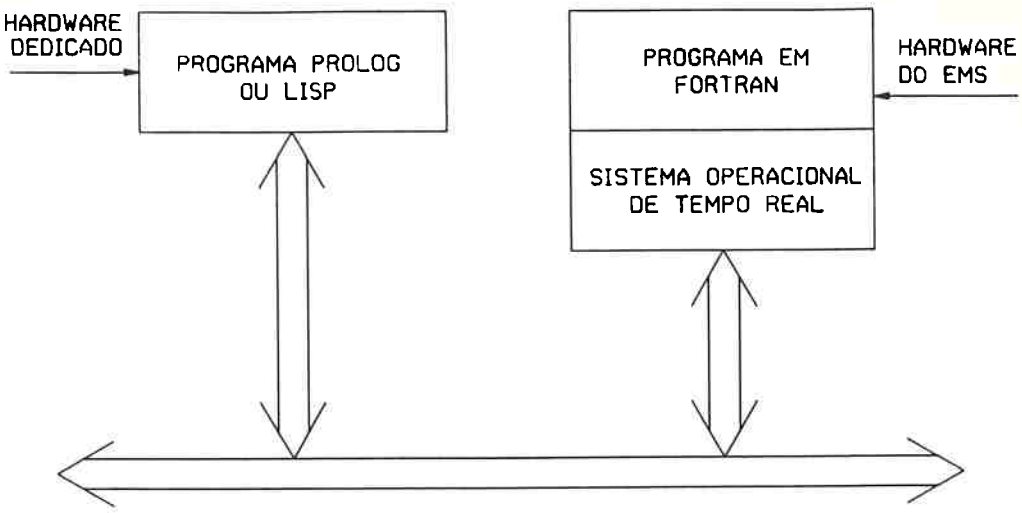


FIGURA IV.4.4 - IMPLEMENTAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM EMS UTILIZANDO HARDWARE DEDICADO

utilizando-se um hardware especial que execute eficientemente LISP ou PROLOG e que esteja ligado ao EMS (figura IV.4.4).

Serão analisadas, a seguir, algumas aplicações voltadas ao processamento em tempo real.

Uma das possíveis aplicações refere-se ao processamento de alarmes.

Os Sistemas de Gerenciamento de Energia possuem maneiras de processar alarmes para alertar os operadores de parâmetros que estão fora da faixa ou de mudanças que possam afetar a operação normal do sistema. Os alarmes são gerados em situações como operação de disjuntores, sobrecorrente, desvio de frequência, variação de tensão, operação de elementos de proteção e não funcionamento de controladores remotos. Em alguns sistemas pode-

se ter de 20000 a 50000 pontos analógicos e digitais sendo monitorados periodicamente (tipicamente a cada 2-10 segundos). Um processador de mensagens de tais sistemas pode gerar 500 ou mais mensagens por minuto (ver referência [34]). Frequentemente é impossível ao operador analisar o sistema de potência e tomar decisões apropriadas quando alarmes são mostrados nessa taxa. Além disso, pode-se citar outros problemas enfrentados pelos operadores durante o dia-a-dia de operações do sistema de potência: alarmes não muito específicos, alarmes muito específicos, muitos alarmes durante um distúrbio, falta de alarmes em parâmetros chave, alarmes falsos, multiplicidade de alarmes para o mesmo evento, alarmes mudando muito rapidamente para serem lidos no vídeo, alarmes em ordem não prioritária e alarmes permanecendo no vídeo após terem sido reconhecidos.

Na referência [35] é apresentado um processador inteligente de alarmes ("Intelligent Alarm Processor - IAP") que faz uma análise contínua das mensagens de alarme geradas por um EMS, apresentado-as de maneira transformada e relatando a condição do sistema, ao invés de simplesmente imprimir um grande número de mensagens específicas, além de permitir mudar dinamicamente as prioridades dos alarmes com as mudanças de estado do sistema.

O objetivo deste Sistema Especialista não é substituir o processamento de alarmes existente, mas complementá-lo. O operador pode ter acesso a todas as mensagens de alarme geradas pelo EMS quando for necessário.

A figura IV.4.5 mostra um diagrama do Sistema Especialista.

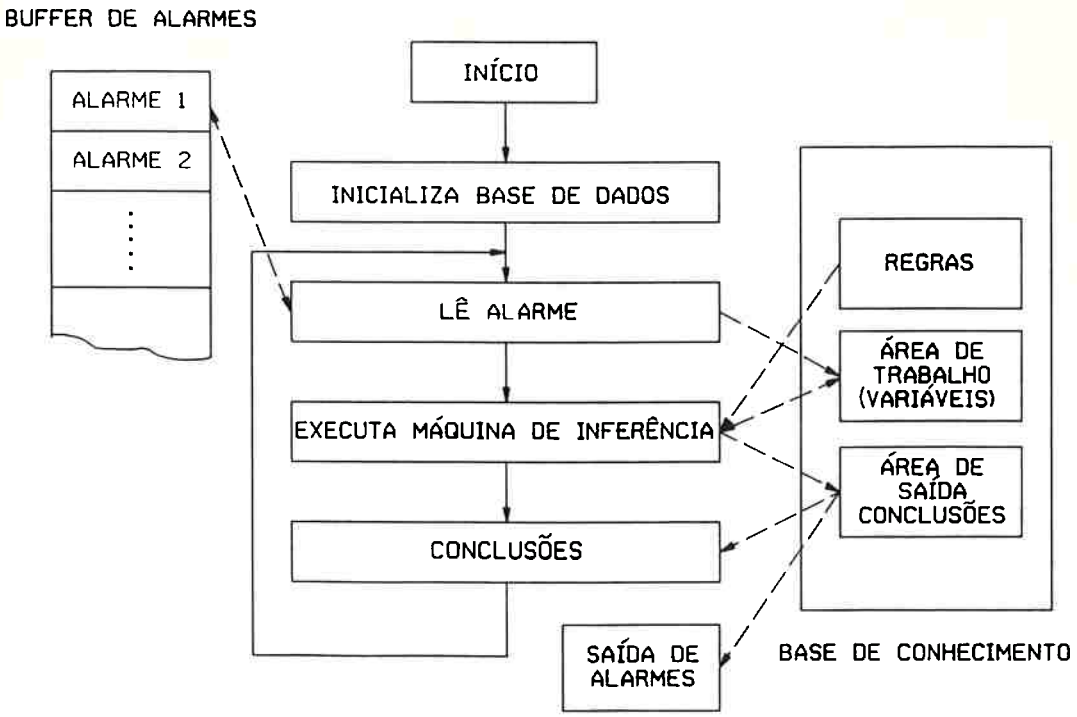


FIGURA IV.4.5 - DIAGRAMA DO IAP

O IAP possui uma base de conhecimento e uma máquina de inferência. A máquina de inferência é o módulo responsável pela análise dos alarmes gerados pelo EMS, a partir das informações contidas na base de conhecimento. Conforme a conclusão resultante, o IAP pode suprimir um alarme, imprimí-lo ou apresentar uma mensagem especial. A base de conhecimento foi obtida estudando-se as mensagens de alarme e perguntando para cada mensagem: "o que deve ser mostrado e o que poderia ser suprimido?". O conhecimento é representado por regras da forma:

IF ALARM A AND

ALARM B

THEN ISSUE ALARM C

As regras da base de conhecimento são divididas em várias categorias: regras para alarmes de nível, regras para perda de geração, regras para supressão de alarmes, regras para impressão de alarmes e de mensagens especiais.

As regras para alarmes de nível determinam o tipo de elemento (barramento, linha ou transformador) que tem algum parâmetro em alarme. Determinam também se esse elemento já foi alarmado e se o mesmo mudou de nível, por exemplo de alarme nível 1 para alarme nível 2.

As regras de perda de geração verificam alarmes de ações de chaveamento, alarmes de controle e alarmes de estados de geradores para determinar qual gerador foi perdido. Isso permite ao IAP sugerir a perda de geração mesmo que uma mensagem de perda de geração não tenha sido recebida.

As regras para supressão de alarmes determinam os casos em que alarmes podem ser eliminados. Por exemplo, poderiam ser eliminados todos os alarmes de disjuntores abertos ou fechados se mensagens de conexão, desconexão, energização ou desenergização de linhas ou transformadores forem recebidas.

As regras de impressão de alarmes determinam quando imprimir ou mostrar alarmes. Por exemplo, podem ser impressos todos os alarmes de ligação ou desligamento de geradores.

As regras de impressão de mensagens especiais cuidam de mensagens que são derivadas de alguma combinação de significado especial. Exemplos de mensagens especiais são:

"ANALYSIS SUGGESTS GENERATION LOSS"

"ANALYSIS SUGGESTS LOAD LOST"

"PROCESSING ERROR"

O IAP foi escrito em LISP, linguagem escolhida pelas facilidades que oferece na manipulação de símbolos. Um exemplo de implementação de uma regra e seu código correspondente em LISP é

apresentado a seguir.

```
R31: IF      ALARM = DC
     AND     NOT (DE_ALARM_ISSUED_ON_SAME_LINE_ALREADY)
     AND     DIFF (PRESENT_TIME TIME_OF_DC_ALARM)(SCANNING_TIME
     THEN    SUPRESS ALARM
```

O código correspondente em LISP é:

```
(B - RULE ALARM 31
          (IF
             (EQUAL ALARM 'DC)
             (NOT  DE_ALARM_ISSUED_ON_SAME_LINE_ALREADY)
             (LESSEQP
                    (DIFF PRESENT_TIME TIME_OF_DC_ALARM)
                    SCANNING_TIME))
          (THEN
             (SET_PARAM 'ALARM_ANALYSIS
                  '(RECOMMENDS SUPRESSING THIS ALARM))))
```

onde: LESSEQP é verdadeiro se o primeiro argumento é menor ou igual ao segundo argumento.

DIFF retorna a diferença entre 2 argumentos na forma de um número positivo.

DC é alarme que indica linha/transformador desconectado (aberto em um lado).

DE é alarme que indica linha/transformador desenergizado (aberto em ambos os lados).

A base de conhecimento deve possuir regras que otimizem o caminho de análise de cada alarme. Uma das exigências de projeto de um processador inteligente de alarmes é o tempo de processamento, que deve ser dimensionado de tal forma que a

apresentação dos resultados das análises não se tornem lentas demais.

Outra área de interesse para o desenvolvimento de Sistemas Especialistas é relacionado com o problema de diagnóstico. Para ilustrar o que vem a ser um diagnóstico considere-se um sistema que receba como entradas eventos e forneça como saídas resultados.

Três tipos de problemas podem ser resolvidos sobre esse sistema: análise, diagnóstico e síntese. (ver figura IV.4.6). No problema de análise, tem-se os eventos e o sistema conhecidos e deseja-se determinar os resultados. No problema de diagnóstico, tem-se o sistema e os resultados conhecidos e deseja-se determinar os eventos que causaram os resultados. Finalmente, no processo de síntese, tem-se os eventos e os resultados conhecidos e deve-se determinar um modelo para o sistema.

PROBLEMA	EVENTOS	SISTEMA	RESULTADOS
ANÁLISE	CONHECIDOS	CONHECIDO	DESCONHECIDOS
DIAGNÓSTICO	DESCONHECIDOS	CONHECIDO	COMHECIDOS
SÍNTESE	CONHECIDOS	DESCONHECIDO	CONHECIDOS

FIGURA IV.4.6 - TIPOS DE PROBLEMA SOBRE UM SISTEMA

Dentre as tarefas de monitoração e controle da rede elétrica, o diagnóstico de faltas no sistema elétrico é essencial. O objetivo do problema é estimar quando e que tipo de evento ocorreu, a partir dos alarmes gerados pelo Centro de Gerenciamento de Energia (EMS). Em geral, os eventos correspondem a distúrbios (como faltas), reações corretas do sistema a distúrbios (por exemplo, abertura de disjuntores) e falhas de operação de dispositivos (operação inadequada de disjuntores, alarmes falsos, alarmes não recebidos). Os diagnósticos devem partir da adoção de hipóteses e depois tentar verificar se elas explicam o conjunto de alarmes gerado. Por exemplo, para o alarme gerado na rede da figura IV.4.7 pode-se assumir as seguintes hipóteses:

a) O alarme informando a abertura de B1 é falso.

b) L1 tem uma falta permanente, B1 e B2 abriram e o alarme reportando a abertura de B2 foi perdido.

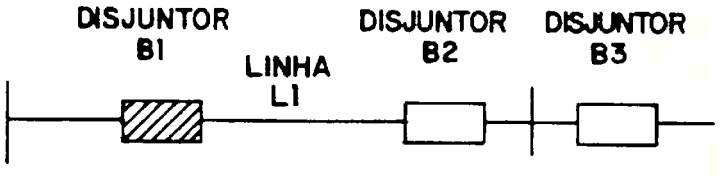


FIGURA IV.4.7 - EXEMPLO DE REDE ELÉTRICA

c) L1 tem uma falta permanente, B2 falhou no processo de abertura, B1 e B3 operaram e o alarme reportando a abertura de B3 não foi recebido.

Esse exemplo fornece uma idéia do quadro complexo que pode-se formar em situações de emergência.

Torna-se difícil diagnosticar os eventos que ocorreram, pois o padrão de alarmes torna-se complexo. Além disso, os operadores tem pouco tempo para analisar os eventos que ocorreram e planejar uma resposta.

Por isso, Sistemas Especialistas de diagnóstico podem se tornar ferramentas muito úteis aos operadores, já que permitem diminuir o tempo de diagnóstico dando mais tempo aos operadores planejar uma resposta.

A referência [32] apresenta um Sistema Especialista cuja função é identificar distúrbios e falhas envolvidas em uma modificação acidental da configuração da rede elétrica, denominado de "Diagnostician". Esse Sistema Especialista é um sistema baseado em regras, escrito em OPS5, e contém cerca de 350 regras. O diagnóstico parte dos estados inicial e final da rede, e de um conjunto de hipóteses. Exemplos de hipóteses são ocorrências de uma falta, uma falta e uma falha de operação de um relé, uma falta e uma falha de operação de um disjuntor, etc. As hipóteses adotadas são aplicadas a um simulador que reproduz o comportamento da rede, a partir do seu estado inicial. O estado final da rede gerado pelo simulador é comparado com o estado final real da rede. O diagnóstico consistirá do conjunto de hipóteses que explicam a transição da configuração da rede

elétrica do estado inicial para o final.

Outros desenvolvimentos de Sistemas Especialistas de diagnósticos de eventos do Sistema de Potência são apresentados nas referências [21], [30], [31] e [33].

Uma outra área de aplicação refere-se às operações de chaveamento em subestações. As operações de chaveamento tem como objetivo efetuar mudanças nos circuitos elétricos para ligar ou desligar cargas conectadas à subestação e podem ser executadas tanto em situações de emergência como em atividades programadas ou de manutenção. Para garantir que essas operações sejam executadas com segurança, devem existir mecanismos de intertravamento que não permitam chaveamentos que causem danos ao sistema elétrico ou aos equipamentos.

Esquemas de intertravamento convencionais são descritas em termos de equações booleanas e são específicos para cada diagrama da subestação. Os intertravamentos resultantes são implementados através de relés eletromecânicos, ou então mais recentemente, através de microprocessadores. Neste caso, os esquemas de intertravamento são expressos através de algoritmos. Entretanto, se houver uma mudança no projeto da subestação ou na sua filosofia operacional deve-se alterar os programas que representam os algoritmos, o que pode ser uma tarefa difícil. A utilização de representação dos esquemas de intertravamento através das técnicas de representação do conhecimento por regras é uma nova abordagem. Pode-se dividir os intertravamentos em dois componentes: o primeiro composto de regras que descrevem as operações dos dispositivos de chaveamento, e o segundo que descreve a arquitetura da subestação.

Note-se que em caso de alteração do projeto da subestação necessita-se alterar apenas a parte da base de conhecimento que descreve a arquitetura da subestação. Além disso, uma representação de uma especificação através de métodos declarativos como regras é mais fácil de se atualizar do que programas que representam algoritmos

Na referência [22] é feita a descrição de um Sistema Especialista que procura implementar essa nova abordagem nos Sistemas de Controle e Supervisão para operações de chaveamento de subestações. Na figura IV.4.8 pode-se observar um diagrama desse Sistema Especialista, que foi escrito em PROLOG e possui como interface homem-máquina uma interface gráfica escrita em C.

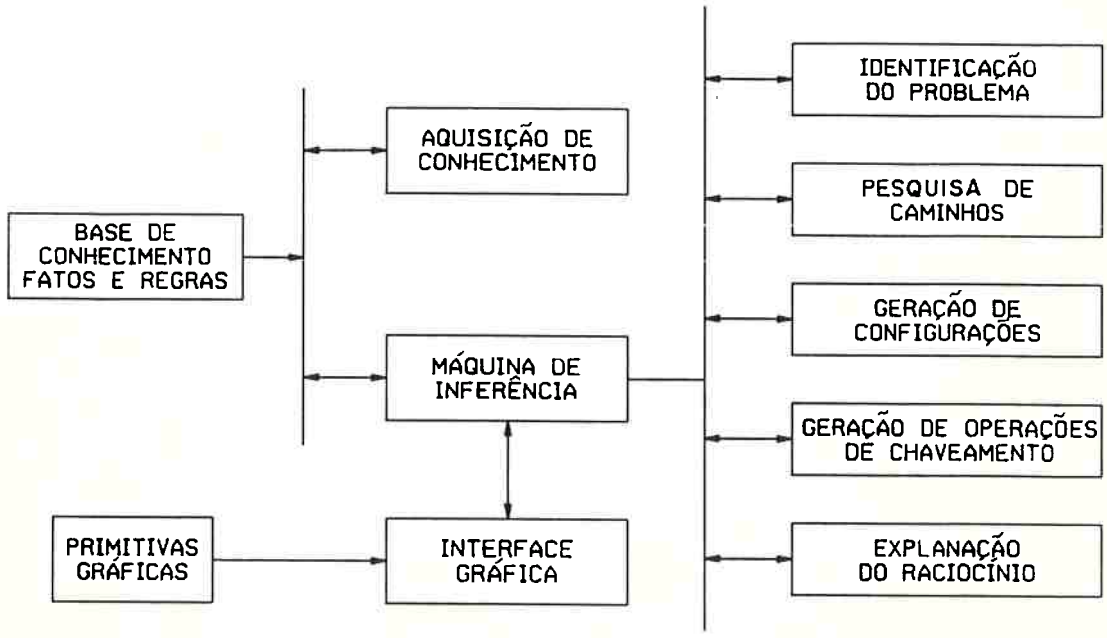


FIGURA IV.4.8 - DIAGRAMA DE BLOCOS DO SISTEMA ESPECIALISTA DE OPERAÇÕES DE CHAVEAMENTO EM SUBESTAÇÕES

A base de conhecimento é dividida em base de fatos e base de regras. A base de fatos é responsável pela descrição da topologia da subestação, dos dispositivos que a compõem (geradores, transformadores, disjuntores, chaves secionadoras) e dos estados operacionais desses dispositivos. Na figura IV.4.9 tem-se um diagrama de um trecho de um circuito. Os tipos de declarações que fazem parte da base de fatos são:

```
fact: d3 boundary b2 connected_to t1 boundary b1
fact: d3 type breaker
      t1 type transformer
fact: d3 state off
```

A base de regras descreve as regras de operação dos dispositivos de chaveamento. Essas regras são independentes da topologia da rede, mas dependentes dos componentes. Por exemplo, uma chave secionalizadora só pode ser aberta quando não houver tensão em seus dois terminais simultaneamente. A descrição dessa regra é apresentada a seguir:

```
isolator_rule1: if X type isolator
                and no_voltage_supply(X)
                and switch_off(X)
                then open(X)
```

O Sistema Especialista pode verificar se uma operação de chaveamento é permitida, explicando eventualmente, quais foram os seus passos de raciocínio. Além disso, ele pode propor sequências de chaveamento no caso de haver cargas não-supridas. Isso é interessante em situações de emergência. A identificação do estado de emergência e a posterior reconfiguração da subestação

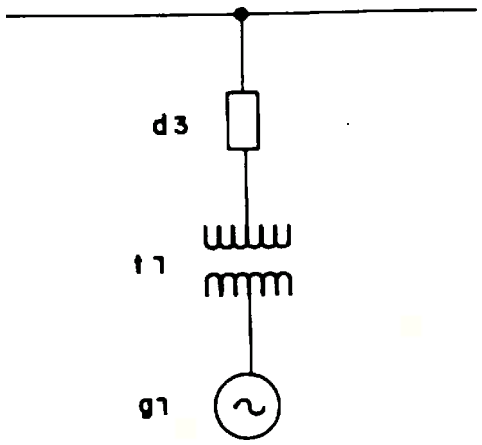


Figura IV.4.9

deve iniciar com uma análise de alarmes e identificação dos elementos que falharam e das cargas não supridas. Depois deve-se fazer a pesquisa de uma nova configuração da subestação. Finalmente executa-se as operações de chaveamento para se chegar à nova configuração.

A máquina de inferência tem acesso a outros módulos de conhecimento, que a orientam a identificar problemas (como cargas não supridas), pesquisa de caminhos pela rede, de configuração da subestação e de sequências de chaveamento, além de um módulo de explanação.

A referência [39] também apresenta um sistema que implementa os esquemas de intertravamento através da metodologia apresentada, isto é, divisão do conhecimento em parte dependente da arquitetura da subestação e parte dependente de seus componentes. O artigo apresenta tabelas onde são descritas regras para operação de dispositivos de chaveamento (disjuntores, secionadores, chaves de aterramento), que implementam requisitos de segurança (de pessoas e de equipamentos), de seletividade, entre outros.

O desenvolvimento de Sistemas Especialistas aplicados aos problemas de Sistemas de Potência em sua maioria ainda restringem-se aos sistemas experimentais. O uso desses sistemas no dia-a-dia da operação dos Sistemas de Potência depende de teste exaustivos para adquirir e demonstrar a sua confiabilidade.

Notas bibliográficas: este capítulo é baseado nas referências [8], [9], [17], [18], [19], [21], [22], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [39], [42] e [43]. A parte referente à discussão sobre os Sistemas de Potência é baseada nas referências [8], [9], [42] e [43]. As referências [24], [27] e [34] tratam dos Sistemas de Supervisão e Controle dos Sistemas de Potência. Aplicações de Inteligência Artificial em Sistemas de Potência são apresentadas em [17], [18], [19], [25], [26], [28], [29], e [32]. As referências [21], [30], [31] e [33] apresentam aplicações de Inteligência Artificial que envolvem o problema do diagnóstico de eventos no Sistema de Potência, [22] e [39] tratam das operações de chaveamento em subestações, e [35] discute o processamento inteligente de alarmes.

V - Análise da Construção de um Sistema Especialista de Diagnóstico de Faltas, utilizando PROLOG.

Em Sistemas de Potência umas das tarefas que estão sob responsabilidade dos operadores é o diagnóstico de faltas. Quando ocorre uma falta, o sistema de proteção deve atuar e isolar a área afetada do resto do Sistema de Potência. Durante a ocorrência desses eventos, o Sistema de Supervisão e Controle recebe mensagens de operação de relés e disjuntores, permitindo que seja apresentada a nova topologia da rede elétrica. A estimativa da região onde ocorreu a falta é o primeiro passo nos procedimentos de restauração do Sistema de Potência. Operadores experientes dos centros de controle podem localizar onde ocorreu uma falta através das mensagens que indicam a operação de relés e disjuntores, e do conhecimento da topologia da rede antes e depois da ocorrência do distúrbio. Entretanto, em situações mais complexas, como na ocorrência de faltas múltiplas ou falhas na operação de dispositivos, aumenta a dificuldade para se fazer um diagnóstico. Além disso, existe o problema da barreira cognitiva, isto é, a incapacidade dos operadores absorverem todas as informações de alarmes e do estado do Sistema de Potência, para realizar um diagnóstico num tempo hábil. Diante desses fatos, vários Sistemas Especialistas para diagnóstico de faltas tem sido desenvolvidos. Os métodos de representação do conhecimento e resolução de problemas, nos quais são baseados os Sistemas Especialistas, são abordagens que se encaixam adequadamente às necessidades de inferências implícitas ao problema de diagnóstico.

Para se resolver o problema de diagnóstico de faltas, deve-se inicialmente tentar formalizá-lo. Conforme foi discutido

no ítem IV, dado um sistema que receba eventos como entradas e forneça resultados como saídas, tem-se que um processo de diagnóstico é uma tarefa em que se conhece os resultados e o sistema, e deseja-se determinar os eventos que ocorreram. No caso do diagnóstico de faltas, o sistema representa o Sistema de Potência, com um certo estado inicial, e o resultado é o seu estado final. Os eventos representam as faltas ou falhas de operação que ocorreram (ver figura V.1).

Pode-se descrever o problema do diagnóstico de faltas através da representação por espaço de estados. Um ponto no espaço de estados do problema representa uma determinada topologia da rede. A transição de um ponto para outro é descrita por regras de operação de relés e disjuntores.

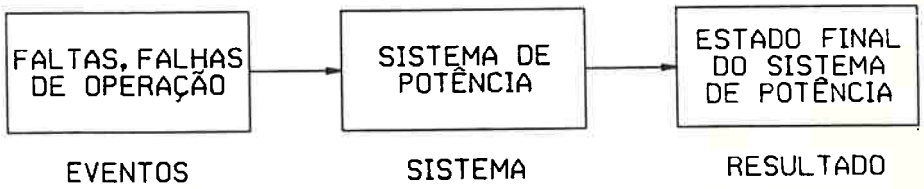


FIGURA V.1 - PROBLEMA DO DIAGNÓSTICO DE FALTAS

Os conceitos e metodologias utilizados para a análise e construção de um Sistema Especialista de Diagnóstico de Faltas são baseados no trabalho apresentado na referência [20].

A topologia do Sistema de Potência pode ser descrita através de um grafo não-orientado, onde os ramos representam os disjuntores e os nós as áreas que podem ser desconectadas de outras partes do sistema. Como exemplo, considere-se o Sistema de Potência apresentado na figura V.2. Para esta rede elétrica resulta o grafo apresentado na figura V.3.

Cada disjuntor pode estar num estado aberto ou fechado. Abrindo ou fechando-se os disjuntores muda-se a topologia da rede.

A operação de um relé é descrita pela seguinte regra:

"O relé opera se a falta ocorreu na sua seção de proteção S."

Então, se ocorrer uma falta simples numa seção que pode-se denominar de Sf, esta seção será a intersecção das seções de proteção de todos os relés que operaram (ver a figura V.4).

No caso de ocorrerem faltas múltiplas, várias intersecções aparecerão, cada uma correspondendo a uma falta (ver figura V.5).

Para se realizar o diagnóstico de faltas pode-se utilizar o método de pesquisa para trás, isto é, escolhe-se uma seção Sf e tenta-se verificar se esta seção explica o estado final através das regras de operação dos relés e disjuntores. A

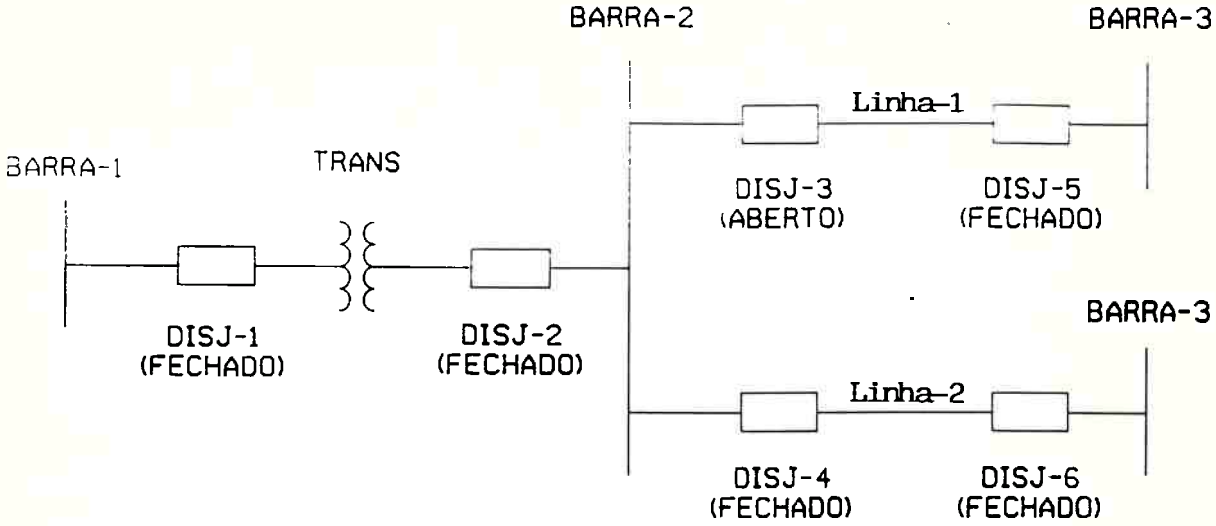


FIGURA V.2 - DIAGRAMA UNIFILAR DE UM SISTEMA DE POTÊNCIA

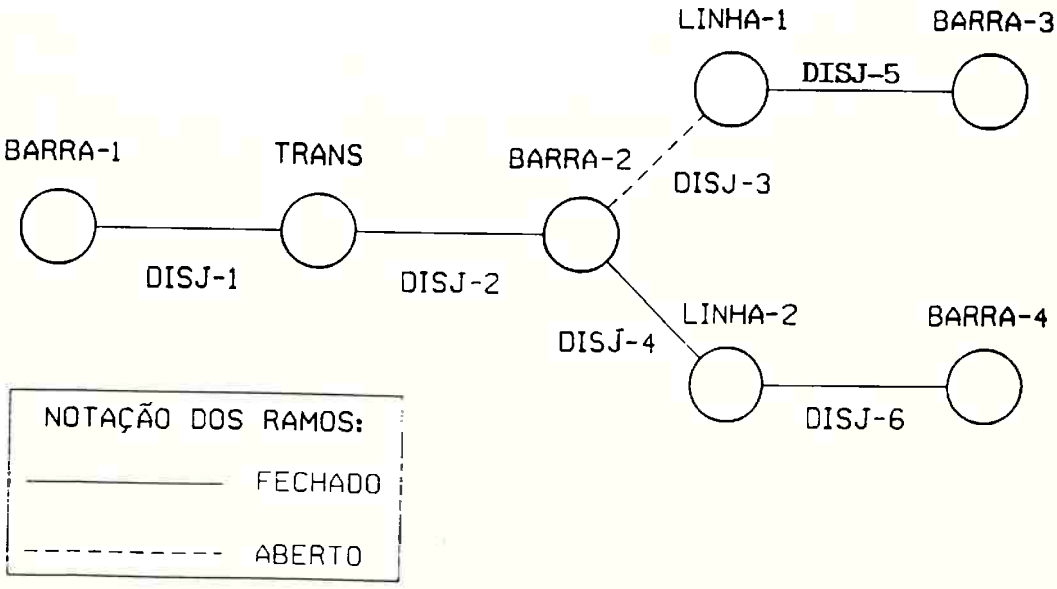


FIGURA V.3 - GRAFO QUE REPRESENTA O SISTEMA DE POTÊNCIA DA FIGURA V.2

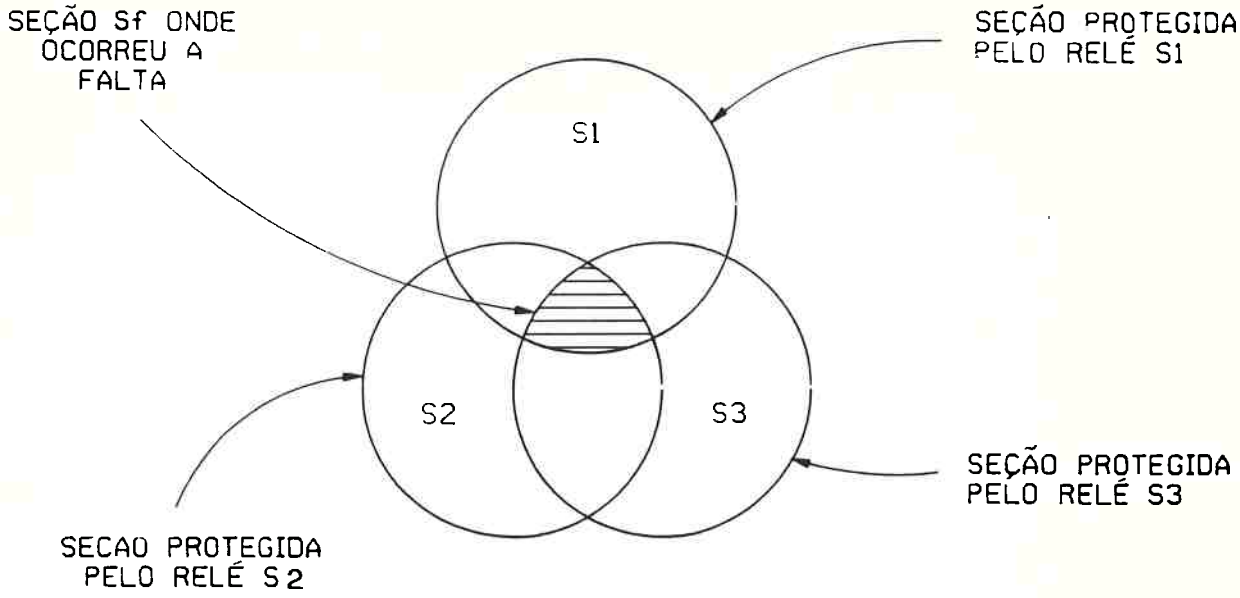


FIGURA V.4 - ESQUEMA DE SEÇÃO DE OCORRÊNCIA DE UMA FALTA SIMPLES

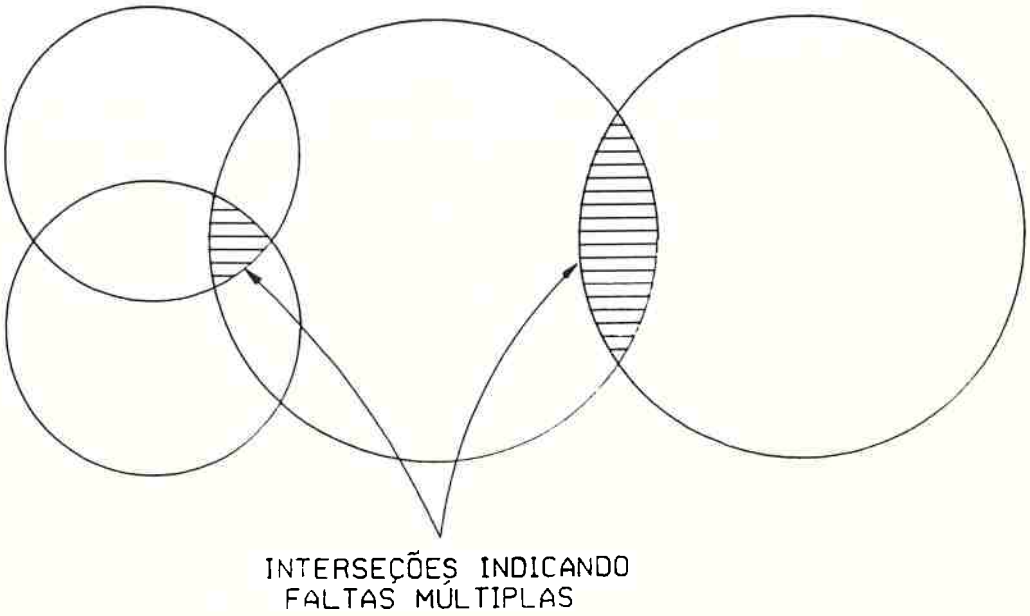


FIGURA V.5 - ESQUEMA DE SEÇÕES DE OCORRÊNCIA DE FALTAS MÚLTIPLAS

escolha da seção Sf, onde supõe-se que a falta tenha ocorrido, pode obedecer a uma heurística que permita otimizar o processo de inferência.

A linguagem PROLOG fornece os mecanismos que permitem resolver o problema com a abordagem apresentada.

O Sistema de Potência pode ser descrita pela seguinte declaração em PROLOG:

rede(⟨nó 1⟩, ⟨disjuntor⟩, ⟨nó 2⟩, ⟨estado do disjuntor⟩).

Para o Sistema de Potência apresentado na figura V.2 tem-se as seguintes declarações:

```
rede("BARRA_1","DISJ_1","TRANS",conectado).
rede("TRANS","DISJ_2","BARRA_2",conectado).
rede("BARRA_2","DISJ_3","LINHA_1",desconectado).
rede("BARRA_2","DISJ_4","LINHA_2",conectado).
rede("LINHA_1","DISJ_5","BARRA_3",conectado).
rede("LINHA_2","DISJ_6","BARRA_4",conectado).
```

Nos sistemas de proteção, vários tipos de relés são combinados para efetuar a proteção dos Sistemas de Potência. Geralmente os centros de controle reconhecem sinais integrados de cada sistema de proteção e não de sinais individuais de cada componente. Isso significa que há interesse de informações apenas em um nível integrado e simplificado, e não num nível detalhado. Esse nível integrado e simplificado é descrito por relés que serão denominados "relés simplificados". A figura V.6 apresenta alguns tipos de relés simplificados.

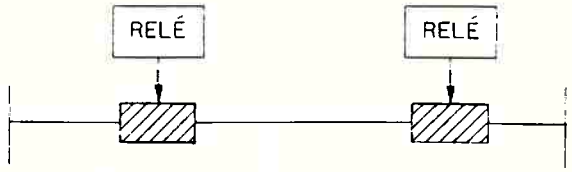
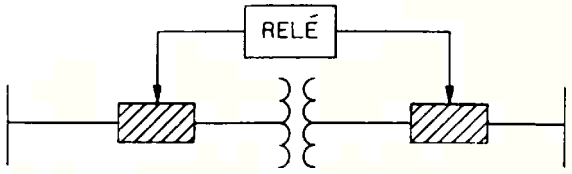
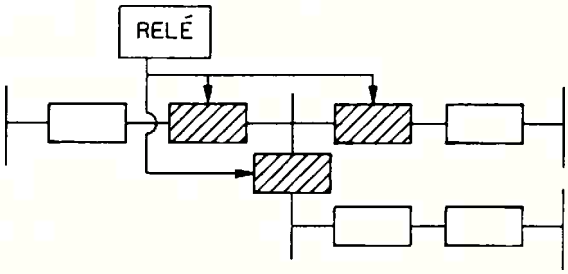
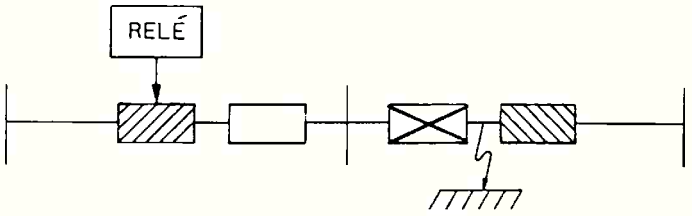
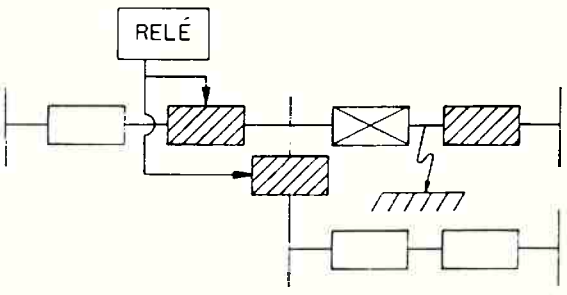
TIPO	MODO DE OPERAÇÃO	DIAGRAMA ESQUEMÁTICO
mr	RELÉS PARA PROTEÇÃO PRIMÁRIA DE LINHAS	
tr	RELÉ PARA PROTEÇÃO PRIMÁRIA DE TRANSFORMADORES	
br	RELÉ PARA PROTEÇÃO PRIMÁRIA DE BARRAMENTOS	
rr	RELÉS DE DISTÂNCIA	
lr	RELÉS DE "BACK-UP" LOCAL	

FIGURA V.6 - ALGUNS TIPOS DE RELÉS REDUZIDOS

Os relés simplificados podem ser representados pela seguinte declaração PROLOG:

relé(⟨nome do relé⟩, ⟨tipo de relé⟩, ⟨disjuntores⟩, ⟨nós⟩, ⟨relés primários⟩, ⟨nós para trás⟩).

Como argumentos dessa declaração tem-se que ⟨nome do relé⟩ representa a identificação do relé reduzido (conforme a figura V.6), ⟨tipo do relé⟩ é um dos tipos de relés reduzidos apresentados na figura V.6, ⟨disjuntores⟩ representa o conjunto de disjuntores disparados pelo relé reduzido, ⟨nós⟩ representa a área primária de proteção se o relé é do tipo mr, tr ou br, ⟨relé primário⟩ é necessário somente se o relé é do tipo lr e representa o nome dos relés primários das quais é "back-up", ⟨nós para trás⟩ é necessário somente se o relé é do tipo rr e representa os nós para trás da área de proteção.

Para cada tipo de relé reduzido descrito na figura V.6 existe uma regra de operação. Por exemplo, para os relés dos tipos mr, tr e br a regra de operação é a seguinte:

"O relé opera se a falta ocorre na sua seção de proteção."

A declaração em PROLOG da regra acima é:

```
opera(Rele,Falta):-
                    rele(Rele,⟨tipo⟩,_,Secao,[],[]),
                    membro(Falta,Secao).
```

onde ⟨tipo⟩ = mr, tr ou br.

Para encontrar a seção de faltas, no caso de falta simples, deve-se procurar a seção que satisfaz a todas as

cláusulas "opera(Rele,Falta)", ou seja, seleciona-se uma seção através de algum critério ou heurística e verifica-se se ela atende às regras de operação de todos os relés que atuaram. Então, se a seção selecionada estiver ligada à variável Falta, a consulta que deve ser realizada ao programa PROLOG, para verificar se ocorreu uma falta simples, deve ser:

opera(rele_1,Falta), opera(rele_2,Falta),.., opera(rele_n,Falta).

Para exemplificar o problema, considere-se um programa que descreve o Sistema de Potência da figura V.3 e a regra de operação do relé Rele_1, do tipo "br":

```
/* Clausulas que descrevem o Sistema de Potencia
   apos a ocorrencia de uma falta na secao Barra_2 */
rede("Barra_1","Disj_1","Trans",conectado).
rede("Trans","Disj_2","Barra_2",desconectado).
rede("Barra_2","Disj_3","Linha_1",desconectado).
rede("Barra_2","Disj_4","Linha_2",desconectado).
rede("Linha_1","Disj_5","Barra_3",conectado).
rede("Linha_2","Disj_6","Barra_4",conectado).

/* Descricao do Rele_1, tipo br */
rele("Rele_1",br,["Disj_2","Disj_3","Disj_4"],
     ["Barra_2"],[],[]).

/* Regra de operacao dos reles tipo br */
opera(Rele,Falta):-
     rele(Rele,br,_,Secao,[],[]),
     membro(Falta,Secao).
```

```
membro(X, [X|_]).
membro(X, [_|Y]):- membro(X,Y).
```

Nesse exemplo, o relé Rele_1 atua sobre os disjuntores Disj_2, Disj_3, Disj_4. Se houver uma falta na seção Barra_2, o relé Rele_1 deve operar e disparar a abertura dos três disjuntores. Supondo que o relé Rele_1 tenha operado corretamente e que o programa, por algum critério ou heurística, tenha selecionado a seção "Barra_2", então a resposta á consulta "opera("Rele_1","Barra_2")" é:

```
Goal: opera("Rele_1","Barra_2")
Yes
```

Essa consulta irá confirmar a hipótese de uma falta simples na seção "Barra_2".

Se não for possível encontrar uma solução adotando a hipótese de falta simples, deve-se adotar regras que concluam por múltiplas faltas, regras que consigam encontrar respostas na falta de informações ou regras que encontrem falhas de operação. O uso dessas regras segue uma hierarquia, que é apresentada na figura V.7.

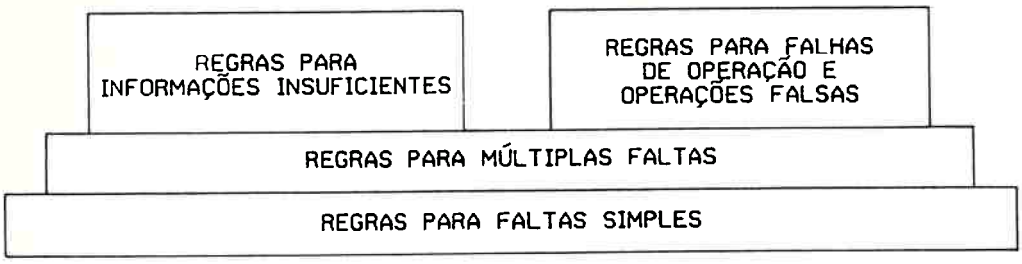


FIGURA V.7 - HIERARQUIA PARA ADOÇÃO DE REGRAS

VI - Considerações Finais.

Neste trabalho foi efetuado um estudo sobre técnicas de Inteligência Artificial e suas possíveis aplicações nos Sistemas de Supervisão e Controle de Sistemas de Potência.

Essas técnicas de Inteligência Artificial contribuem com novos métodos de representação e resolução de problemas, que permitem o desenvolvimento de programas que podem realizar inferências a partir de um certo conjunto de conhecimentos e manipular informações incertas ou incompletas. Sistemas Especialistas, que são programas capazes de resolver problemas em áreas específicas de conhecimento, são exemplos desses tipos de programas.

O estudo das técnicas de Inteligência Artificial permite obter uma visão conceitual de como representação e resolução de problemas poderiam ser implementados num programa. Existe uma liberdade na escolha da forma de se implementar essas técnicas de Inteligência Artificial. Pode-se utilizar diversas abordagens, e a ferramenta matemática escolhida para análise neste trabalho foi a Lógica.

Verificou-se que a Lógica, através dos Sistemas de Programação em Lógica fornece um novo paradigma de programação, a Programação Declarativa, que permite descrever a resolução de um problema através de regras e não através de algoritmos. Em outras palavras, um programa em lógica descreve "o que deve ser feito" (lógica do problema) e não "como deve ser feito" (algoritmo). Quem se encarrega da tarefa de como as regras serão usadas é o mecanismo de inferência que for implementado na máquina de

execução dos programas em lógica. A vantagem dessa abordagem reside no fato de que o ser humano tem maior facilidade em descrever regras do que pensar em termos de algoritmos. Além disso, a utilização de um programa em Lógica permite descrever o problema e os dados através da mesma notação, que são as cláusulas, que expressam fatos e regras.

Um dos resultados da mecanização da Lógica é a linguagem PROLOG. O interesse nessa linguagem reside no fato de que, além de fornecer algumas características da Programação em Lógica, ela permite implementar mecanismos de processamento paralelo para aumentar a sua eficiência, já que deficiências no desempenho de realização de inferências é um dos problemas nos sistemas de processamento simbólicos atuais. A linguagem PROLOG utiliza as cláusulas de Horn para a descrição de fatos e regras e adota como mecanismo de inferência um procedimento denominado de Resolução.

O novo ambiente de programação fornecido pela linguagem PROLOG e os computadores especialmente construídos para executar programas nessa linguagem podem representar um grande avanço nos Sistemas de Supervisão e Controle de Sistemas de Potência. Exemplos de aplicações consistem de Sistemas Especialistas que possam realizar tarefas de tempo real como processamento de alarmes, chaveamento em subestações e diagnóstico de eventos.

Para que Sistemas Especialistas possam se integrar de forma definitiva aos Sistemas de Supervisão e Controle, devem ser estudados e resolvidos problemas como a sua construção de tal maneira que atendam a requisitos de aplicações de tempo real, isto é, realizem as inferências solicitadas em tempos admissíveis. Esses estudos podem envolvem tanto a incorporação de

máquinas dedicadas, como máquinas PROLOG, ou a implementação dos novos programas nas máquinas já existentes.

Outro campo importante que resulta do problema da construção dos Sistemas Especialistas é a Engenharia do Conhecimento. Essa área de estudo é importante para a aquisição e modelagem do conhecimento específico necessário para se resolver problemas como o processamento de alarmes, diagnóstico de eventos e restauração do Sistema de Potência.

Esse fato foi ilustrado no capítulo IV, onde foi descrito como poderia ser construído um Sistema Especialista de Diagnóstico de Faltas, baseado na linguagem PROLOG. As regras apresentadas representam os casos mais simples. Para que se possa implementar um Sistema Especialista de Diagnóstico de Faltas realmente operacional deve-se realizar todo um trabalho que envolve Engenharia do Conhecimento, com o objetivo de se determinar as regras que permitam solucionar o problema.

Finalmente, espera-se que este trabalho forneça as bases conceituais para novos desenvolvimentos envolvendo técnicas de Inteligência Artificial, em particular aqueles que procurem utilizar a linguagem PROLOG, nos Sistemas de Supervisão e Controle de Sistemas de Potência. Exemplos de aplicação foram apresentados e uma área em que as técnicas analisadas podem ser de grande utilidade é aquela que se refere aos problemas de diagnóstico. Não apenas referindo-se ao diagnóstico de faltas, mas a uma infinidade de diagnósticos que podem e devem ser realizados na operação de Sistemas de Potência.

Bibliografia.

[1] Kowalski, R. Logic For Problem Solving. New York, Elsevier Publishing Co., Inc., 1979. 287p.

[2] Turbo Prolog 2.0 IBM Version User's Guide. Scotts Valley, Borland International, Inc., 1988. 479p.

[3] Kinnucan, P. Computers That think like experts High Technology, 30-42, Jan. 1984.

[4] Marcus, C. PROLOG Programming Reading, Addison-Wesley, 1986. 325p.

[5] Rich, E. Artificial Intelligence McGraw-Hill, Inc., 1983. 436p.

[6] Lucena C.J.P. Inteligência Artificial e Engenharia de Software Rio de Janeiro, Jorge Zahar Editor, 1987. 305p.

[7] Charniak, E.; McDermott, D. Introduction to Artificial Intelligence Reading, Addison-Wesley, 1987. 701p.

[8] Applied Protective Relaying Newark, Westinghouse Electric Corporation, 1976.

[9] Caminha, A.C. Introdução à Proteção dos Sistemas Elétricos São Paulo, Edgard Blucher, 1977. 211p.

[10] Gevarter, W.B. Expert Systems: limited but powerful IEEE Spectrum, 39-45, Aug. 1983.

[11] Clark, K.L.; McCabe, F.G. micro-PROLOG: PROGRAMMING IN LOGIC. London, Prentice-Hall, 1984. 401p.

[12] Kluzniak, F.; Szpakowicz, S. PROLOG for programmers. London, Academic Press, 1985. 400p.

[13] Campbell, J.A. Implementations of PROLOG. Ellis Horwood Limited, 1984. 391p.

[14] Weiss, S.M.; Kulikowski, C.A. A Practical Guide to Designing Expert Systems. Rowman & Allanheld, 1984. 174p.

[15] Wos, L.; Overbeek, R.; Lusk, E.; Boyle, J. Automated Reasoning, Introduction and Applications. London, Prentice Hall, 1984. 482p.

[16] Hayes-Roth, F. The Knowledge-Based Expert System. IEEE Computer, 11-28, Sept. 1984.

[17] Wollemberg, B.F.; Sakaguchi, T. Artificial Intelligence In Power System Operations. Proceedings of the IEEE, 75(12):1678-85, Dec. 1987.

[18] Kirschen, D. et alii. Artificial Intelligence Applications In An Energy Management System Environment. In: Symposium On Expert Systems Application To Power Systems, Stockholm-Helsinki, 1988. p.17.1-6

[19] Suzuki, H. et alii. Experiences Of Expert Systems For Power System Analysis In Japan. In: Symposium On Expert Systems Application To Power Systems, Sotckholm-Helsinki, 1988. p.1.1-7

[20] Fukui, C.; Kawakami, J. An Expert System For Fault Section Estimation Using Information From Protective Relays And Circuit Breakers. IEEE Transactions On power Delivery, PWRD-1(4):83-90, Oct. 1986.

[21] Hein, F. Expert System Using Pattern Recognition By Real Time Signals. In: Cigré International Conference On Large High Electric Systems, Paris, 1986. 39-10 p.1-6

[22] Stalder, M. et alii. A Rule-Based System For Substation Monitoring: The Switching Operations. In: Symposium On Expert Systems Application To Power Systems, Stockholm-Helsinki, 1988. p.6.1-8

[23] Waterman, D.A. A Guide to Expert Systems. Reading, Addison-Wesley, 1986. 419p.

[24] Fundamentals of Supervisory Control Systems. IEEE Tutorial Course, 81 EHO 188-3-PWR, 1981.

[25] Schulte, R.P. et alii. Artificial Intelligence Solutions To Power System Operating Problems. IEEE Transactions on Power Systems, PWRS-2(4):920-26, Nov. 1987.

[26] Taylor, T.; Lubkeman, D. Applications of Knowledge-Based Programming to Power Engineering Problems. IEEE Transactions on Power Systems, 4(1):345-52, Feb. 1989.

[27] Horton, J.S. et alii. Advances in Energy Management Systems. IEEE Transactions on Power Systems, PWRS-1(3): 226-34, Aug. 1986.

[28] Hein, F.; Schellstede, G. Use of Expert Systems in Energy Control Centres. In: Cigré International Conference on Large High Voltage Electric Systems, Paris, 1986. 39-15 p.1-6

[29] Liu, C.; Lee, S.J.; Venkata, S.S. An Expert System Operational Aid For Restoration and Loss Reduction of Distribution Systems. IEEE Transactions on Power Systems, 3(2):619-26, May 1988.

[30] Laresgoiti, I. et alii. LAIDA: Development of an Expert System for Disturbance Analysis in an Electrical Network. In: Symposium On Expert Systems Application To Power Systems, Stockholm-Helsinki, 1988. p.4/32-4/39

[31] Mihelcic, M.; Gubina, F.; Ogorelec, A. An Approach to Power Network Fault Location Diagnosis. In: Symposium On Expert Systems Application To Power Systems, Stockholm-Helsinki, 1988. p.4/26-4/31

[32] Talukdar, S.N.; Cardozo, E.; Perry, T. The Operator's Assistant - An Intelligent, Expandable Program For Power System Trouble Analysis. IEEE Transactions on Power Systems, PWRS-1(3):182-87, Aug. 1986.

[33] Cardozo, E.; Talukdar, S. A Distributed Expert System for Fault Diagnosis. IEEE Transactions on Power Systems, 3(2): 641-46, May 1988.

[34] Amelink, H.; Forte, A.M.; Guberman, R.P. Dispatcher Alarm and Message Processing. IEEE Transactions on Power Systems, PWRS-1(3):188-94, Aug. 1986.

[35] Wollenberg, B.F. Feasibility Study for an Energy Management System Intelligent Alarm Processor. IEEE Transactions on Power Systems, PWRS-1(2):241-47, May 1986.

[36] Hogger, C.J. Introduction to Logic Programming. London, Academic Press, 1984. 278p.

[37] Casanova, M.A.; Giorno, F.A.C.; Furtado, A.L. Programação em Lógica e a Linguagem PROLOG. São Paulo, Edgard Blucher, 1987. 461p.

[38] Graña, J.L. O Leitor Pergunta. Ciência Hoje. 9(52):18-20, Abril 1989.

[39] Brand, K.P.; Kopainsky, J.; Wimmer, W. Topology-Based Interlocking of Electrical Substations. IEEE Transactions on Power Delivery, PWRD-1(3):118-26, July 1986.

[40] Hwang, K.; Ghosh, J.; Chowkwanyun, R. Computer Architectures for Artificial Intelligence Processing. IEEE Computer, Jan. 1987. p.19-27

[41] Moto-Oka, T.; Stone, H.S. Fifth-Generation Computer Systems: A Japanese Project. IEEE Computer, March 1984. p.6-13

[42] Robba, E.J. Aspectos Gerais dos Sistemas Elétricos de Potência. São Paulo, Escola Politécnica, 1982. 55p.

[43] Stevenson, W.D. Elementos de Análise de Sistemas de Potência. São Paulo, McGraw-Hill, 1974. 347p.