UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO E SISTEMAS

PROTÓTIPO DE UMA FERRAMENTA PARA CRIAÇÃO DE SISTEMAS ESPECIALISTAS BASEADOS EM REGRAS DE PRODUÇÃO

DISSERTAÇÃO SUBMETIDA À UNIVERSIDADE FEDERAL DE SANTA CATARINA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA

ROBERTO HEINZLE

FLORIANÓPOLIS
SANTA CATARINA - BRASIL
NOVEMBRO DE 1995

"PROTÓTIPO DE UMA FERRAMENTA PARA CRIAÇÃO DE SISTEMAS ESPECIALISTAS BASEADOS EM REGRAS DE PRODUÇÃO"

ROBERTO HEINZLE

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE

MESTRE EM ENGENHARIA

ESPECIALIDADE ENGENHARIA DE PRODUÇÃO, OPÇÃO PESQUISA OPERACIONAL E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO.

RICARDO MIRANDA BARCIA, PhD
COORDENADOR DO PROGRAMA DE PÓS-GRADUAÇÃO

BANCA EXAMINADORA:

RENATO ANTONIO RABUSKE, DSc
PRESIDENTE

LUIZ CARLOS DUCLÓS, PhD

MARCIA AGUIAR RABUSKE, DSc

A Ivete,
ao Fernando,
ao Rodrigo.

A quem sempre me deu amor, carinho e apoio

RALF e RUTH, meus pais.

Àqueles que me deram incentivo

MAURÍCIO e MARILSE
ÁLVARO e ÂNGELA
PITA e LISE, meus irmãos e amigos.

AGRADECIMENTOS

Ao professor Renato Antonio Rabuske pela amizade, apoio e, principalmente, pela eficiente orientação ao longo do desenvolvimento deste trabalho.

À NTS - Núcleo de Tecnologia de Software Ltda que viabilizou financeiramente a realização do trabalho. Muito em particular, meu agradecimento ao diretor presidente da empresa, Luiz Carlos Duclós, que permanentemente me incentivou e apoiou.

Ao médico Peter Carlos Kühr que desenvolveu a base de conhecimentos da aplicação experimental que permitiu a realização dos testes do sistema desenvolvido.

Aos amigos Almir Küpers, Cassiano Reis e Érico Olavo Weissheimer pelas sugestões, comentários e apoio que foram importantíssimos e, certamente, enriqueceram o trabalho.

Ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas, em especial ao seu corpo docente, pela oportunidade da realização do curso e pelos ensinamentos oferecidos.

RESUMO

Este trabalho apresenta os aspectos relacionados a construção de sistemas especialistas. São mostrados os principais conceitos, um histórico da evolução destes sistemas bem como as funções e características dos elementos que os compõem.

As formas de representação e armazenamento do conhecimento, as técnicas de raciocínio e as ferramentas utilizadas no desenvolvimento de sistemas especialistas são abordados detalhadamente. Foi dado maior ênfase aos sistemas que utilizam as regras de produção na representação do conhecimento.

Adicionalmente, é especificado e implementado um protótipo de uma ferramenta para a construção de sistemas especialistas baseados em regras de produção. No desenvolvimento desta ferramenta, que é apresentada em detalhes, utilizou-se a linguagem de programação ARITY/PROLOG.

Para verificar a funcionalidade do protótipo desenvolvido procedeu-se uma aplicação prática. Foi criada e implementada uma aplicação experimental sobre uma base de conhecimentos na área médica. Esta aplicação experimental permite o diagnóstico de algumas doenças comuns do aparelho digestivo e respiratório.

ABSTRACT

This work presents the features of the expert systems design process. The main conceps, the historic review of these systems, as well as the functions and characteristics of its elements are showed here.

The knowledge representation and storage ways, reasoning technics and tools used to develop expert systems are detailed. The systems based on production rules used on knowledge representation are emphasized.

In addition of that, it is specified and implemented a prototype toll to build up expert systems based on production rules. ARITY/PROLOG was the programming language choosen to construct this tool.

In order to verify the prototype tool fitness, it was done a practical application. It was created an experimental application in the medical area. This experimental apllication allows to identify respiratory and digestive diseases.

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

1 - INTRODUÇÃO

1.1 - ORIGEM DO TRABALHO

Nos anos recentes tem ficado evidenciado a necessidade de utilização de novas tecnologias para o projeto e desenvolvimento de sistemas computacionais. A crescente necessidade das empresas e profissionais por novos e complexos sistemas demonstram explicitamente as limitações da utilização das técnicas convencionais para satisfazê-las.

A partir dos anos setenta, por outro lado, a tecnologia dos sistemas especialistas mostrava-se uma alternativa real. As técnicas de representação do conhecimento e os métodos de inferência foram melhorados e consolidados fazendo com que resultados positivos fossem obtidos nos trabalhos realizados nos centros de pesquisa e universidades.

Como conseqüência, as empresas e os profissionais de informática passaram a buscar a incorporação desta tecnologia para melhorar os seus sistemas computacionais. Entretanto, se por um lado é cada vez maior o interesse das empresas em utilizarem sistemas especialistas, existem todas as dificuldades relativas a incorporação de uma nova tecnologia. Estas dificuldades estão relacionadas, entre

outras, à falta de profissionais pois a sua maioria é treinada para o desenvolvimento de sistemas de processamento de dados convencionais. Os sistemas especialistas são, portanto, uma tecnologia da qual as empresas pretendem tirar proveito mas, de forma geral, ainda não dominam.

Assim, com o objetivo de apoiar a transferência da tecnologia dos sistemas especialistas do meio acadêmico para o meio profissional e empresarial, surgiu o interesse por um trabalho de pesquisa na área.

1.2 - OBJETIVOS

Os objetivos deste trabalho são:

- expor os principais aspectos relacionados à criação e utilização de sistemas especialistas.

- verificar as ferramentas de apoio disponíveis para a construção destes sistemas.

- avaliar a aplicabilidade da tecnologia como uma alternativa ao enfoque convencional de desenvolvimento de sistemas.

- aquilatar as potencialidades de uma linguagem de programação não procedural.

- explorar os recentes avanços tecnológicos verificados na área de sistemas especialistas.

1.3 - IMPORTÂNCIA DO TRABALHO

A importância do presente trabalho está intimamente ligado ao interesse demonstrado pelas empresas na incorporação da tecnologia dos sistemas especialistas pelos seus ambientes de desenvolvimento de sistemas.

Esta dissertação pretende apresentar-se como uma fonte de consulta para os profissionais interessados em conhecer os detalhes relacionados aos sistemas especialistas. Ela permite conhecer os elementos componentes destes sistemas, suas funções e características.

Este trabalho apresenta-se, ainda, como um instrumento de apoio aos trabalhos acadêmicos de pesquisa e desenvolvimento na área. Ele poderá contribuir para despertar o interesse em futuros estudos.

O presente trabalho será útil também para aqueles que pretendem conhecer os princípios da programação lógica. O protótipo, que é parte integrante da dissertação, foi inteiramente desenvolvido utilizando-se esta técnica de programação.

1.4 - METODOLOGIA

Visando atingir os objetivos citados anteriormente, foi realizada uma pesquisa teórica, a especificação e a implementação de um protótipo e, finalmente, a elaboração de uma aplicação experimental.

A pesquisa teórica abrangeu uma revisão da literatura especializada, tanto nacional quanto estrangeira. Foram pesquisados livros, revistas, artigos científicos, dissertações de mestrado e teses de doutorado.

Na especificação e implementação do protótipo foram observados as recomendações levantadas ao longo da pesquisa teórica, testados sistemas disponíveis comercialmente e, ainda, verificados materiais promocionais de softwares afins.

A aplicação experimental desenvolveu-se na área médica. Ela permite diagnosticar algumas doenças comuns do aparelho digestivo e respiratório. O trabalho de elaboração e introdução da base de conhecimentos no sistema foi executado com o apoio de um médico. Depois de construída a base de conhecimentos foram efetuadas sucessivas inferências para avaliar o comportamento do sistema.

1.5 - LIMITAÇÕES

O presente trabalho procurou ser o mais abrangente possível. Entretanto, por questões práticas, o sistema desenvolvido apresenta-se como um protótipo já que alguns aspectos computacionais, como segurança da base e credenciamento de usuários, foram suprimidos.

1.6 - ESTRUTURA

O capítulo 1 mostra os objetivos, o escopo, a estrutura e as limitações do trabalho.

O capítulo 2 apresenta alguns conceitos e definições, as características dos sistemas especialistas, seus componentes, o histórico da sua evolução e, ainda, as principais formas de representação do conhecimento.

Os capítulos 3, 4 e 5, referem-se aos sistemas especialistas que utilizam as regras de produção para a representação do conhecimento. O capítulo 3 detalha o funcionamento destes sistemas. São mostrados os requisitos, as ferramentas utilizadas na sua construção, os métodos de busca e encadeamento existentes e a representação da incerteza na base de conhecimentos.

O capítulo 4 apresenta a especificação e implementação do SISREP, o protótipo desenvolvido. São mostrados os módulos componentes do sistema, a

arquitetura dos arquivos e o formato das regras. O capítulo 5 é composto pelas orientações para utilização do sistema e pela aplicação experimental realizada.

O no capítulo 6 apresenta as conclusões do trabalho e as recomendações.

CAPÍTULO 2

2 - SISTEMAS ESPECIALISTAS

2.1 - O QUE SÃO SISTEMAS ESPECIALISTAS

Os sistemas especialistas são sistemas computacionais projetados e desenvolvidos para solucionarem problemas que normalmente exigem especialistas humanos com conhecimento na área de domínio da aplicação. Tal como um especialista o sistema deve ser capaz de emitir decisões justificadas acerca de um determinado assunto a partir de uma substancial base de conhecimentos. Para tomar uma decisão o especialista busca em sua memória conhecimentos prévios, formula hipóteses, verifica os fatos que encontra e compara-os com as informações já conhecidas e então emite a decisão. Neste processo o especialista realimenta a sua "base de conhecimentos" acerca do assunto.

Os sistemas especialistas, portanto, caracterizam-se por armazenar um vasto e profundo conhecimento acerca de uma área específica devendo armazenar estas informações de forma organizada permitindo simplificar a busca a respostas requeridas. Devem ainda possuir um mecanismo de raciocínio ou inferência que possibilite responder aos questionamentos, justificar suas conclusões e ainda ter capacidade para "aprender" novos conhecimentos.

Alguns autores apresentam definições formais de sistemas especialistas, Feigenbaum,[FEIGENBAUM,83] escreve:

"... um programa inteligente de computador que usa conhecimento e procedimentos inferenciais para resolver problemas que são bastante difíceis, de forma a requererem para a sua solução muita perícia humana."

Já Waterman,[WATERMAN,86] tem uma definição que explicita mais as propriedades destes sistemas ao afirmar:

" Um sistema especialista é aquele que tem regras especializadas e não faz busca exaustiva desempenhando bem, raciocina através da manipulação de símbolos, contém os princípios fundamentais do domínio do conhecimento, possui métodos de raciocínio que podem ser usados quando as regras do especialista falham e podem ser usados para produzir explicações. Trata com problemas difíceis em um domínio complexo."

Ribeiro,[RIBEIRO,87] escreve a respeito dos sistemas especialistas:

"Um sistema especialista deve, além de inferir conclusões, ter capacidade de aprender novos conhecimentos e, desse modo, melhorar o seu desempenho de raciocínio e a qualidade de suas decisões.

Já Rabuske [RABUSKE,95], afirma:

"Sistemas especialistas devem, também, ter habilidade para aprender com a experiência e explicar o que estão fazendo e porque o fazem. Esta última é uma das principais características que distinguem estes sistemas dos tradicionais sistemas de informação".

2.2 - CARACTERÍSTICAS PRÓPRIAS

Os sistemas especialistas de forma geral tentam modelar o comportamento do raciocínio humano na tentativa de resolver problemas. Por isso, a abordagem do sistema especialista introduz novas idéias que a programação tradicional não usa.

Os sistemas especialistas caracterizam-se pela manipulação de informações composta por fatos a respeito do assunto e regras formais que descrevem relações. Estas informações compõem a chamada base de conhecimentos sobre a qual será feito o processamento. O sistema processa, portanto, o conhecimento não existindo o processamento de dados, típico de sistemas convencionais.

Outra característica própria dos sistemas especialistas é a utilização de técnicas de inferência para manipular informações visando uma solução. O mecanismo de inferência utiliza estratégias genéricas para adquirir conhecimento, processá-lo, tirar conclusões e dar explicações a respeito do processo de raciocínio. Esta abordagem baseada em conhecimento oferece a possibilidade de separar o conhecimento que descreve o domínio do problema do código de procedimentos que examina esse conhecimento. Este mecanismo dos sistemas especialistas distingue-os de programas tradicionais.

2.3 - ABORDAGEM HISTÓRICA

No início da década de 1960 começaram os primeiros trabalhos nos sistemas que hoje são chamados de especialistas. Inicialmente pretendia-se construir máquinas inteligentes com grande poder de raciocínio e solução de problemas. Imaginava-se que a partir de um pequeno conjunto de normas ou regras de raciocínio introduzidas num poderoso computador criariam-se sistemas de capacidade superior a humana. Não tardou para que os pesquisadores observassem o engano e verificassem as reais dimensões do trabalho.

Em 1964 foi construído o DENDRAL, por Joshua Lederberg da Universidade de Stanford. O DENDRAL a partir de um determinado conjunto de dados como massa espectrográfica e ressonância magnética, deduz a possível estrutura de um determinado composto químico. Este programa era do tipo algorítmico. Em 1965, Joshua juntou-se a Edward Feigenbaum e Bruce Buchanan para tentar construir um programa que não fosse algorítmico e que usasse regras heurísticas para resolver os mesmos problemas do DENDRAL. Este novo DENDRAL mostrou a viabilidade dos sistemas especialistas e levou pesquisadores de outras universidades a trabalharem no assunto.

Em 1968, surge no MIT - Massachusetts Institute of Technology, o MACSYMA, destinado a auxiliar matemáticos na resolução de problemas complexos. O programa foi originalmente elaborado por Carol Engleman, William Martin e Joel Mores. MACSYMA é um sistema ainda hoje amplamente utilizado em universidades e laboratórios de pesquisa.

Posteriormente, já na década de 1970, surgiram importantes e complexos sistemas especialistas entre os quais o MYCIN e o PROSPECTOR. O MYCIN é um sistema na área médica para detectar e diagnosticar doenças infecciosas. O objetivo principal do sistema é recomendar uma terapia para o paciente encontrando primeiro a causa da doença com base em observações clínicas que são informadas pelo usuário e então avaliadas utilizando a sua base interna de conhecimentos. O PROSPECTOR é um sistema para dar suporte a geólogos na exploração mineral. O objetivo principal do sistema é apoiar um geólogo que esteja nas fases iniciais de uma exploração ou prospecção. O usuário informa as principais características da prospecção, como tipo de rocha, minerais etc, e o programa usa as informações para tentar satisfazer os seus modelos internos para chegar a conclusões relacionadas a existência e viabilidade de exploração de certos tipos de minerais.

A década de 1980 foi marcada pelo grande crescimento de aplicações, inclusive, com larga disponibilização de produtos comerciais no mercado de software. Este acelerado processo de desenvolvimento de aplicações deve-se em parte ao avanço dos recursos de equipamentos, ou hardware, ocorrida paralelamente

neste período. Entre os produtos surgidos nesta década podem ser destacados os das áreas médica e matemática.

No Brasil, a Pontifícia Universidade Católica do Rio de Janeiro desenvolveu, nos anos 80, importantes trabalhos com sistemas especialistas. O principal resultado da universidade nesta área é um sistema chamado SAFO cuja finalidade é a demonstração de teoremas matemáticos. Outra referência histórica no Brasil, é o Instituto Militar de Engenharia que há alguns anos vem desenvolvendo sistemas de recuperação em grandes bases de conhecimentos. Quanto aos produtos comerciais brasileiros vale citar o surgimento, também nos anos 80, do sistema PATER. Este produto é um software para a construção de sistemas especialistas de caráter geral e foi projetado para utilização em microcomputadores.

2.4 - COMPONENTES DE UM SISTEMA ESPECIALISTA

A composição de um sistema especialista depende de fatores como a generalidade pretendida, os objetivos do mesmo, a representação interna do conhecimento e as ferramentas usadas na implementação. Entretanto, o modelo geral de arquitetura apresentada por um grande número de autores é mostrado na figura 1. Também na terminologia empregada há diferenças entre os autores, mas de uma forma geral o sistema é constituído de seis elementos básicos que são: base de conhecimentos, mecanismo de aprendizagem e aquisição de conhecimento, máquina ou motor de inferência, sistema de justificação, sistema de consulta e quadro negro.

2.4.1 - BASE DE CONHECIMENTOS

A base de conhecimentos é o local onde se armazenam fatos, heurísticas, crenças etc, ou seja, é um depósito de conhecimentos acerca de um determinado assunto. Este conhecimento é passado ao sistema pelo especialista e armazenado de uma forma própria que permite ao sistema fazer posteriormente o processamento ou inferências.

A qualidade deste conhecimento armazenado é o fator determinante no potencial do sistema especialista. O sistema deve possuir uma base de conhecimentos o mais flexível possível de ser atualizada, e também poder fazer aprendizagem direta durante as consultas.

A fase de construção da base de conhecimentos é uma das mais complexas na implementação de um sistema especialista pois o conhecimento de um especialista não se encontra formalizado, precisando portanto de um trabalho prévio para tal. A base de conhecimentos está interligada com quase todos os demais elementos do sistema, especialmente com a máquina de inferência, o mecanismo de aprendizagem e aquisição do conhecimento e o quadro negro.

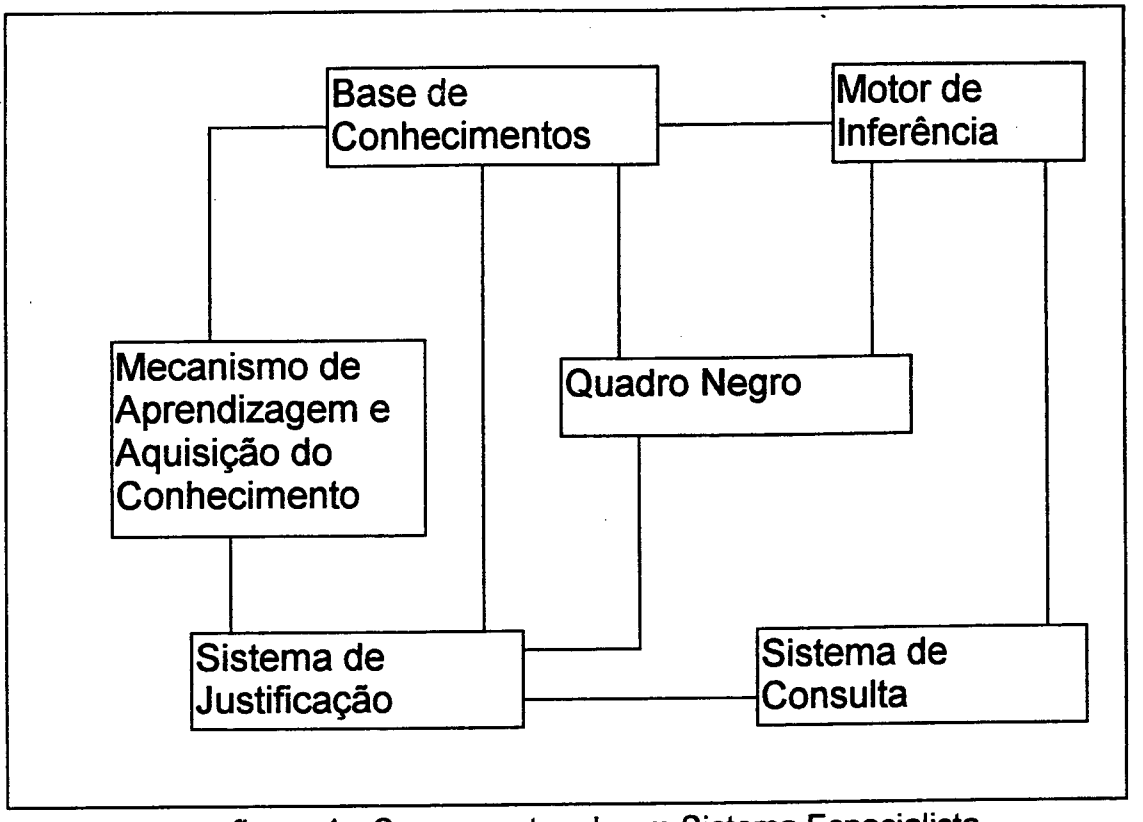

figura 1 - Componentes de um Sistema Especialista

2.4.2- MECANISMOS DE APRENDIZAGEM E AQUISIÇÃO DE CONHECIMENTO

Os sistemas especialistas devem possuir meios que permitam ampliar, alterar ou atualizar o seu conhecimento. Geralmente existe um módulo no sistema que se utiliza de recursos como editores de textos próprios ou não, classificadores etc. que permitem adequar ou formatar o conhecimento para ser introduzido na base de conhecimentos.

A tarefa de extrair o conhecimento e utilizá-lo adequadamente é bastante complexa. Este aspecto tem sido motivo de intensas pesquisas visando a simplificação e otimização deste processo. Segundo Rabuske [RABUSKE,95] a aquisição do conhecimento tende a caracterizar áreas de pesquisa específicas nas universidades e nos centros de pesquisa, geralmente ligadas à engenharia do conhecimento. Ainda segundo Rabuske, obter o conhecimento é a parte mais crítica

da construção de um sistema especialista. Já Rich [RICH,93] afirma que o processo convencional, ou seja, entrevistas com o especialista formalização e introdução do conhecimento na base, é caro e lento e que vale a pena procurar maneiras mais automatizadas de construir bases de conhecimentos. Afirma ainda Rich que embora já existam muitos programas úteis que interagem com os especialistas para extrair conhecimento especializado com eficiência, ainda não existe nenhum sistema de aquisição de conhecimento totalmente automatizado.

2.4.3 - MOTOR OU MÁQUINA DE INFERÊNCIA

As informações armazenadas numa base de conhecimentos são, evidentemente, estáticas até que uma força externa analise e processe este conhecimento para dele tirar proveito. Este mecanismo que permite "tirar o proveito" de uma base de conhecimentos é chamado de motor, máquina, ou ainda, engenho de inferência. O motor de inferência é o elemento do sistema especialista que é capaz de buscar na base o conhecimento necessário a ser avaliado em cada situação, direcionar o processo de raciocínio, gerenciar situações de incerteza e levar ao resultado final.

O processo de inferência está diretamente associado com a estrutura utilizada para o armazenamento do conhecimento na base de conhecimentos. Entretanto, de forma geral, pode-se afirmar que o processo envolve um encadeamento lógico que permita tirar conclusões a partir do conhecimento existente. O motor de inferência é, portanto, o responsável pela ação repetitiva de buscar, analisar e gerar novos conhecimentos.

Este elemento não é, geralmente, um módulo único no sistema. Pois na verdade estão envolvidos vários processos que devem trabalhar de forma conjunta, tais como busca do conhecimento relevante num determinado contexto, resolução de conflitos, gerenciamento de incertezas e finalmente a execução propriamente dita.

Um motor de inferência para base de conhecimentos com a utilização de regras de produção será mostrado detalhadamente no capítulo seguinte deste trabalho e sua implementação é apresentada no capítulo quatro.

2.4.4 - SISTEMA DE CONSULTA

Os usuários de sistemas especialistas interagem de forma intensa com o sistema pois além de receberem dele as conclusões alcançadas também participam ativamente do processo de inferência e da construção da base de conhecimentos. Este sistemas devem portanto oferecer bons recursos de comunicação que permitam, até ao usuário sem conhecimentos computacionais, tirar proveito dos mesmos.

Aspectos internos dos sistemas, terminologia computacional etc. devem ser evitados e detalhes técnicos relativos a implementação devem ser transparentes ao usuário. A linguagem a ser utilizada deve ser orientada para o problema ou para a área do especialista e o mais perto possível da linguagem natural.

A maioria dos sistemas existentes usam técnicas simples de interação com o usuário, quase sempre utilizando perguntas já pré-formatadas e respostas tipo múltipla escolha. Outra técnica é a definição de uma gramática sintética simples com um vocabulário restrito e limitado, própria para utilização no sistema. Recentemente, entretanto, intensas pesquisas tem sido feitas no sentido de tornar o computador capaz de entender a linguagem natural humana. Esta tecnologia é todavia um outro campo de estudo da inteligência artificial cujo desenvolvimento será de extrema valia para toda a área da computação.

2.4.5 - SISTEMA DE JUSTIFICAÇÃO

O módulo de justificação, também chamado por alguns autores de explanação, tem a função de esclarecer o usuário a respeito de uma conclusão apresentada pelo sistema ou ainda explicar por que uma pergunta está sendo feita. Ele é na verdade um recurso de questionamentos fornecido ao usuário.

A justificação é um requisito importante dos sistemas especialistas. Em muitos dos domínios nos quais os sistemas operam, as pessoas não aceitam resultados se estes não estiverem devidamente justificados. Na medicina, por exemplo, onde um médico tem a responsabilidade final por um diagnóstico, certamente um sistema teria que mostrar os motivos que o levaram a alcançar uma determinada conclusão. De forma geral, os sistemas são implementados para responder as seguintes perguntas:

Como chegou a esta conclusão ?

Por que chegou a esta conclusão ?

Por que não chegou a tal conclusão ?

2.4.6 - QUADRO NEGRO

O quadro negro, ou rascunho, é uma área de trabalho que o sistema utiliza durante o processo de inferência. Nesta área são armazenadas informações de apoio e suporte ao funcionamento do sistema quando este está raciocinando. Embora todos os sistemas especialistas usem o quadro negro, nem todos o explicitam como componente do sistema.

2.5 - REPRESENTAÇÃO DO CONHECIMENTO

2.5.1 - NECESSIDADE DE REPRESENTAR O CONHECIMENTO

Para que um sistema especialista possa resolver problemas é imprescindível que esteja associado a ele um razoável volume de conhecimentos relativos ao domínio do problema. Este conhecimento, por sua vez, deve ser transformado em organizadas estruturas de dados que permitam a sua utilização pelo computador e ao mesmo tempo sejam facilmente administradas pelo especialista e usuário do sistema.

Segundo Winston [WINSTON,87] alguns aspectos são determinantes na qualidade da representação do conhecimento :

- explicitar as coisas importantes
- revelar restrições naturais
- ser completa
- ser concisa
- ser clara ou transparente, podendo-se entender o que quer dizer
- facilitar o armazenamento e recuperação
- permitir manter o conhecimento coerente, utilizando recursos para proteger o acesso.

De forma geral, pode-se afirmar que representa-se o conhecimento para posteriormente recuperá-lo, para raciocinar com ele e para adquirir mais conhecimento. A representação do conhecimento é a formalização do conhecimento do sistema. Para que isto seja possível existem técnicas que permitem modelar o conhecimento de forma eficiente e deixá-lo pronto para ser acessado facilmente.

2.5.2- FORMAS DE REPRESENTAÇÃO DO CONHECIMENTO

2.5.2.1 - REDES SEMÂNTICAS

Redes semânticas são estruturas formadas por nós, interconectados através de arcos rotulados. Os nós representam objetos, conceitos, situações ou ações enquanto os arcos representam as relações entre os nós. Para ilustrar o assunto, considere o seguinte texto:

A dissertação é um requisito do curso de mestrado. Roberto, que é um estudante de mestrado, tem um bom computador que ele usa para escrever a dissertação.

A figura 2 mostra uma forma de representação do texto usando uma rede semântica.

As redes semânticas permitem qualquer tipo de ligação entre os nós desde que estas consigam transmitir o que significam. As ligações mais comuns, no entanto, são "é-parte-de" e "é-um". As ligações podem ser divididas em quatro tipos:

- tipo propriedade, relaciona dois nós para demonstrar uma propriedade. Na figura 2 a ligação entre "computador" e "bom" é deste tipo
- tipo subparte, indica que um nó é uma subparte ou componente de outro
- tipo subclasse, exprime esta característica entre os nós envolvidos. A dissertação (figura 2) poderia ter sido apresentada como uma subclasse em relação a um nó pesquisa
- tipo relacionamento, indica que, de alguma forma, uma relação existe entre os nós. Na figura 2 a ligação entre "escrever" e "dissertação" é um exemplo desta relação.

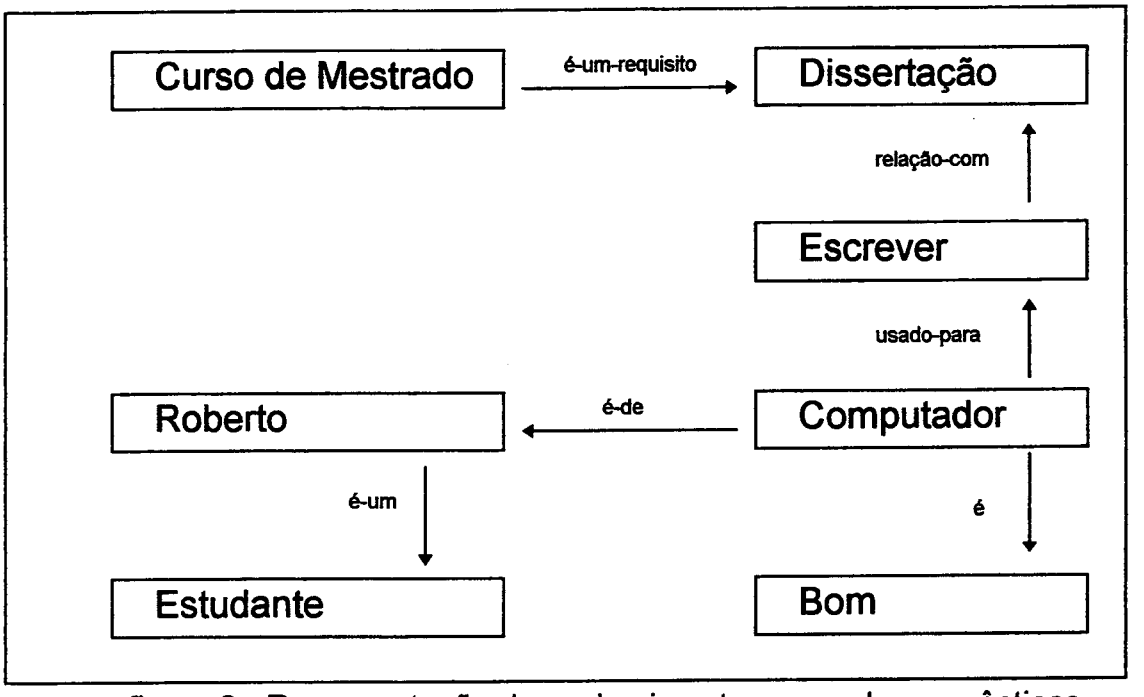

figura 2 - Representação do conhecimento com redes semânticas

Evidentemente, para se representar um grande volume de informações, uma rede semântica pode tornar-se bastante complexa e de difícil representação gráfica. Em geral se utiliza uma estrutura de dados adequada para representá-la além de uma linguagem de programação apropriada para utilizar esta forma de representação do conhecimento.

2.5.2.2 - QUADROS

A idéia de representar o conhecimento através de quadros foi apresentada inicialmente por Marvin Minsky [MINSKY,74] como uma forma de descrever estruturadamente experiências anteriores. Os quadros dizem respeito ao uso de conhecimento anterior ou experiências já vividas para interpretar novas situações. Conforme [MARCUS,86] o modelo de quadros, ou frames, baseia-se no processo humano de resolver muitos tipos de problemas através do agrupamento de informações afins. Segundo ela o homem classifica as informações que recebe de forma a armazenar em conjuntos as que estiverem relacionadas

Os quadros, ou frames, são estruturas de preenchimento que descrevem uma entidade real ou imaginária. Um quadro é constituído por um nome, uma coleção de atributos, chamados de escaninhos ou slots e valores a eles associados. O exemplo abaixo ilustra o modelo de quadros.

Quadro: Automóvel

Slot: tipo - chevette, fusca, ômega, quantum (default: fusca)

Slot: número-de-rodas - um inteiro (default: 4)

Slot: número-de-lugares - um inteiro (default: 5)

Slot: combustível - álcool, gasolina, diesel (default: gasolina)

Slot: cor - branco, verde, azul, vermelho (default: azul)

Slot: número-de-portas - 2, 3, 4 (default: 2)

Slot: teto - existente, inexistente (default: existente)

Slot: modelo - utilitário, coupê, perua (default: coupê)

Quadro: Automóvel-da-Ivete

Slot: tipo - chevette

Slot: número-de-rodas - 4

Slot: número-de-lugares - 5

Slot: combustível - álcool

Slot: cor - branco

Slot: número-de-portas - 4

Slot: teto - existente

O quadro Automóvel-da-Ivete tem os mesmos slots que o quadro genérico, constituindo-se assim uma instância do mesmo. O slot modelo, por não ter um valor explicitado assume o valor default que é coupê.

O quadro descreve uma entidade em termos absolutos ou de um determinado ponto de vista. Os sistemas são criados a partir de coleções de frames; um quadro sozinho não tem utilidade. Cada quadro representa uma classe (um conjunto), já uma instância do quadro é um elemento deste conjunto. As instâncias surgem a medida que são incluídas as informações ou o conhecimento no sistema.

A flexibilidade é uma característica marcante da representação do conhecimento por quadros já que não existe nenhuma restrição quanto ao formato e tipo dos quadros. Esta flexibilidade na utilização de frames merece inclusive alguns cuidados, conforme [KELLER,91]:

"Certamente, é preciso bastante cuidado ao se projetar um sistema frame. Eles oferecem tanta flexibilidade que você pode se imaginar mais uma vez caindo em uma armadilha", onde tudo é possível e nada é fácil. Entretanto, o problema é da mesma ordem de magnitude de qualquer tarefa de modelagem de informação e de análise. Nós somos tanto auxiliados como enganados neste processo pela natureza da informação simbólica".

2.5.2.3 - LÓGICA DAS PROPOSIÇÕES E DOS PREDICADOS

A criação da lógica clássica é creditada a Aristóteles (384-322 A.C.). A lógica criada por ele trabalhava essencialmente com a veracidade de argumentos. Num desenvolvimento posterior, usando símbolos, nasceu a lógica booleana descoberta por George Bool. A lógica simbólica tem dois ramos distintos, a lógica proposicional que se preocupa com a veracidade ou falsidade de proposições e a lógica dos predicados que inclui relações entre objetos e classes.

Na lógica das proposições, será atribuído o valor lógico verdadeiro se as informações disponíveis permitirem tirar esta conclusão a respeito de uma proposição; caso contrário é atribuído o valor falso. Para se trabalhar com várias proposições utiliza-se operadores de conexão para assim obter as chamadas proposições compostas e aumentar a capacidade de expressão. Estes operadores são AND, OR, NOT, IMPLIES, EQUIVALENT.

Pode-se facilmente representar fatos do mundo real usando a lógica proposicional, entretanto, isto não é suficiente para fazer dela uma forma eficiente de representação do conhecimento em sistemas especialistas pois são poucos os problemas que se resumem ao falso e verdadeiro suportados por ela. Desta limitação, surge a opção da lógica dos predicados.

A lógica dos predicados, ou cálculo dos predicados, é uma extensão da lógica proposicional. Ela introduziu funções, termos, quantificadores e predicados trazendo assim maior capacidade de expressão. A lógica dos predicados também permite que as expressões tenham variáveis. Com as variáveis consegue-se generalizar declarações sobre classes ou entidades. Poderia-se, por exemplo, afirmar que para todos os valores de X, onde X é um dia da semana, a declaração *clima(X,chuva)* seja verdadeira, ou seja, chove todos os dias da semana.

Na criação dos predicados que devem constituir o conhecimento do sistema, são usados identificadores para representar objetos, propriedades ou relações que podem ser livremente criados. Assim, *gosta(maria,joão)* poderia descrever uma relação (gostar) entre dois objetos (maria e joão). Cada predicado, por sua vez, tem associado uma aridade que é o número de elementos ou argumentos utilizados numa relação. Por exemplo, as relações *pai(josé,joão)*, *soma(2,3)*, *irmão(caim,abel)* são predicados de aridade dois, enquanto *cidade(florianópolis)* tem aridade um.

Para exemplificar a representação do conhecimento utilizando a lógica dos predicados é mostrado abaixo um conjunto de relações familiares:

- *Eva é mãe de Caim e Abel*
- *Adão é o pai de Caim e Abel*
- *Pais de uma pessoa é o termo usado para dizer que uma pessoa é pai ou mãe de outra pessoa*
- *Duas pessoas são irmãs quando tem o mesmo pai ou a mesma mãe*

Estas informações podem facilmente ser representadas pela uso da lógica dos predicados:

mãe(eva,caim)

mãe(eva,abel)

pai(adão,caim)

pai(adão,abel)

$\forall$ X $\forall$ Y pai(X,Y) $\vee$ mãe(X,Y) $\Rightarrow$ pais(X,Y)

$\forall$ X $\forall$ Y $\forall$ Z pais(X,Y) $\wedge$ pais(X,Z) $\Rightarrow$ irmãos(Y,Z)

Um segundo exemplo é mostrado abaixo para descrever as seguintes afirmações:

- *Senna era um piloto*
- *Senna nasceu em São Paulo*
- *Todos os que nascem em São Paulo são brasileiros*
- *Todos os brasileiros gostavam de Senna ou odiavam-no*

A representação destas sentenças utilizando-se a lógica dos predicados é:

piloto(senna)

paulista(senna)

$\forall$ X paulista(X) $\Rightarrow$ brasileiro(X)

$\forall$ X brasileiro(X) $\Rightarrow$ gostava(X,senna) $\vee$ odiava(X,senna)

Esta forma de representação do conhecimento é uma ferramenta poderosa pois possui uma notação simples capaz, porém, de traduzir em sentenças de sintaxe e semântica bem definidas as situações da vida cotidiana. Entre os aspectos negativos da lógica dos predicados está a impossibilidade de representar os tempos de ocorrência dos fatos quando incrementos na representação precisam ser feitos.

2.5.2.4 - REGRAS DE PRODUÇÃO

A representação do conhecimento por regras de produção é baseada nas propostas concebidas pelo matemático Emil Post (1943) que via nos sistemas de produção um modelo computacional geral de solução de problemas. Na década de oitenta passou-se a utilizar esta técnica como suporte para o modelo mental.

O termo "sistema de produção" é atualmente usado para descrever os sistemas que taiêm em comum o fato de serem constituídos de um conjunto de regras para descrever condições e ações. As regras são armazenadas como uma coleção de declarações SE-ENTÃO.

SE < premissas > ENTÃO <conclusões>

A parte SE da regra é chamada de corpo, parte antecedente ou lado esquerdo e deve ser avaliada em relação à base de conhecimentos como um todo. Quando existe o ajuste buscado pelo mecanismo de avaliação a ação correspondente especificada no lado direito, ou parte conseqüente, é executada. As condições na parte antecedente da regra devem ser satisfeitas para que a ação, na parte conseqüente, seja considerada. Se qualquer premissa falhar o lado direito também falha.

A representação do conhecimento por regras de produção é a forma mais utilizada em sistemas especialistas. A justificativa é a naturalidade que representa para o homem pois o par "condição-ação", para raciocinar e decidir, também é usado pela mente humana. Estima-se que cerca de oitenta por cento dos sistemas existentes utilizam esta forma de representação do conhecimento.

Como ilustração ao uso de regras de produção são mostradas a seguir algumas regras que poderiam fazer parte, por exemplo, de um sistema especialista para orientação de candidatos ao curso de mestrado.

Regra:

SE um candidato se inscrever para o curso de mestrado
e o candidato preencher todos os requisitos exigidos
e existirem vagas disponíveis
ENTÃO o candidato passa a ser aluno do curso de mestrado

Regra:

SE o aluno de mestrado for aprovado em todas as disciplinas do currículo
ENTÃO ele deve escrever uma dissertação

Regra:

SE a dissertação está concluída será submetida a uma banca examinadora
e a banca examinadora aprovar o trabalho
ENTÃO o aluno receberá o grau de mestre

Além da naturalidade para a interpretação humana a utilização de regras de produção apresenta outros aspectos positivos, como a modularidade e a uniformidade. As regras podem ser manipuladas como peças independentes e novas regras podem ser incluídas a qualquer tempo o que é uma característica importante pois o conhecimento de qualquer sistema especialista tende a aumentar com o passar do tempo. A uniformidade fica caracterizada no padrão único utilizado para todas as regras do sistema. Esta uniformidade pode gerar alguns inconvenientes para o

usuário mas também traz muitos benefícios como facilidade de manutenção e de uso do sistema por pessoas não familiarizadas com o mesmo.

CAPÍTULO 3

3 - SISTEMAS BASEADOS EM REGRAS DE PRODUÇÃO

3.1 - FUNCIONAMENTO

Num sistema baseado em regras de produção o conhecimento a ser processado é representado através do uso de regras com uma arquitetura previamente definida. Estas regras utilizam um par condição-ação onde as condições são premissas e a ação é a conclusão.

No processo de inferência o sistema busca uma primeira regra arbitrariamente, ou em alguns casos aquela predefinida como regra inicial, e tenta atender as premissas da regra. As premissas descritas na regra são apresentadas ao usuário em forma de questionamentos. As respostas fornecidas pelo usuário são então armazenadas na lista de verdades fazendo com que estas informações fiquem disponíveis ao longo do processo de raciocínio e possam ser utilizadas para a validação de outras regras. Se as respostas fornecidas pelo usuário atenderem as premissas da regra e a regra contiver na sua parte conclusiva uma solução para o problema o processo de inferência estará concluído com sucesso.

Se, por outro lado, a regra não permitiu alcançar uma solução para o problema, o sistema seguirá avaliando outras regras, sempre acumulando o conhecimento adquirido ao longo do processo na sua lista de verdades. O processo continua até que seja alcançada uma regra que leve à solução do problema, ou quando não for mais possível continuá-lo.

Um exemplo de um pequeno conjunto de regras para diagnóstico de problemas em veículos é mostrado abaixo.

Regra 1

SE tem combustível no tanque
E tem combustível no carburador
ENTÃO o motor recebe combustível

Regra 2

SE o motor recebe combustível
E o motor vira
ENTÃO o problema é nas velas

Regra 3

SE o motor não vira
E as lâmpadas não ascendem
ENTÃO o problema é na bateria ou nos cabos

Regra 4

SE o motor não vira
E as lâmpadas ascendem
ENTÃO o problema é o motor de partida

Observa-se no exemplo apresentado que existe um encadeamento lógico entre as regras. Esta rede de encadeamentos é chamada de árvore de busca. O raciocínio com regras de produção envolve, portanto, a aplicação de um algoritmo para fazer a busca dos possíveis caminhos da árvore. Este algoritmo, por sua vez, deve oferecer recursos para que o usuário possa optar por estratégias diferenciadas de raciocínio ou encadeamento.

3.2 - RACIOCÍNIO E ENCADEAMENTO

3.2.1 - ENCADEAMENTO PROGRESSIVO

O método de encadeamento progressivo, também chamado de encadeamento-para-frente ou "forward", começa com a seleção de uma regra qualquer da base de conhecimentos. Simultaneamente é criada a lista de verdades onde serão armazenadas todas as informações tidas como verdadeiras. No início do processo de inferência alguns questionamentos ao usuário serão obrigatórios já que a lista de verdades está ainda vazia.

As respostas do usuário são então colocadas na lista de verdades. Após a utilização de uma regra, as conclusões desta regra também devem ser adicionadas à lista de verdades. Na seqüência, o sistema procura por uma regra que tenha entre as

suas premissas a conclusão da regra avaliada. Este encadeamento que vai da conclusão de uma regra para as premissas de outra regra é chamado de encadeamento progressivo.

Utilizando-se o exemplo do diagnóstico de problemas em veículos apresentado anteriormente, o sistema avaliaria inicialmente a regra 1. As premissas da regra 1 são *"a existência de combustível no tanque"* e *"a existência de combustível no carburador"*. Como neste estágio a lista de verdades ainda se encontra vazia e não existe outra regra na base de conhecimentos que permita verificar a veracidade destas premissas o sistema questionará diretamente o usuário a respeito delas.

As respostas do usuário às premissas da regra 1 seriam então colocadas na lista de verdades. Caso estas respostas confirmem as premissas, também a conclusão da regra 1, ou seja *"o motor recebe combustível"*, seria introduzida na lista de verdades e o sistema buscaria uma outra regra que tivesse entre as suas premissas a conclusão da regra 1.

A regra 2 seria selecionada e o sistema tentaria provar as suas premissas. A primeira delas, *"o motor recebe combustível"* seria atendida imediatamente pois esta informação já está na lista de verdades. A segunda premissa, *"o motor vira"*, não pode ser comprovada pois não se tem nenhuma informação acerca desta premissa na lista de verdades. A questão deve ser então apresentada ao usuário. Se a resposta atender a premissa, a busca terminaria com sucesso com o sistema concluindo que o problema está nas velas.. Se, por outro lado, a resposta fosse contrária, o sistema passaria a verificar a condição *"as lâmpadas não ascendem"* que é a segunda premissa da regra 3.

Novamente não é possível buscar a informação na lista de verdades nem através de outra regra, devendo ser apresentada a questão ao usuário. Se a resposta atender a condição, o processo de inferência estaria encerrado com o sistema concluindo que o problema é nas velas. Se a resposta não atender a premissa, a regra 4 passaria a ser avaliada.

A regra 4 pode ser automaticamente avaliada pelo sistema, pois as informações relacionadas a todas as suas premissas já estariam na lista de verdades. O processo teria continuidade até que alguma conclusão fosse alcançada, ou não existissem mais regras a serem avaliadas.

Alguns aspectos computacionais, que foram ignorados no exemplo, teriam que ser observados em caso de implementação. Um sistema real deve considerar a existência de caminhos alternativos ao longo do processamento, inclusive com um mecanismo que possibilite retorno ou retrocesso, chamado de "backtracking", quando necessário. Neste caso as regras 2 e 3 também podem levar a uma solução; o sistema deve, por isto, oferecer ao usuário a alternativa de buscar outras soluções além daquela alcançada e deve também sustentar as conclusões através de recursos de justificação.

3.2.2 - ENCADEAMENTO REGRESSIVO

O encadeamento regressivo, também chamado de "backward" ou para-trás, difere do encadeamento progressivo pelo fato de que inicia numa conclusão e passa então a usar as regras para provar esta conclusão. Para provar que a conclusão é verdadeira, o sistema verifica se as premissas da regra são verdadeiras. Para provar a veracidade de uma premissa, o método procura por outra regra que tenha esta premissa como sua conclusão. Quando a regra é encontrada, procura-se prová-la demonstrando a veracidade de cada uma de suas premissas. Na seqüência , novamente o sistema procura por uma regra cuja conclusão atenda a premissa da regra em avaliação. O processo termina quando não mais existirem regras a serem provadas.

Também é criada, neste método, a lista de verdades que conterá todas as informações tidas como verdadeiras. Esta lista, além das respostas fornecidas diretamente pelo usuário, engloba também as conclusões do próprio sistema.

No caso do exemplo, o sistema teria que, inicialmente, optar entre as regras 2, 3 ou 4, já que todas elas levam a uma conclusão ou solução do problema. Se o sistema resolver o conflito, por exemplo, em favor daquela de menor número, a regra 2 passaria a ser avaliada.

Neste caso, o sistema assume que o problema é nas velas e tentará provar as premissas da regra 2, que são *"o motor recebe combustível"* e *"o motor vira"*. O problema é então decomposto em dois subproblemas, cada um deles relativo a uma das premissas. Como a primeira premissa desta regra é a conclusão da regra 1, o sistema passa a considerar esta regra. Neste momento existem três condições a serem confirmadas: *"tem combustível no tanque"*, *"tem combustível no carburador"* e *"o motor vira"*. Nenhuma das três condições está contida como parte conclusiva de outra regra, fazendo com que o sistema questione diretamente o usuário a respeito delas.

As respostas são então colocadas na lista de verdades. Se todas as respostas confirmarem as premissas, o processamento termina com sucesso e o sistema responderia que o problema está nas velas.

Se, no entanto, as respostas não atenderem as premissas, as regras 1 e 2 seriam desprezadas e o sistema passaria a considerar a regra 3, assumindo que o problema é na bateria ou nos cabos, e tentará provar as premissas correspondentes. O processo continua até que se consiga chegar a uma conclusão ou não restem mais regras para serem avaliadas. A figura 3 mostra a árvore de busca para o encadeamento regressivo.

O encadeamento regressivo mostra-se mais apropriado nas situações em que se tem uma conclusão preliminar que precisa ser confirmada. A seqüência de avaliação das regras neste caso parecerá mais lógica ao usuário pois os questionamentos apresentados estarão concentrados no objetivo que se quer provar.

As regras que foram utilizadas como exemplo na demonstração dos mecanismos de raciocínio progressivo e regressivo têm um encadeamento bastante simples e direto. As conexões existentes entre as regras são sempre únicas, fazendo com que as opções de pesquisa na árvore sejam reduzidas, facilmente visualizadas e não exigindo mecanismos sofisticados para determinar a seqüência de avaliação das regras. Entretanto, numa situação real, a árvore de inferência cresce em complexidade, especialmente no que se refere às conexões entre regras, exigindo do algoritmo de busca, além da especificação da direção, a determinação da ordem em que estas conexões, ou caminhos, serão examinados.

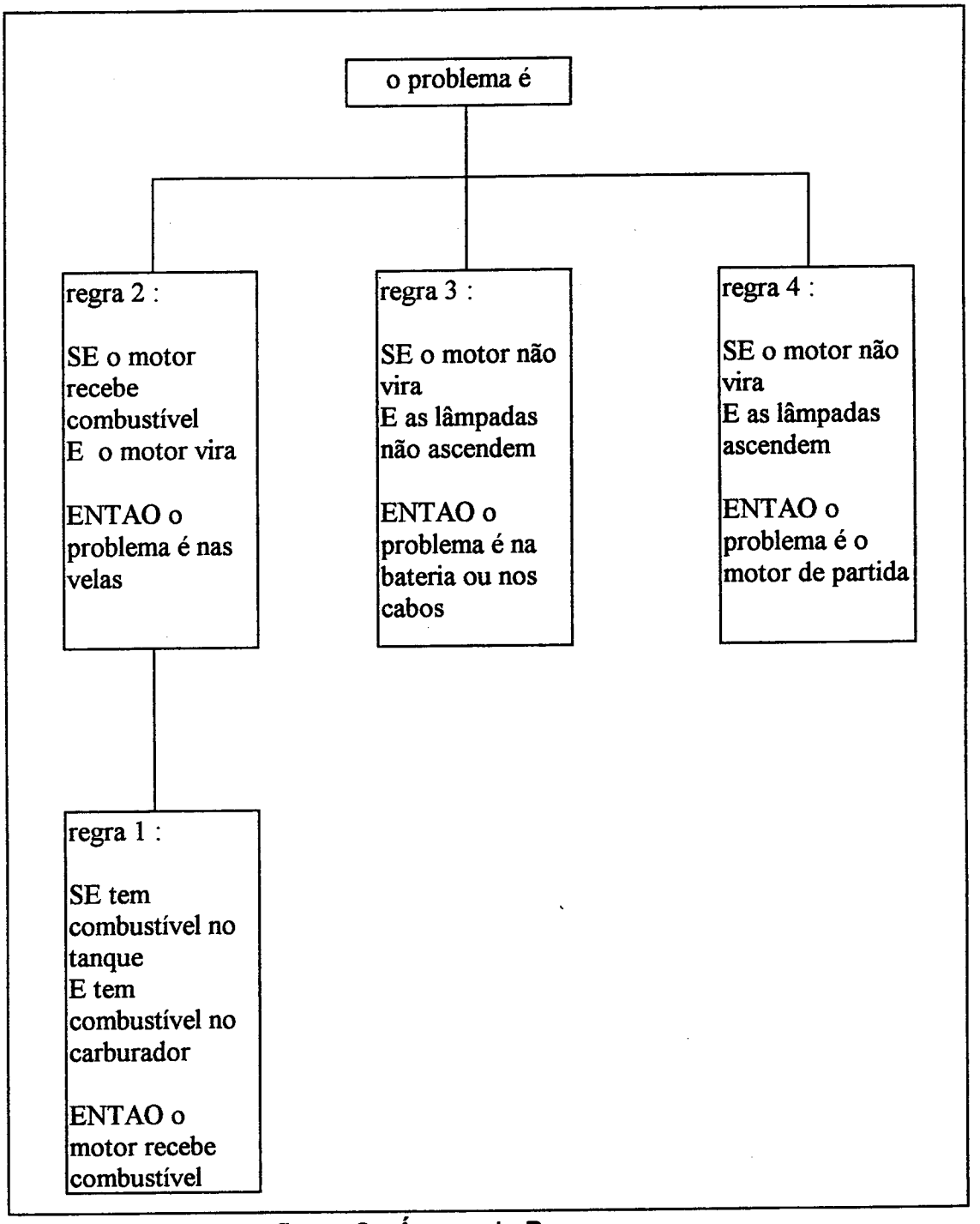

figura 3 - Árvore de Busca

3.3 - MÉTODOS BÁSICOS DE BUSCA

Um método de busca define os critérios da ordem em que as regras devem ser avaliadas no processo de raciocínio do sistema. A figura 4 mostra graficamente a base de conhecimentos de um sistema hipotético que será utilizado como exemplo no funcionamento dos métodos de busca, que serão apresentados na seqüência.

Os pontos, ou nós, representam as premissas de uma regra enquanto as setas, ou arcos, mostram a existência de conexão entre duas regras. O número associado a cada ponto representa o número da regra.

3.3.1 - BUSCA EM PROFUNDIDADE

Este método, também chamado de primeiro-em-profundidade ou ainda "depth-first", consiste em explorar preferencialmente os "descendentes" de um nó que tenha sido avaliado em prejuízo dos "vizinhos" ou "irmãos" deste nó. Na árvore da figura 4, por exemplo, se o nó 2 tivesse sido avaliado, e fosse dado prioridade aos descendentes mais à esquerda, o algoritmo passaria a considerar os nós 5, 11 e 19. Na seqüência seria priorizada a sub-árvore composta pelos nós 6, 12 e 20 e depois a sub-árvore 6 e 13. Somente depois de explorar todas estas sub-árvores é que o nó 3, que é irmão do nó 2, seria considerado.

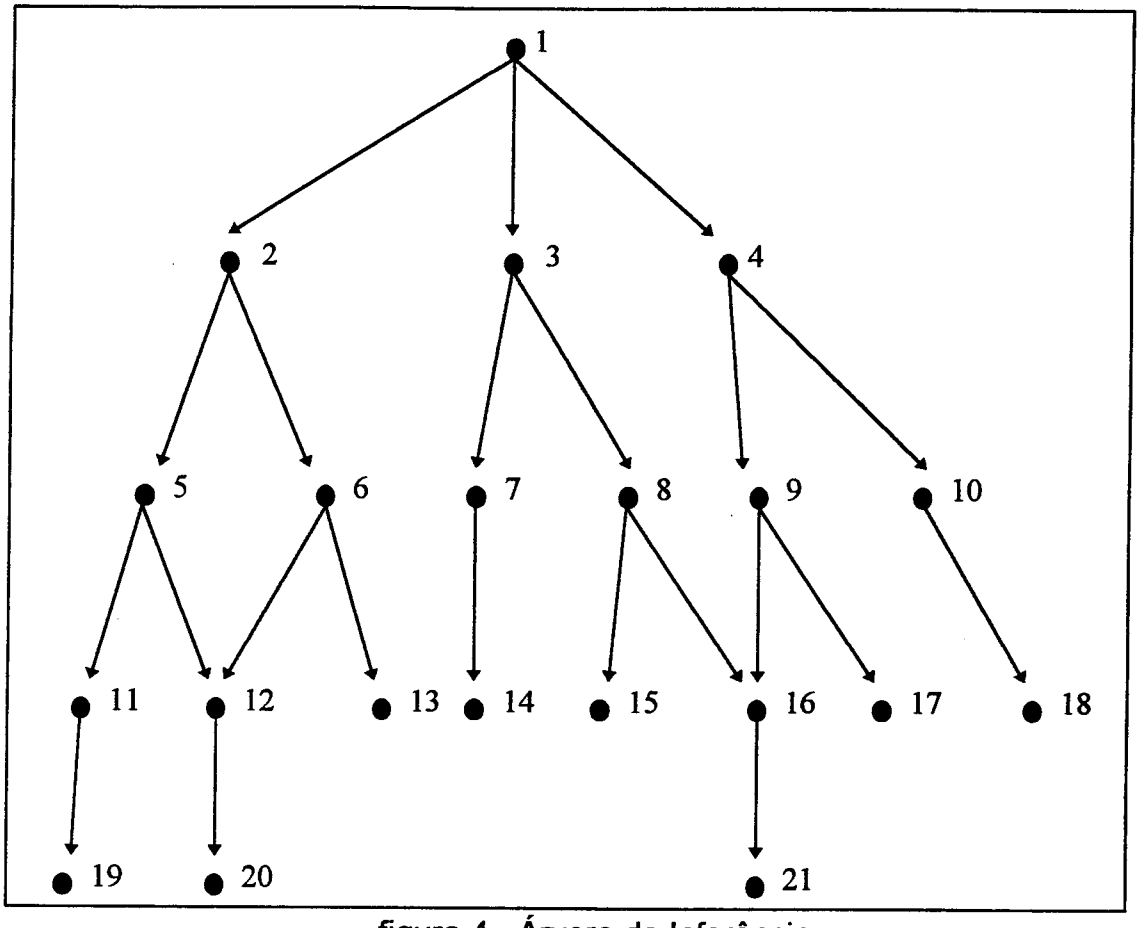

figura 4 - Árvore de Inferências

A busca em profundidade, portanto, segue sempre no sentido de priorizar os "descendentes" de um nó e só quando este caminho estiver esgotado, sem ter levado a uma solução, os caminhos alternativos são considerados. O algoritmo para implementação computacional desta técnica é mostrado a seguir:

** ALGORITMO BUSCA-EM-PROFUNDIDADE **

. crie duas pilhas ABERTOS e FECHADOS

. inicialize a pilha ABERTOS = [nó início]

. enquanto ABERTOS não estiver vazia faça

. remova o nó do topo da pilha ABERTOS e chame-o de X

. se X é conclusivo termine com sucesso

. senão busque todos os filhos de X

. se algum dos filhos de X é conclusivo termine com sucesso

. senão dispense os filhos de X que já estão em ABERTOS ou FECHADOS

. coloque sobre a pilha ABERTOS os filhos remanescentes de X na ordem em que foram buscados

. coloque X sobre a pilha FECHADOS

. fim enquanto

Aplicando-se este algoritmo sobre a árvore da figura 4 o comportamento das pilhas ABERTOS e FECHADOS seria, para cada iteração do laço enquanto:

iteração 1 - ABERTOS = [1]
FECHADOS = []

iteração 2 - ABERTOS = [2,3,4]
FECHADOS = [1]

iteração 3 - ABERTOS = [5,6,3,4]
FECHADOS = [2,1]

iteração 4 - ABERTOS = [11,12,6,3,4]
FECHADOS = [5,2,1]

iteração 5 - ABERTOS = [19,12,6,3,4]
FECHADOS = [11,5,2,1]

iteração 6 - ABERTOS = [12,6,3,4]
FECHADOS = [19,11,5,2,1]

iteração 7 - ABERTOS = [20,6,3,4]
FECHADOS = [12,19,11,5,2,1]

iteração 8 - ABERTOS = [6,3,4]
FECHADOS = [20,12,19,11,5,2,1]

.
.
.

O algoritmo seguiria pesquisando a árvore até que o nó 18 fosse alcançado ou a pilha ABERTOS ficasse vazia.

Numa situação em que fosse necessário o exame de toda a árvore e adotado o critério "o da esquerda antes", o método da busca em profundidade avaliaria os nós na seguinte ordem de seqüência: 1-2-5-11-19-12-20-6-13-3-7-14-8-15-16-21-4-9-17- 10-18.

Dada a seqüência com que o método avalia os nós observa-se a preferência àqueles que estão mais distantes da raiz, ou início da árvore de busca. Esta política é especialmente interessante quando todas as possíveis soluções são igualmente desejadas. Por outro lado, deve-se adicionar neste algoritmo algum dispositivo de controle de limite de aprofundamento pois em certos casos a busca poderia perder-se, especialmente, se a árvore for muito grande ou com profundidade "infinita".

3.3.1 - BUSCA EM LARGURA

Este método, também chamado de primeiro-em-largura, em contraste com o método da busca em profundidade, consiste em explorar os nós nível-a-nível. Somente quando não existem mais nós para serem explorados num certo nível da árvore o algoritmo considera o nível seguinte, ou seja, os nós descendentes. Neste caso são priorizados os "vizinhos" ou "irmãos" de um nó que tenha sido avaliado em prejuízo dos "descendentes". Na árvore da figura 4, por exemplo, se o nó 5 tivesse sido avaliado, e fosse dado prioridade aos vizinhos mais à esquerda, o algoritmo passaria a considerar, na ordem, os nós 6, 7, 8, 9 e 10. Só depois de explorar todos os nós deste nível da árvore é que o nó 11, que é descendente do nó 5, seria considerado.

A busca em largura, portanto, segue sempre no sentido de buscar os "irmãos" de um nó e só quando estes estiverem esgotados sem ter-se chegado a uma solução,

os nós do nível seguinte da árvore são considerados. O algoritmo para implementação computacional desta técnica é mostrado a seguir:

** ALGORITMO BUSCA-EM-LARGURA **

. crie duas filas ABERTOS e FECHADOS

. inicialize a fila ABERTOS = [nó início]

. enquanto ABERTOS não estiver vazia faça

. remova o primeiro nó da fila ABERTOS e chame-o de X

. se X é conclusivo termine com sucesso

. senão busque todos os filhos de X

. se algum dos filhos de X é conclusivo termine com sucesso

. senão dispense os filhos de X que já estão em ABERTOS ou FECHADOS

. acrescente os filhos remanescentes de X na fila ABERTOS na ordem em que foram buscados

. acrescente X na fila FECHADOS

. fim enquanto

Novamente considerando a situação em que fosse necessário o exame de toda a árvore e adotado o critério "o da esquerda antes", o método de busca em largura avaliaria os nós na seguinte ordem de seqüência: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21.

3.3.3 - BUSCA HEURÍSTICA

Heurística, segundo os dicionários, é uma forma de solucionar problemas em que se utiliza experiências passadas mas que não tem garantia de êxito. A inteligência humana muitas vezes usa regras práticas para resolver problemas rotineiros. Por exemplo, uma pessoa que mora em um bairro da cidade e trabalha em outro e quer gastar o menor tempo possível no deslocamento poderia concluir, baseado em experiências anteriores, que de segunda a sexta feira é melhor ir de ônibus já que este tem reservado uma pista da avenida. No sábado a freqüência dos ônibus é menor e por isso precisa-se esperar muito, fazendo com que o uso de veículo próprio seja recomendado. Estas regras práticas, ou heurísticas, podem dar bons resultados mas não há garantia de que funcionem sempre e, certamente, algumas vezes falham.

A aplicação da heurística nos sistemas especialistas baseados em regras de produção também pode ser muito útil, especialmente nos mecanismos de raciocínio, diminuindo as buscas por tentativa-e-erro. As heurísticas são incluídas no sistema através da formalização de critérios para selecionar um determinado caminho nas situações em que várias opções são possíveis.

Segundo Luger, [LUGER,89], os sistemas especialistas podem utilizar a heurística para resolver problemas em, basicamente, duas situações:

1) Quando um problema pode não ter uma solução exata por ambigüidades inerentes à própria natureza do problema. Os diagnóstico médicos são situações deste tipo pois um conjunto de sintomas pode ter múltiplas causas. Nestas situações o médico usa a heurística para selecionar o diagnóstico mais provável.

2) Quando um problema tem uma solução exata mas o custo computacional para achá-la é muito alto. Num jogo de xadrez, por exemplo, a árvore de opções de movimentos associada a uma situação do tabuleiro levaria à combinação explosiva já que o número de opções cresce exponencialmente. Eliminando-se parte destas opções ou caminhos através de uma análise heurística pode-se reduzir o problema e achar uma solução aceitável.

Assim, pode-se afirmar que a heurística é uma "recomendação", "sugestão" ou "palpite" acerca do próximo passo a ser tomado para resolver um problema. Esta recomendação é baseada em experiências anteriores ou mesmo em uma intuição. Comparativamente com os algoritmos apresentados anteriormente na busca em profundidade e busca em largura, a busca heurística determinaria qual dos nós contidos na lista ABERTOS seria avaliado prioritariamente em cada iteração, não havendo um único critério previamente definido.

Existem muitos algoritmos que permitem o uso de heurística. Entre os principais pode-se citar: busca em feixe, subida de encosta, o-melhor-primeiro, entre outros. O principal e mais consistente destes métodos é o o-melhor-primeiro, ou "best-first" que combina as vantagens tanto da busca em largura como em profundidade.

O algoritmo o-melhor-primeiro também usa as listas ABERTOS e FECHADOS para controlar o estado dos nós, porém, para cada um dos nós incluídos na lista haverá um valor associado que é calculado com base numa estimativa heurística. Um passo adicional do algoritmo permite ordenar os nós da lista ABERTOS de acordo com este valor heurístico. Assim, a cada iteração será privilegiado o nó mais promissor da lista. O algoritmo para implementação computacional desta técnica é mostrado a seguir:

** ALGORITMO O-MELHOR-PRIMEIRO **

. crie duas listas ABERTOS e FECHADOS

. inicialize a lista ABERTOS com o nó início e associe a ele o valor zero

. enquanto ABERTOS não estiver vazia faça

. remova o primeiro nó da lista ABERTOS e chame-o de X

. se X é conclusivo termine com sucesso

. senão busque todos os filhos de X e, para cada um deles, verifique

. se ele já está em ABERTOS e o valor associado é pior do que o atual, troque o valor anterior pelo atual

. se ele já está em FECHADOS e o valor associado é pior do que o atual, troque o valor anterior pelo atual e passe-o para ABERTOS

. se ele não está em ABERTOS e nem em FECHADOS, atribua um valor a este filho com base na heurística e inclua-o em ABERTOS

. reordene a lista ABERTOS com base nos valores associados - os melhores primeiro

. acrescente X na lista FECHADOS

. fim enquanto

A cada iteração o algoritmo remove o primeiro elemento da lista ABERTOS. Se este elemento é conclusivo a solução terá sido alcançada. Se o elemento não é conclusivo o algoritmo passa a considerar os seus descendentes ou filhos. Se um destes filhos já está em ABERTOS ou FECHADOS o algoritmo certifica-se de que o valor associado existente na lista é o melhor possível. A lista será então reordenada com base nos valores heurísticos fazendo com que os melhores nós, ou os mais promissores, estejam no início da lista e tenham precedência na iteração seguinte.

O algoritmo seguiria pesquisando a árvore até que uma solução fosse encontrada ou o último nó fosse alcançado, ou ainda, a lista ABERTOS ficasse vazia.

3.4 - FERRAMENTAS PARA DESENVOLVIMENTO

Ferramentas para desenvolvimento de sistemas especialistas são softwares que apoiam e simplificam o trabalho de construção destes sistemas. Dentre as ferramentas as principais são as linguagens de programação com características dirigidas ao desenvolvimento de aplicações na área de inteligência artificial e as "shells" que visam simplificar a construção e gerência da base de conhecimentos nos sistemas especialistas.

3.4.1 - PROLOG

Prolog, uma abreviação de "Programming in Logic", é uma linguagem de programação nascida em Edimburgo na Escócia no início da década de 70. Inicialmente sua utilização ficou restrita a algumas universidades e centros de pesquisa europeus tendo alcançado popularidade mundial depois de o projeto japonês de computadores de quinta geração ter adotado o PROLOG como sua linguagem básica de desenvolvimento. Hoje é a principal linguagem de programação das que permitem representação lógica.

Prolog é uma linguagem declarativa orientada para o processamento simbólico. Um programa escrito nesta linguagem consiste em conjunto de regras que descrevem relações entre objetos. Estas relações, chamadas de predicados do programa, são escritas a partir de um subconjunto do cálculo de predicados, denominado "cláusulas de Horn". Programar em PROLOG envolve aspectos declarativos, que determinam se um certo objetivo é verdadeiro; e aspectos procedimentais que permitem melhorar a eficiência dos programas.

Na maioria dos Prologs os nomes dos predicados são identificadores criados com uma seqüência de caracteres alfanuméricos iniciando com minúsculo. As variáveis também são identificadores livremente criados usando-se caracteres alfanuméricos mas devem iniciar com maiúsculo. Assim, poderia-se escrever *gosta(mario,maria)* para afirmar que Mario gosta de Maria; *gosta(X,maria)* para afirmar que todos gostam de Maria e *gosta(X,maria),gosta (Y,zeca)* para representar o conjunto de pessoas que gostam de Maria e de Zeca simultaneamente. Em outra situação poder-se-ia escrever "Ricardo gosta de Vera se Ricardo gostar de Ana" que seria: *gosta(ricardo,vera) :-gosta(ricardo,ana)* . Já a negação, por exemplo, "Carolina não gosta de Zeca" seria escrito em PROLOG como *not(gosta(carolina,zeca)*. O predicado *gosta*, exige dois parâmetros (nomes próprios). O número de parâmetros determina a aridade do predicado, portanto, o predicado *gosta* tem aridade dois.

Observa-se, assim, que um programa PROLOG é um conjunto de especificações que descrevem objetos e suas relações acerca do domínio do problema utilizando o cálculo de predicados. Este conjunto de especificações formam a base de dados do problema e determinam o que é verdadeiro ou falso em cada situação. O programa responderá questionamentos sobre este conjunto de especificações.

Nos questionamentos, ou "queries", utiliza-se a mesma sintaxe dos comandos mostrados nos exemplos. O PROLOG pesquisará a base de dados para verificar se uma "query" é uma conseqüência das especificações existentes. A "query" *?gosta(mario,maria)* teria resposta positiva e *?gosta(mario,X)*, traria como resposta X = maria, com base nas relações apresentadas anteriormente.

A seguir é apresentado um exemplo clássico para ilustrar o funcionamento de programas PROLOG. As relações que compõem o programa descrevem as relações familiares convencionais.

```
pai(alfredo,joão).
pai(malaquias,mariza).
pai(joão,pedro).
pai(joão,zeca).
pai(joão,maria).
mãe(mariza,pedro).
mãe(mariza,zeca).
mãe(mariza,maria).
pai(pedro,felipe).
mãe(ana,felipe).
irmãos(A,B) :- pai(X,A),pai(X,B),mãe(Z,A),mãe(Z,B).
pais(X,A) :- pai(X,A).
pais(X,A) :- mãe(X,A).
avós(I,J) :- pais(I,K),pais(K,J).
```

```
ancestral(X,Y) :- pais(X,Y).
ancestral(X,Y) :- pais(X,Z),ancestral(Z,Y).
```

Inicialmente foram criadas as relações *pai* e *mãe*, ambos com aridade dois, para descrever os pares pai-filho e mãe-filho. Já a relação *irmãos* é uma regra que define a existência desta relação entre dois elementos. A relação *irmãos* será verdadeira se os dois parâmetros envolvidos tiverem o mesmo pai e a mesma mãe. A regra *pais* também é descrita com base nas relações *pai* e *mãe*. Um questionamento do tipo *pais (joão,zeca)* será verdadeiro se existir a relação *pai(joão,zeca)* ou *mãe(joão,zeca)*; neste caso a relação existe e a resposta será positiva. Para descrever a relação *avós* utilizou-se a relação *pais* para facilitar a descrição da regra. A resposta para a query *?avós(malaquias,maria)* seria positiva, neste exemplo, porque as relações *pais(malaquias,mariza)* e também *pais(mariza,maria)* são verdadeiras.

Já a relação *ancestral* introduz o princípio da recursividade. A primeira cláusula do predicado verifica se a relação *pais* entre os dois elementos envolvidos, X e Y, existe. Em caso positivo a relação *ancestral* também será verdadeira. Em caso negativo, a segunda cláusula será testada quando será selecionada inicialmente uma relação *pais* entre o primeiro elemento, X, e um outro elemento qualquer, Z. Este elemento qualquer é então introduzido como primeiro parâmetro e a relação é novamente verificada como *ancestral(Z,Y)*. Este processo seria repetido até que se confirmasse a existência da relação *ancestral* ou quando todos os pares *pais(Pai_Mãe,Filho)* fossem esgotadas.

Outro recurso da linguagem PROLOG é a utilização de listas nos programas. Lista é uma seqüência ordenada de elementos que é representada na linguagem por um par de colchetes [], onde os elementos podem também serem listas, separados por vírgula. Alguns exemplos de listas em PROLOG são:

```
[paulo,zeca,joão,pedro]
[[joinville,blumenau],[cascavel,londrina],[carazinho,caxias]]
[1,2,3,4]
```

O primeiro elemento da lista é chamado de cabeça da lista e os demais elementos formam a cauda da lista. A cabeça da lista pode ser separada do restante da lista pela introdução do símbolo "|". Para a primeira lista mostrada acima, poderia-se também utilizar a seguinte forma:

```
[paulo|zeca,joão,pedro].
```

Onde o elemento paulo é a cabeça da lista e o restante dos elementos são a cauda da lista. A utilização de listas permite que os programas cresçam em complexidade. Abaixo é mostrado um exemplo em que os elementos integrantes de uma lista são os associados de um clube e um conjunto de relações permitirá verificar se uma pessoa está incluída entre os sócios deste clube.

```
associados([pedro,maria,zeca,almir,rodrigo,beth,hermes,jussara] ).

socio(Nome) :- associados(Todos),membro(Nome,Todos).

membro(Nome,[Nome|Cauda]).

membro(Nome,[Cabeça|Cauda]) :- membro(Nome,Cauda).
```

Se fosse apresentada a query *?socio(zeca)* a resposta seria positiva já que o programa conseguiria verificar a existência deste nome na lista. Se o usuário precisasse, por outro lado, verificar o nome de todos os associados do clube bastaria fazer a query *?associados(X)* e o programa apresentaria como resposta a lista dos nomes.

Observe que o programa funciona recursivamente, ou seja o predicado *membro* tem como condição para verificar a veracidade de uma situação o próprio predicado *membro*. Ele tem aridade dois, exigindo como parâmetros um elemento e uma lista. Ele será verdadeiro quando o elemento estiver contido na lista. Na primeira cláusula do predicado, é verificado se o sócio procurado é o primeiro elemento, ou cabeça, da lista: *membro(Nome,[Nome|Cauda])*; ou seja ele testa se *Nome* e o primeiro elemento da lista são iguais.

Se este elemento não for a cabeça da lista o programa deve, naturalmente, procurá-lo no restante da lista. A segunda cláusula verifica se o sócio procurado está entre os demais elementos (Cauda) da lista. Assim a cabeça da lista é desprezada, o segundo elemento passa a ser a nova cabeça e a comparação é feita novamente; o processo segue até que o elemento procurado seja identificado ou a lista termine (fique vazia sem que ele seja achado) quando então a resposta do programa seria negativa. Já o predicado *socio* tem como condição a veracidade de *membro*.

A criação de predicados num programa PROLOG é feita de forma bastante liberal não existindo restrições quanto ao seu número ou aridade. Adicionalmente aos predicados criados pelo programador a linguagem oferece os chamados predicados meta-lógicos e os predicados extra-lógicos. Eles não fazem parte da lógica dos predicados mas são necessários para o desenvolvimento de programas. Estes predicados estão relacionados a aspectos computacionais da implementação dos programas como tratamento de entrada e saída e interface com sistema operacional do computador.

A linguagem oferece também mecanismo para abreviar as pesquisas inerentes a um programa. Este mecanismo é chamado de corte, ou cut, representado pelo símbolo "!" e permite que o programa seja gerenciado de forma a limitar, quando necessário, as tentativas de verificar a veracidade de um fato.

3.4.2 - LISP

LISP, uma abreviação de "List Processing", é uma linguagem de programação criada por John McCarthy em 1958 no MIT - Massachusets Institute of Technology. Originalmente a linguagem destinava-se à computação simbólica tendo sido posteriormente extendida e refinada para atender às necessidades dos programas na área de inteligência artificial. Durante os quase quarenta anos de existência o LISP tornou-se a linguagem dominante nesta área.

Em contraste com o PROLOG, onde os programas são escritos de forma declarativa, o LISP é uma linguagem procedural, ou seja, as instruções são escritas seqüencialmente em forma de um algoritmo. A sintaxe e a semântica da linguagem são derivadas da teoria matemática e utiliza-se apenas uma estrutura básica que é a lista. Estes fatores fazem do LISP uma linguagem simples que permite um aprendizado rápido.

No início do desenvolvimento da área da inteligência artificial os programas eram implementados em LISP para tirarem proveito das facilidades apresentadas pela linguagem no processamento simbólico. Posteriormente, quando o formalismo na área tornou-se mais acentuado, com a introdução dos princípios dos sistemas especialistas baseados em regras de produção e os provadores de teoremas lógicos, a linguagem foi extendida e refinada, fazendo dela a primeira linguagem para estas implementações. Atualmente, muitos dos sistemas especialistas existentes são programados em LISP.

Os elementos sintáticos da linguagem são expressões simbólicas conhecidas como s-expressions. Tanto as instruções como os dados são representados como s-expressions. Uma s-expression pode ser tanto um átomo como uma lista. O átomo é a unidade sintática básica e inclui números e símbolos. Os átomos simbólicos são compostos por letras, números e outros caracteres como *, -, +, %, &, _, >, < etc. Alguns exemplos de átomos são: 3.1416, 100, x, um-nome, ?outro e nil.

Uma lista é uma seqüência de átomos ou outras listas separadas por espaço e inclusas entre parênteses. Alguns exemplos de lista são:

1. (1 2 3 4)
2. (ana maria zeca juca)
3. (gosta maria ?X)
4. ( )
5. ((ceará pará) acre (rio minas))
6. (* 7 9)

Observe que o exemplo 5 mostra o caso em que listas são elementos da lista. A lista vazia, exemplo 4, tem um papel especial na construção e manipulação de estruturas do LISP e a ela é dada o nome de "nil". A lista vazia é a única s-expression que é considerada simultaneamente um átomo e uma lista. No exemplo 3 a lista é uma variação sintática do cálculo dos predicados. O último exemplo mostra uma expressão aritmética utilizada no LISP para representar o produto de sete e nove.

O LISP é uma linguagem interpretada que quando está ativa no computador mantém um diálogo interativo com o usuário colocando na tela um símbolo, chamado prompt, e aguardando para que os comandos a serem executados sejam digitados. O interpretador lê o comando, tenta avaliá-lo e apresenta o resultado correspondente. Por exemplo, considerando que o prompt seja o sinal ">" :

```
> (* 7 5)
> 35
>
```

Neste caso o usuário digitou (* 7 5) e o interpretador respondeu "35", que é o resultado da multiplicação dos elementos. Na seqüência foi apresentado o prompt para que um novo comando fosse digitado.

Quando é digitada uma lista, o LISP interpreta o primeiro elemento como sendo o nome de uma função e os demais elementos como sendo os parâmetros desta função. Assim, um comando (f x y) é equivalente a tradicional representação matemática f(x,y). O resultado apresentado neste caso seria a aplicação destes argumentos na função. Se for digitado uma expressão que não pode ser avaliada o LISP irá mostrar uma mensagem de erro. Uma sessão simples, porém completa, no LISP é mostrada a seguir.

```
> (+ 14 5)
19
> (+ 1 2 3 4)
10
> (-(+ 3 4) 7)
0
> (* (+ 2 5) (- 7 (/ 21 7)))
28
> (= (+ 2 3) 5)
t
> (> (* 5 6) (+ 4 5))
t
```

Estes exemplos mostram o comportamento do LISP na avaliação de expressões matemáticas. Nos casos em que o primeiro elemento é um operador de comparação a resposta é "t" significando verdadeiro ("true") ou "nil" representando falso.

Quando avalia uma função, o LISP verifica primeiro os argumentos para depois aplicar a função indicada pelo primeiro elemento. Se os elementos também forem funções, a regra é aplicada recursivamente. Por exemplo, ao avaliar a expressão (+(* 2 3) (* 3 5)) os argumentos (* 2 3) e (* 3 5) são avaliados primeiro. Ao avaliar os argumentos 2 e 3, o LISP retorna os seus respectivos valores numéricos, que em caso de números são os próprios. Estes valores são então multiplicados. Similarmente, a segunda lista, (* 3 5), é avaliada retornando o número 15 que é o produto da multiplicação. Estes resultados são então passados para a execução da adição retornando o valor 21.

Observa-se portanto que programação em LISP usa quase exclusivamente as chamadas de funções. Por isto a linguagem oferece um grande número delas para uso do programador. Estas funções, no entanto, variam conforme a versão e o fornecedor da linguagem. O MULISP-86 da Microsoft Corporation, por exemplo, divide as funções em vinte e dois grupos organizados por afinidade. Entre estes grupos os principais são:

. funções de seleção e busca: são funções que tem por objetivo a verificação e a obtenção de informações em lista.

. funções de construção: permitem construir, separar, ver intersecção e outras operações sobre listas.

. funções modificadores: permitem inverter, destruir, ordenar e intercalar listas.

. funções de reconhecimento: permitem reconhecer e identificar listas ou átomos.

. funções comparadoras: são funções para comparar e verificar características dos objetos

. funções lógicas: executam funções booleanas sobre os argumentos

. funções de atribuição: servem para atribuir valores a variáveis do programa.

. funções primitivas: usadas para manipular a definição de funções, alteração de nomes, definição de macros e outras.

. funções numéricas: agrupam as funções para efetuar operações aritméticas.

. funções de entrada: oferecem recursos para carga de programa e leitura de informações.

. funções de saída: reúne as funções relacionadas a saída de informações do computador, gravação de arquivos e impressora.

. funções de controle de memória: permitem gerenciar a utilização da memória principal da máquina.

. funções interface do usuário: reúne recursos para gerenciar interrupções do programa.

O programador LISP cria ainda as suas próprias funções a partir das funções internas da linguagem. As funções criadas a partir das já existentes usam a forma *"defun"* que é uma contração de "define function". Depois de uma função ter sido definida usando *defun* ela poderá ser usada da mesma forma como as demais. Como exemplo, poderia-se criar uma função que calculasse o quadrado de um número, esta função seria:

```
(defun quadrado (x)
(* x x))
```

O primeiro argumento para o defun deve ser o nome da função que está sendo definida; o segundo é a lista de parâmetros associados a esta função, que devem ser todos átomos; os demais argumentos são "s-expressions" que constituem o corpo da nova função e é o que na verdade determina o seu comportamento. Ao contrário de outras funções do LISP, defun não avalia os seus argumentos, ela os usa como especificação para criar uma nova função.

Depois de criada uma função pode ser utilizada diretamente pelo usuário, a exemplo das funções internas, ou para criar novas funções. Poder-se-ia por exemplo utilizar a função quadrado, definida acima, para implementar o teorema de Pitágoras, para o cálculo da hipotenusa de um triângulo. O teorema afirma que a hipotenusa de um triângulo é o resultado da raíz quadrada da soma dos outros lados elevados ao quadrado. A função LISP para efetuar este cálculo seria:

```
(defun hipotenusa (x,y)
(sqrt (+ (quadrado x)
(quadrado y))))
```

A função criada com o nome de hipotenusa utilizou-se da função criada anteriormente com o nome de quadrado e também da função interna sqrt, que retorna a raíz quadrada de um número.

Os programas LISP são construídos, portanto, a partir do agrupamento de funções internas e funções criadas pelo usuário. Estas funções são, geralmente, pequenos pedaços de programa e tem uma tarefa bem definida.

3.4.3 - LINGUAGENS PROCEDURAIS

Linguagens procedurais são aquelas em que são necessários algoritmos para resolver problemas. É necessário descrever, passo a passo, todas as etapas do programa, com todos os possíveis caminhos alternativos.

As linguagens procedurais de uso geral, tais como PASCAL, C, CLIPPER, MÓDULA etc, a princípio também podem ser usadas na construção de sistemas especialistas. Entretanto, a utilização destas linguagens não é comum porque é difícil implementar com elas, programas que infiram soluções não previsíveis e por não oferecerem recursos convenientes para representação do conhecimento, tratamento de listas e manipulação de símbolos. Estes recursos, oferecidos pela linguagens dirigidas às aplicações de sistemas especialistas, poderiam, no caso das linguagens procedurais, serem criados pelo programador, mas demandariam um grande esforço adicional.

Independente da linguagem procedural que use, o programador constrói o programa para fazer o computador executar uma seqüência de passos detalhadamente descritos para alcançar uma solução. Esta relação de passos, ou procedimento, em qualquer linguagem procedural precisa ser excessivamente detalhada para as aplicações em sistemas especialistas. Segundo Keller, [KELLER,91] este nível de detalhamento faz com que o programador tenha

facilmente sua atenção desviada da essência do problema a ser resolvido, e, somado à falta de facilidades específicas para estes tipos de problemas, podem criar um obstáculo intransponível para o programador.

3.4.4 - "SHELLS"

"Shells" é o nome dado a uma família de ferramentas, não linguagens de programação, que objetivam apoiar e simplificar o processo de construção de sistemas especialistas. São softwares que contêm alguns dos principais elementos de um sistema especialista, tais como, o motor inferência, o justificador e outros. Estas ferramentas também pré-definem a estruturação do conhecimento a ser utilizada pelo sistema. Ao projetista do sistema especialista usuário de uma shell cabe apenas a tarefa de construir a base de conhecimentos.

Nos primeiros sistemas especialistas desenvolvidos construía-se todos os módulos do sistema antes que fosse criada a base de conhecimentos sobre a qual o sistema iria agir. Posteriormente, observou-se que muitos deles tinham em comum alguns destes módulos, pois eles eram construídos como um conjunto de representações declarativas, especialmente em forma de regras, que eram combinadas com um interpretador destas representações. Assim, constatou-se que seria possível criar sistemas genéricos que poderiam ser usados para criar novos sistemas especialistas pela simples adição ou criação da base de conhecimentos correspondente ao domínio do problema. Estes sistemas de uso genérico receberam o nome de "shells".

As "shells" representam a tendência atual no processo de desenvolvimento de sistemas especialistas. As primeiras "shells" surgiram a partir dos sistemas especialistas MYCIN, PROSPECTOR e CASNET, dos quais foram retiradas as bases de conhecimentos específicas, dando origem, respectivamente, as "shells" EMICYN, KAS e EXPERT. Atualmente existem muitas destas ferramentas

comercialmente disponíveis que servem de apoio a muitos sistemas especialistas em desenvolvimento. Elas suportam vários dos mecanismos de racíocinio e utilizam as representações de conhecimento mais comuns, especialmente regras e frames. As "shells" mais modernas incorporaram também recursos para a aquisição do conhecimento e interfaces para bancos de dados de sistemas convencionais.

As "shells" que utilizam regras para a representação do conhecimento podem ser divididas em dois grupos, diferenciadas pela forma de entrada das regras no sistema. As do primeiro grupo utilizam editores de texto de uso geral para editar o arquivo que contém o conjunto de regras do sistema. Posteriormente a "shell" interpreta e executa as regras. A vantagem desta forma de edição é que ela oferece grande flexibilidade sintática ao usuário. Entre as mais conhecidas deste grupo estão o M.1 e o INSIGHT 2+ .

No segundo grupo estão as "shells" que possuem o seu próprio editor. Elas são geralmente orientadas através de menus e facilmente operadas pelo usuário. Entre as mais comuns deste grupo pode-se citar o EXSYS e o PC Easy.

A seguir são apresentados quatro exemplos de regras utilizadas, respectivamente, pelas "shells" M.1, INSIGHT 2+, EXSYS e PC Easy extraídas de [RABUSKE,95] e [THOMPSON,87].

Exemplo 1 - Regra do M.1

rule-1:

if empregado = membro-da-família e
relacionamento = filha ou
relacionamento = filho e
idade = N e
N > 20
display(paga imposto)
then tipo-empregado = ok .

Exemplo 2 - Regra do INSIGHT 2+

RULE acha filhos empregados muito velhos
IF empregado IS membro-da-família
AND o relacionamento com o empregador IS filha
OR o relacionamento com o empregador IS filho
AND idade > 20

THEN informação IS conhecida
AND DISPLAY messagem1

Exemplo 3 - Regra do EXSYS

Regra 1

SE
O semáforo está sinalizando vermelho
e O carro passou em alta velocidade
e O guarda anotou a placa

ENTÃO
O dono do carro receberá uma notificação - Probabilidade 9/10
e O dono do carro será intimado a comparecer ao DETRAN - Probabilidade 4/10.

Exemplo 4 - Regra do PC Easy

RULE 001[KB-RULES]
IF :EMPREGADO = FAMÍLIA AND RELACIONAMENTO = OUTRO
THEN :TIPO-EMPREGADO = OK AND PRINT "paga imposto"

Verificando-se os exemplos fica evidenciado que as "shells" que não usam editores específicos, como o M.1 e o INSIGHT 2+, oferecem maior liberdade ao usuário na edição, exigindo entretanto, que o sistema seja adaptado ao idioma utilizado nas regras ou que as regras sejam escritas na língua que o sistema aceita. No exemplo 3, que mostra a sintaxe de uma regra do EXSYS, observa-se que todas as palavras utilizadas estão escritas em português, incluindo as palavras chaves como o SE e o ENTÃO. O EXSYS, um produto da Exsys Inc de Albuquerque, Novo

México nos Estados Unidos foi adaptado ao português pela Base Tecnologia Ltda do Rio de Janeiro.

O uso de "shells", portanto, facilita bastante a construção de sistemas especialistas porque não exige maiores preocupações com detalhes de programação bastando apenas ao usuário o aprendizado do sistema e a elaboração da base de conhecimentos. Por outro lado, uma das características restritivas das "shells" é a sua rigidez na representação do conhecimento, podendo trazer dificuldades em algumas situações específicas.

3.5 - INCERTEZA

Associado ao conhecimento existe, quase sempre, uma fator de certeza envolvido em cada situação de decisão ou mesmo inerente a própria área do conhecimento. Um médico precisa determinar um diagnóstico e um tratamento para um paciente diante de sintomas ambíguos, as pessoas se comunicam, e se compreendem, utilizando sentenças incompletas ou ambíguas e assim por diante. A incerteza, portanto, integra a vida cotidiana das pessoas e precisa ser considerada na construção de sistemas especialistas.

A incerteza está presente em diversas situações da manipulação e representação do conhecimento dos sistemas especialistas. Está presente na coleta de informações, na definição dos elementos do conhecimento, na combinação dos elementos e na forma de tirar conclusões.

Na coleta de informações o próprio usuário irá se deparar com informações de difícil classificação até mesmo em situações simples, como por exemplo, a caracterização da temperatura da água que poderia ser por um classificado como fria e não aceito por outro, que a classificaria como gelada. Na definição dos elementos de conhecimento, quando é preciso fazer sínteses, é comum a utilização de recursos

de estatística e probabilidade para depurar índices e freqüências. Na combinação de elementos entre si a presença da incerteza é ainda mais acentuada pois é preciso definir se eventos são ou não independentes e estabelecer quanto cada informação pesa sobre a conclusão. A avaliação de uma seqüência de eventos é também um foco de incerteza no sistema, e talvez o mais sensível e complexo, pois ele estabelecerá os raciocínios que o sistema efetuará.

Da impossibilidade natural de descrever precisamente as informações relacionadas a uma área de conhecimentos qualquer, ou mesmo a dificuldade de construir o conjunto completo de informações do domínio do problema, surgiram as propostas de utilização das lógicas alternativas pelos sistemas especialistas. Outra opção adotada por muitos sistemas foi a manutenção da lógica precisa, porém, com a incorporação de um fator de certeza associado as informações.

3.5.1 - LÓGICA FUZZY

A termo fuzzy é utilizado para descrever conjuntos cujos critérios de pertinência são imprecisos. A incerteza a respeito de uma afirmação é expressa através de um número que determina a possibilidade de a afirmação ser verdadeira. Dentro desta perspectiva, utilizando-se o exemplo da temperatura da água, citado anteriormente, poderia-se representar que a água *está muito quente*, *pouco quente*, *morna*, *fria*, *não muito fria*, etc. O conjunto nebuloso "*quente*" seria então representado por valores, entre 0 e 1 ou 0 e 100, que determinariam quão quente a água está. Poderia-se, por exemplo, convencionar que a relação entre a temperatura e o valor associado seria da seguinte forma:

graus Celsius	grau de pertinência a quente	grau de pertinência a não quente
00	0	100
20	5	95
40	15	85
50	25	75
60	40	60
70	60	40
80	80	20
90	95	5
100	100	0

O objetivo da lógica fuzzy seria, então, modificar ou "diluir" (do inglês fuzzify) a lógica para permitir a manutenção de proposições conflitantes. Conforme Turner, [TURNER,85] a lógica fuzzy se origina de dois estágios de "difusão":

1) da introdução de predicados difusos na linguagem objeto, resultando em alguma forma de lógica multivalorada

2) do tratamento dos predicados metalingüísticos "verdadeiro" e "falso" também como vagos ou difusos.

Assim, ao contrário da teoria clássica dos conjuntos que só permitem proposições que assumem valores verdadeiro e falso, a teoria dos conjuntos fuzzy, ou nebulosos, admite que um elemento tenha um grau de pertinência a um conjunto. Em geral este grau de pertinência é um número do intervalo de 0 a 1, ou ainda, de 0 a 100 como mostrado no exemplo acima.

Conforme os conceitos básicos da teoria dos conjuntos, toda propriedade determina um conjunto, que poderia denominar-se de conjunto de coisas que satisfazem a propriedade em questão. Na teoria dos conjuntos difusos os elementos de um conjunto e as suas propriedades também compartilham uma relação de intimidade qualquer, mas a sua natureza é diferente. Propriedades ou predicados como vermelho e alto são difusos ou vagos. Não seria possível, portanto, afirmar de forma definitiva que uma pessoa é alta, pouco alta, muito baixa etc, pois os predicados são vistos como funções do conjunto entre [0,1].

Assim, os conjuntos difusos são determinados por um predicado difuso, e este predicado constitui a função de relacionamento dos elementos. Para um conjunto difuso S criaria-se então a função de relacionamento Us. O conjunto passaria então a ser visto, não como uma coleção de objetos, mas sim como um agrupamento de objetos com uma indicação do grau de relacionamento entre eles.

O primeiro sistema especialista que utilizou alguns dos conceitos de lógica fuzzy foi o PROSPECTOR em suas relações lógicas. Na aplicação desta lógica em sistemas especialistas baseado em regras de produção, em geral, existe um fator de certeza associado às regras ou às cláusulas que compõem a regra. No processo de raciocínio, por suas vez, chega-se a conclusões cuja incerteza é uma acumulação das incertezas das premissas. Como conseqüência as conclusões e também os fatores de certeza são conjuntos difusos caracterizados por suas distribuições de possibilidade.

3.6 - LIMITAÇÕES

A construção de sistemas especialistas baseados em regras de produção apresenta algumas limitações específicas e outras que são também extensivas aos sistemas que se utilizam de outras formas de representação do conhecimento.

Entre as desvantagens específicas desta forma de representação do conhecimento destaca-se a necessidade de modularizar e uniformizar o conhecimento em forma de regras. Esta característica torna difícil a verificação da completeza destes sistemas e a verificação dos possíveis fluxos de processamento do mecanismo de raciocínio. Uma forma de amenizar estes problemas é o agrupamento de regras afins, o que permite segmentar a base de conhecimentos em subconjuntos facilitando o entendimento e a manutenção.

Outro problema relacionado a modularidade e uniformidade destes sistemas é a ineficiência na execução, principalmente quando o número de regras é grande. A execução pode tornar-se lenta pois a verificação das regras que se aplicam a um estado do problema pode demandar um grande esforço computacional.

Segundo Winston, [WINSTON,87] faltam aos sistemas de produção muitas das características dos especialistas humanos, citando especialmente:

a) capacidade de aprender
b) raciocinar em vários níveis
c) olhar o problema a partir de perspectivas diferentes
d) ter acesso ao raciocínio que está por trás das regras.

Certamente as deficiências apontadas por Winston não são exclusivas dos sistemas de produção e sim extensivas a todos os sistemas especialistas .

Outra dificuldade generalizada nos sistemas especialistas é o gerenciamento da incerteza. Muito embora já existam técnicas para embutir a incerteza nestes sistemas, elas ainda são muito primárias e insuficientes. Rabuske, [RABUSKE,95] afirma que "à medida que cresce a complexidade dos problemas, cresce a necessidade do uso de teorias mais amplas, pois o humano não consegue mais modelar estes problemas nas teorias tradicionais, muito precisas e inflexíveis". Espera-se, portanto, que futuramente, com o surgimento e aprimoramento das lógicas alternativas, outras técnicas de representação e tratamento da incerteza sejam incorporadas no desenvolvimento de sistemas especialistas.

CAPÍTULO 4

4 - O PROTÓTIPO DESENVOLVIDO

Neste capítulo será apresentado a especificação e a implementação de uma ferramenta que fornece um ambiente para a construção de sistemas especialistas. Esta ferramenta, denominada **SISREP - SISTEMA PARA REGRAS DE PRODUÇÃO**, pode ser classificada como um software protótipo pois alguns aspectos computacionais importantes, como segurança da base, credenciamento de usuários , controle de códigos de retorno nas operações de entrada/saída e a opção de ajuda ao usuário (help on-line) foram suprimidos para que fosse viável o seu desenvolvimento em tempo hábil. Por outro lado, o software permitirá a construção, desde já, de sistemas especialistas completos pois disponibiliza todos os recursos necessários para tal.

A ferramenta baseia-se na representação do conhecimento através de regras de produção podendo ser utilizado para a criação de sistemas especialistas em geral, não sendo dirigido a qualquer área específica. Para a edição das regras o SISREP oferece ao usuário um editor próprio não se utilizando, portanto, de outros editores de texto. O formato das regras, a arquitetura do sistema e os demais detalhes da implementação serão mostrados ao longo do presente capítulo.

4.1 - AMBIENTE DA IMPLEMENTAÇÃO

No desenvolvimento do software foi utilizada a linguagem de programação ARITY/PROLOG versão 6.0.91 da Arity Corporation. Os programas foram construídos utilizando-se, inicialmente, o interpretador oferecido pela linguagem (versão 6.00.99), tendo sido compilados depois de concluídos e testados.

A escolha desta linguagem deveu-se basicamente aos seguintes aspectos:

- a linguagem de programação é dirigida a aplicações na área de sistemas especialistas. Algumas das caraterísticas desta linguagem, tais como o formalismo, poder de interpretação e inferência e recursos para manipulação de listas facilitam o desenvolvimento de aplicações desta natureza.

- a portabilidade da linguagem. O ARITY/PROLOG está disponível para microcomputadores pessoais da família PC que utilizam a sistema operacional DOS.

- os recursos de interpretação-compilação oferecidos pela linguagem agilizam os trabalhos de programação pois permitem a verificação imediata dos resultados de cada uma das instruções do programa.

- a adoção desta linguagem permite o aprofundamento no estudo de uma linguagem de programação de filosofia nova e moderna, bastante diferente das linguagens procedurais convencionais.

O sistema operacional utilizado foi o MS-DOS versão 3.3 da Microsoft Corporation por ser o ambiente operacional convencional dos microcomputadores compatíveis com IBM-PC. O hardware utilizado foi um microcomputador PC-486 SX de 33 Mhz com 4 Mb de memória RAM e 80 Mb de disco rígido.

4.2 - REQUISITOS DA BASE DE CONHECIMENTOS

O sistema utiliza regras de produção para a representação do conhecimento. A estrutura das regras é apresentada abaixo, expressa pela notação BNF (Backus-Naur Form) :

<regra> ::= SE <condição> ENTÃO <cláusula> <certeza> <validade>

<condição> ::= <cláusula> | <cláusula> E <cláusula>

<cláusula> ::= <atributo> <predicado> <valor>

<atributo> ::= <cadeia>

<predicado> ::= == | !=

<valor> ::= <cadeia>

<cadeia> ::= { letra minúscula }

<letra minúscula> ::= a | b | c | d | ..... | v | x | z | |

<certeza> ::= 1 | 2 | 3 | 4 | ..... | 98 | 99 | 100

<validade> ::= <dia>/<mês>/<ano> <hora>/<min>/<seg>

<dia> ::= 1 | 2 | 3 | ..... | 31

<mês> ::= 1 | 2 | 3 | ..... | 12

<ano> ::= 70 | 71 | 72 | ..... | 99

<hora> ::= 0 | 1 | ..... | 23

<min> ::= 0 | 1 | 2 | ..... | 59

<seg> ::= 0 | 1 | 2 | ..... | 59

Como pode-se observar na BNF, a regra é formada pelas partes SE e ENTÃO. Na parte SE, ou antecedente, da regra estão as premissas. Ela é formada por uma ou mais cláusulas. Já na parte ENTÃO, ou conseqüente, está a conclusão da regra que é formada por uma cláusula única.

Uma cláusula é uma combinação de um atributo, um predicado e um valor. Os atributos e valores são cadeias de caracteres que o usuário cria livremente. Os predicados utilizados na regra são "=="(igual) e "!="(diferente). Para cada regra existe ainda um fator de certeza e uma prazo de validade. O fator de certeza expressa o grau de certeza do conjunto de cláusulas que compõem a regra. Já a validade permite limitar o prazo de validade de uma regra.

4.2.1 - ATRIBUTOS E VALORES

Um atributo é uma das idéias centrais existentes no universo para o qual o sistema especialista está sendo construído. Estas idéias representam os aspectos ou questões que devem ser consideradas e respondidas no processo de avaliação ou raciocínio. Os atributos são classificados em dois tipos: intermediário ou conclusivo. Um atributo conclusivo é aquele que representa uma solução para o problema, ou seja, existindo uma resposta para o atributo estará existindo uma resposta para o próprio problema como um todo. Já os atributos intermediários são os que

representam uma conclusão apenas parcial que exigirá a continuidade do processo de busca para alcançar uma solução completa do problema em questão.

Para cada um dos atributos existe um domínio de respostas que eles podem assumir. Estas possíveis respostas que um atributo pode assumir são chamadas de valores. Abaixo, são apresentados vários exemplos que permitem um melhor entendimento da utilização dos atributos e valores no sistema.

Atributo: *febre*
Valores: *{ ausente, alta, estável, ... }*

Atributo: *coloração das flores*
Valores: *{ verde, marrom, vermelha, ... }*

Atributo: *intensidade do tráfego*
Valores: *{ normal, intenso, baixo, ... }*

Atributo: *número de rodas*
Valores: *{ 2, 4, 12, ... }*

Atributo: *diagnóstico*
Valores: *{ pneumonia, úlcera duodenal, amigdalite, ... }*

A criação de atributos e valores para utilização no sistema é livre, não existindo qualquer restrição quanto ao seu número ou tipo. Cabe ao usuário do sistema fazer o cadastramento prévio dos mesmos para posteriormente utilizá-los na elaboração das regras. Também cabe ao usuário definir o tipo de cada um dos atributos, se conclusivo ou intermediário.

4.2.2 - CLÁUSULAS E PREDICADOS

As cláusulas são formadas pela ligação de um atributo com um valor através do uso de um predicado. Os predicados tornados disponíveis pelo SISREP são a igualdade, representado pelo símbolo ==, e a desigualdade, representado pelo símbolo != . Alguns exemplos de cláusulas aceitas pelo sistema são:

Atributo : *febre*

Predicado: ==

Valor : *alta*

Atributo : *coloração das flores*

Predicado : ==

Valor : *vermelha*

Atributo : *intensidade do tráfego*

Predicado : *!=*

Valor : *normal*

4.2.3 - REGRAS

As regras compõem o conhecimento do sistema especialista. Elas são construídas através da conjunção de cláusulas. Cada regra é composta por uma cláusula, obrigatória, na sua parte antecedente e uma cláusula, também obrigatória, na sua parte conseqüente. Alternativamente a regra pode possuir, ainda, até duas cláusulas ligadas a parte antecedente através do conectivo lógico E .

4.2.3.1 FORMATO

A forma básica das regras utilizadas pelo SISREP é mostrada a seguir:

SE cláusula antecedente

E cláusula optativa 1

E cláusula optativa 2

ENTÃO cláusula conseqüente

CERTEZA : 1 a 100 %

VALIDADE : dia/mês/ano hora/minuto/segundo

Quando no processo de inferência for alcançada uma regra que possua na cláusula conseqüente um atributo conclusivo o sistema encerra o encadeamento e assumirá que uma solução do problema foi encontrada. A solução será o valor

associado a este atributo conclusivo. Já para as cláusulas antecedente e optativas o SISREP não permitirá que atributos conclusivos sejam utilizados. Convencionou-se, ainda, que o sistema não permitirá que um mesmo atributo seja utilizado em duas cláusulas de uma mesma regra.

4.2.3.2 - FATOR DE CERTEZA E VALIDADE

Para cada regra existente na base de conhecimentos do sistema especialista estará associado um fator de certeza permitindo que o conhecimento impreciso seja incorporado à base. O fator de certeza é um número inteiro entre 1 e 100 que demonstra o percentual de certeza do conhecimento descrito pela regra. Deve-se observar que o fator de certeza está relacionado à regra como um todo, ou seja, ao conjunto de cláusulas que compõem a regra.

Cada regra pode ainda ter associado um prazo de validade de sua veracidade. A validade é informada pelo usuário quando da criação da regra. As unidades utilizadas na representação do vencimento da veracidade da regra vão de ano até segundo permitindo que mesmo as situações extremas de intervalo de tempo possam ser representadas. Quando uma regra deixa de ser verdadeira ela não será eliminada fisicamente da base de conhecimentos mas deixará de ser avaliada nos processos de inferência.

4.3 - ARQUITETURA

4.3.1 - ARQUIVOS

O ARITY/PROLOG oferece a opção de um banco de dados próprio para armazenamento de termos (variáveis, números inteiros, números de pontos flutuantes, "strings", etc) e cláusulas. Neste banco de dados denominado "database" são armazenadas todas as informações necessárias ao funcionamento do SISREP.

Cada termo ou conjunto de termos é armazenado sob uma chave ou "Key". Esta chave é o nome pelo qual o termo ou conjunto de termos será conhecido pelo programador e pelo programa. Alguns exemplos de chaves válidas são:

carros	um átomo do PROLOG
1	um número inteiro
$base$	um string do PROLOG
pai(_,_)	um termo de aridade 2

O banco de dados do ARITY/PROLOG utiliza os princípios da chamada memória virtual, ou seja, os dados podem estar tanto na memória principal do computador como na memória auxiliar (em disco), que a própria linguagem administra a disponibilização dos mesmos quando necessário. Para o programador não existe qualquer diferença com relação à utilização de informações que estejam na memória principal ou auxiliar. O espaço em disco é tratado simplesmente como uma extensão da memória do computador.

O banco de dados do ARITY/PROLOG é dividido em páginas de 4 kilobytes. Estas páginas são levadas para a memória do computador ou para o disco conforme as necessidades do programa em execução. Quando um programa, por exemplo, utiliza um termo que está armazenado no banco de dados mas não se encontra em

nenhuma das páginas que estão na memória principal naquele instante a linguagem se encarrega de verificar em qual página ele se encontra. Esta página será então trazida para a memória. Já a página que estava mais tempo na memória principal sem ter sido utilizada será devolvida para o disco para que se tenha espaço para a carga da página requerida pelo programa.

Como conseqüência da utilização dos princípios de memória virtual para o banco de dados a linguagem disponibiliza muito mais espaço para o armazenamento de informações do que se poderia armazenar na memória isoladamente. A memória total que o ARITY/PROLOG permite utilizar suporta até 2 gigabytes. Este limite, no entanto, está condicionado ao limite físico de memória em disco do computador em uso.

4.3.1.1 - ARQUIVO DE BASES DE CONHECIMENTOS

O SISREP armazena sob a chave *bases* o nome de todas as bases de conhecimentos criadas pelo usuário. O nome da base de conhecimentos é definida pelo usuário do sistema quando da criação da mesma. Estes nomes são livremente criados e serão os identificadores dos sistemas especialistas criados pela ferramenta. O tamanho máximo do nome da base é vinte letras não sendo permitida a utilização de números e caracteres especiais.

O nome dado a uma base de conhecimentos também é utilizada como uma chave do banco de dados. Sob esta chave estará armazenada a descrição da base, o número do último atributo cadastrado para esta base, o número do último valor cadastrado para esta base e o número da última regra cadastrada para esta base. A descrição da base é um campo oferecido pela ferramenta para que o usuário possa fazer uma breve descrição, com no máximo sessenta letras, da finalidade de uma base de conhecimentos. Já os números associados ao último atributo, valor e regra são

campos de controle utilizados exclusivamente pelo SISREP. A seguir são mostrados dois exemplos para ilustrar o formato destes arquivos.

chave : bases
termos : medica
classificar vegetais

chave : medica
termos : diagnóstico de doenças infecto-contagiosas
12
27
36

chave : classificar vegetais
termos : apoio a classificação de vegetais da mata atlântica
0
1
0

4.3.1.2 - ARQUIVO DE ATRIBUTOS

O SISREP concatena o nome da base com o string *+atb* para formar um novo string que será a chave sob a qual serão armazenados os atributos daquela base. Para cada atributo cadastrado existirão três termos associados: o número do atributo, a descrição do atributo e o tipo do atributo, sendo zero para intermediário e um para conclusivo. Os atributos também são livremente criados pelo usuário respeitando-se o limite convencionado de 35 letras para a descrição. Para que um atributo possa ser utilizado numa regra ele deve ser previamente cadastrado.

A seguir é mostrado um exemplo para ilustrar a forma de armazenamento de dois atributos que poderiam pertencer a primeira base mostrada anteriormente.

chave :	$medica+atb$	
termos :	1	(número do atributo)
	febre	(descrição do atributo)
	0	(tipo do atributo)
	2	(número do atributo)
	diagnóstico	(descrição do atributo)
	1	(tipo do atributo)

Cada atributo tem associado a ele um número permitindo que o mesmo seja referenciado no SISREP sem que se tenha que digitar toda a sua descrição. Este número é determinado pelo sistema quando do cadastramento do atributo. Quando um atributo é eliminado da base o seu número não é reutilizado pois para cada novo atributo cadastrado o sistema verifica no banco de dados o número do último atributo já cadastrado, incrementa este número e o utiliza para o novo atributo.

4.3.1.3 - ARQUIVO DE VALORES

O SISREP concatena o nome da base com o string *+vlr* para formar um novo string que será a chave sob a qual serão armazenados os valores daquela base. Para cada valor cadastrado existirão dois termos associados: o número do valor e a descrição do valor. Os valores também são livremente criados pelo usuário respeitando-se o limite convencionado de 20 letras para a descrição. Para que um valor possa ser utilizado numa regra ele deve ser previamente cadastrado.

Na criação de uma base o SISREP cadastra automaticamente o valor de número um, cuja descrição é "*desconhecido*". Isto se deve ao fato de que este valor

poderá ser a resposta para qualquer um dos atributos no processo de inferência. O valor um não pode ser alterado nem eliminado de uma base.

A seguir é mostrado um exemplo para ilustrar a forma de armazenamento de dois valores que poderiam pertencer a segunda base mostrada anteriormente.

chave :	$classificar vegetais+vlr$	
termos :	3	(número do valor)
	verde	(descrição do valor)
	4	(número do valor)
	avermelhado	(descrição do valor)

A exemplo dos atributos, quando um valor é eliminado da base o seu número não é reutilizado pois para cada novo valor cadastrado o sistema verifica no banco de dados o número do último valor já cadastrado, incrementa este número e o utiliza para o novo valor.

4.3.1.4 - ARQUIVO DE REGRAS

O SISREP concatena o nome da base com o string *+regras* para formar um novo string que será a chave sob a qual serão armazenadas as regras do sistema. Para cada regra cadastrada existirão oito termos associados: o número da regra, a situação da regra, a cláusula antecedente, a primeira cláusula optativa, a segunda cláusula optativa, a cláusula conseqüente, o fator de certeza e a validade.

A situação da regra é um campo utilizado pelo algoritmo de busca, no processo de inferência, para controle de regras "abertas" e "fechadas". As cláusulas são armazenadas em forma de um string que concatena o número do atributo, o predicado e o número do valor; as descrições do atributo e do valor não são incluídas. Quando uma das cláusulas optativas não existir, o campo é preenchido com

o string *$A V$* que representa para o SISREP a ausência da mesma. A seguir é mostrado um exemplo para ilustrar a forma de armazenamento das regras de uma base de nome "medica".

chave :	$medica+regras$	
termos :	27	(número da regra)
	normal	(situação)
	$A12==V5$	(cláusula antecedente)
	$A3!=V8$	(cláusula optativa 1)
	$A V$	(cláusula inexistente)
	$A27==V41$	(cláusula conseqüente)
	82	(fator de certeza)
	$2/8/96/3/12/25$	(validade até 02/08/1996 as3hr12min25seg)

4.3.2 - MÓDULOS DESENVOLVIDOS

4.3.2.1 - MANUTENÇÃO DO ARQUIVO DE BASES

Para a manutenção do arquivo de bases são implementadas as opções de criação de uma base, alteração do nome de uma base, alteração da descrição de uma base e eliminação de uma base.

Para a criação de uma base, é apresentada ao usuário uma tela com dois campos, de preenchimento obrigatório, para que seja informado o nome da base e a descrição da base. Após o preenchimento destes campos é mostrada uma mensagem para que o usuário cadastre os atributos, valores e regras da base criada. A ferramenta oferece ainda a opção de alteração do nome e descrição de uma base existente. A alteração do nome de uma base fará com que as chaves sob as quais estão armazenados os atributos, valores e regras também sejam alteradas automaticamente pelo SISREP. Já a alteração da descrição de uma base não tem maiores implicações pois somente este campo será alterado no banco de dados.

A eliminação de uma base é sempre precedida pela solicitação de confirmação do usuário para que sejam evitados acidentes. Quando houver a confirmação da eliminação de uma base a ferramenta fará automaticamente a exclusão de todos os arquivos associados a esta base, ou seja, os seus atributos, valores e regras.

O SISREP não impõe qualquer limite com relação à criação de bases. O limite para a sua criação está condicionado aos limites de recursos de memória do computador utilizado e ao limite de 2 gigabytes de memória virtual do banco de dados do ARITY/PROLOG.

4.3.2.2 - MANUTENÇÃO DO ARQUIVO DE ATRIBUTOS

Para a manutenção do arquivo de atributos são oferecidas as opções de inclusão, alteração e exclusão. Para a inclusão de um atributo é apresentado ao usuário uma tela com dois campos, de preenchimento obrigatório, para que seja informado a descrição do atributo e o seu tipo. O número que um atributo recebe quando de sua inclusão é automaticamente definido pelo SISREP, sendo que o usuário é apenas informado.

A alteração de um atributo se resume ao seu tipo e descrição pois o seu número não pode ser alterado. Quando a descrição de um atributo é alterada não há qualquer alteração nos demais arquivos. Já a alteração do tipo do atributo poderá ter implicações também no arquivo de regras. Isto acontecerá sempre que um atributo mudar de intermediário para conclusivo e ter sido anteriormente utilizado na cláusula antecedente ou em uma das cláusulas optativas de uma regra da base de conhecimentos. Neste caso o usuário será informado e, caso confirme a alteração, a regra terá sido corrompida e portanto é eliminada.

A eliminação de um atributo é sempre precedida pela solicitação de confirmação do usuário para que sejam evitados acidentes. Quando houver a confirmação a ferramenta eliminará automaticamente o atributo e todas as regras que o utilizam em suas cláusulas.

4.3.2.3 - MANUTENÇÃO DO ARQUIVO DE VALORES

A exemplo dos atributos, para a manutenção do arquivo de valores, são oferecidas as opções de inclusão, alteração e exclusão. Para a inclusão de um valor é apresentado ao usuário uma tela com um campo único onde deve ser informado a descrição do valor. O número que um valor recebe quando de sua inclusão é automaticamente definido pelo SISREP, sendo que o usuário é apenas informado.

A alteração de um valor se resume à sua descrição pois o seu número não pode ser alterado. Quando a descrição de um valor é alterada, não há qualquer alteração nos demais arquivos. Já a eliminação de um valor poderá ter implicações também no arquivo de regras. Isto acontecerá sempre que um valor tiver sido anteriormente utilizado numa cláusula de uma das regras da base de conhecimentos. Neste caso o usuário será informado e se confirmar a sua intenção a ferramenta eliminará automaticamente o valor e todas as regras que o utilizam em suas cláusulas. O valor 1, ou seja, "*desconhecido*" não pode ser alterado nem eliminado.

4.3.2.4 - MANUTENÇÃO DO ARQUIVO DE REGRAS

Para a manutenção do arquivo de regras são oferecidas as opções de inclusão, alteração e exclusão. Para a inclusão de uma regra é apresentada ao usuário uma tela com quinze campos. Doze destes campos correspondem às quatro cláusulas possíveis de serem utilizados numa regra. Cada uma destas cláusulas utiliza três campos que correspondem a um atributo, um predicado e um valor. As cláusulas antecedente e conseqüente são obrigatórias em todas as regras. Será solicitado, ainda, que seja informado o fator de certeza da regra, a data e a hora limites de validade da regra. O número que uma regra recebe quando de sua inclusão é automaticamente definido pelo SISREP, sendo o usuário apenas informado.

A alteração de uma regra pode estar relacionada a qualquer um dos seus quinze campos. Quando o usuário informar a número da regra a ser alterada o SISREP apresentará esta regra na tela e permitirá que qualquer campo seja alterado. O número da regra não pode ser alterado. A alteração de uma regra não implica em qualquer alteração nos demais arquivos mas poderá mudar substancialmente o comportamento do processo de inferência sobre a base de conhecimentos. Isto é conseqüência da alteração de cláusulas das regras que provocam um novo encadeamento entre as mesmas.

A eliminação de uma regra é sempre precedida pela solicitação de confirmação do usuário para que sejam evitados acidentes. A exemplo da opção de alteração, a eliminação de regras da base de conhecimentos também pode implicar em mudanças no comportamento das inferências sobre a mesma.

4.3.2.5 - INFERÊNCIAS

O SISREP oferece quatro opções de inferências sobre uma base de conhecimentos: progressivo passo-a-passo, progressivo direto, regressivo passo-a-passo e regressivo direto. Todas estas inferências utilizam-se do método de busca em profundidade apresentada na seção 3.3.1 deste trabalho. Adicionalmente, após a conclusão de uma inferência, será oferecido ainda um conjunto de informações (*justificador*) que são: o comportamento de uma regra específica durante a busca da solução; as cláusulas acumuladas, deduzidas e questionadas durante o processo (*lista de verdades*); o conjunto de regras utilizadas; o caminho da solução do algoritmo de busca e finalmente a opção de desprezar a solução encontrada para que outra seja pesquisada partindo-se do estado presente da busca.

A inferência progressiva implementada corresponde àquela apresentada na seção 3.2.1 deste trabalho. A opção passo-a-passo fará com que o SISREP mostre ao longo do processamento o número da regra que está sendo avaliada a cada passo permitindo que o usuário acompanhe o comportamento do algoritmo. Já a opção progressivo-direto fará a mesma pesquisa sem que o usuário seja informado do seu andamento.

A inferência regressiva corresponde àquela apresentada na seção 3.2.2 deste trabalho. Também neste caso são oferecidas as opções passo-a-passo e direto.

Após uma conclusão ter sido alcançada e apresentada, o usuário poderá solicitar informações relacionadas ao comportamento de uma regra específica ao

longo da busca. A ferramenta responderá então se a regra chegou a ser avaliada no processo; se a regra foi avaliada mas os questionamentos diretos ao usuário apresentaram respostas que negaram a regra, ou ainda, se a lista de verdades inviabilizou a utilização da regra com base em deduções decorrentes de outras regras.

A opção de mostrar as cláusulas deduzidas e questionadas fará com que o SISREP apresente a lista de verdades formada ao longo da busca. São mostradas as cláusulas (atributos, predicados e valores) acumuladas.

Concluída uma inferência com sucesso o usuário terá acesso ainda ao conjunto de regras que foram utilizadas no processo. Neste caso são mostradas todas as regras que chegaram a ser utilizadas, mesmo aquelas que estavam em caminhos que não levaram à solução. Se o usuário estiver interessado somente no conjunto de regras que compuseram o caminho solução, o SISREP também oferece esta alternativa.

A última opção apresentada é a busca de uma solução alternativa partindo-se da solução já alcançada. Neste caso o algoritmo de busca é novamente acionado. A lista de verdades formada até então é reutilizada e a situação de cada uma das regras definida na pesquisa anterior é respeitada.

4.3.3 - PRINCIPAIS PREDICADOS UTILIZADOS

Todos os programas do SISREP foram escritos utilizando-se a linguagem de programação ARITY/PROLOG. Estes programas são compostos por predicados internos da linguagem que são desenvolvidos pelo próprio fornecedor, e acompanham o compilador, e ainda os predicados que são construídos pelo programador.

4.3.3.1 - PREDICADOS INTERNOS

O ARITY/PROLOG oferece um conjunto de predicados que podem ser utilizados pelo programador. As principais funções destes predicados estão relacionadas as operações de entrada e saída de dados, gerência do banco de dados, utilização da tela, manipulação de strings, criação de menus e interface com o sistema operacional. No desenvolvimento do SISREP estes predicados foram largamente utilizados. Os predicados mais comuns e suas ações serão mostrados a seguir, agrupados por funções.

a) PREDICADOS RELACIONADOS À ENTRADA E SAÍDA DE DADOS

read(T) - recebe do teclado o termo T.

write(T) - mostra o termo T no monitor.

Keyb(A,S) - retorna em A o código ASCII e em S o código SCAN de uma tecla digitada.

nl - provoca o salto para uma nova linha.

create(H,F) - cria e abre um arquivo F associando H como sendo o seu nome lógico.

open(H,F,A) - abre o arquivo existente F para o modo de acesso A associando H como o seu nome lógico.

read(H,T) - recebe o termo T do arquivo de nome lógico H.

write(H,T) - escreve o termo T no arquivo de nome lógico H.

close(H) - fecha o arquivo de nome lógico H.

b) PREDICADOS RELACIONADOS À MANIPULAÇÃO DO BANCO DE DADOS

recorda(K,T,R) - adiciona T como o primeiro termo sob a chave K no banco de dados. R retorna o número de referência atribuído ao termo adicionado.

recordz(K,T,R) - adiciona T como o último termo sob a chave K no banco de dados. R retorna o número de referência atribuído ao termo adicionado.

recorded(K,T,R) - retorna em T um termo e em R a referência correspondente a um termo armazenado sob a chave K no banco de dados.

instance(R,T) - retorna em T o termo associado à referência R no banco de dados.

nref(R,N) - retorna em N a referência do termo seguinte ao termo associado à referência R no banco de dados.

nth_ref(K,N,R) - retorna em R a referência do N-ésimo termo armazenado sob a chave K no banco de dados.

erase(R) - elimina o termo do banco de dados associado a referência R.

eraseall(K) - elimina todos os termos do banco de dados armazenados sob a chave K.

replace(R,T) - substitui o termo associado à referência R pelo termo T no banco de dados.

c) PREDICADOS RELACIONADOS À UTILIZAÇÃO DA TELA

define_window(N,L,(Lse,Cse),(Lid,Cid),(Wa,Ba)) - define uma janela de nome N com cabeçalho L que inicia na linha número Lse e na coluna número Cse do monitor e termina na linha número Lid e coluna Cid do monitor. Wa define a cor da janela e Ba o tipo e cor das bordas.

current_window(V,N) - coloca a janela N sobre a janela V na tela. N passa a ser a janela corrente.

resize_window(R,C) - redimensiona a janela corrente, aumentando-a em R linhas e C colunas.

relabel_window(L) - altera o cabeçalho da janela corrente para L.

tmove(L,C) - desloca o cursor para a linha L e coluna C da janela corrente.

tget(L,C) - traz em L a linha e em C a coluna do posicionamento do cursor.

set_cursor(L,H) - muda o formato do cursor obedecendo uma tabela de formatos predeterminada pelo PROLOG.

wc(N,C) - escreve N vezes o caracter C sem que o cursor seja movimentado.

region_c((Li,Ci),(Lf,Cf),S) - traz em S um string que representa o conteúdo da tela da linha Ci e coluna Ci até a linha Lf e coluna Cf.

cls - limpa a tela.

d) PREDICADOS PARA A MANIPULAÇÃO DE STRINGS

string_search(Ss,S,L) - traz em L a posição de início do substring Ss no string S. O predicado falha se o substring não estiver contido no string.

substring(Is,N,C,Os) - extrai o substring Os do string Is iniciando na posição N. O substring extraído terá C caracteres.

string_length(S,C) - retorna em C o comprimento do string S.

concat(S1,S2,Sr) - retorna em Sr a concatenação dos strings S1 e S2.

int_text(N,T) - converte o string T no número N, ou vice-versa, dependendo de qual das variáveis estiver instanciada.

e) PREDICADOS RELACIONADOS À UTILIZAÇÃO DE MENUS

begin_menu(N,L,colors((Tn,Tr),(Pm,Pr),(Gn,Gr),(A,B))) - inicia a definição de um menu que será encerrado pelo predicado *end_menu(N)*. O menu definido será identificado por N. L determina o comprimento do menu. As variáveis Tn,Tr,Pm,Pr,Gn,Gr estão relacionadas às cores dos itens do menu. A e B definem as cores da borda e das letras para a opção direta. Os itens que compõem o menu são especificadas imediatamente abaixo de begin_menu. A definição de menus deve ser feita em um arquivo diferente daquele que contém o programa que o usa.

send_menu_msg(activate(N,(L,C)),Rv) - ativa o menu identificado por N. L e C definem respectivamente a linha e coluna onde o menu deve ser colocado na tela. Rv é a variável que retorna o item do menu selecionado pelo usuário.

f) PREDICADOS DE INTERFACE COM O SISTEMA OPERACIONAL

consult(F) - incorpora as cláusulas existentes no arquivo F ao banco de dados.

date(A,M,D) - retorna em A o ano, em M o mês e em D o dia da data corrente.

time(H,M,S,D) - retorna em H a hora, em M os minutos, em S os segundos e em D os décimos de segundo da hora corrente.

Os predicados internos, acima descritos, são os mais utilizados pelo SISREP. Além destes o ARITY/PROLOG oferece dezenas de outros que não foram citados. Para o conhecimento de todos os predicados internos disponibilizados deve ser consultado o manual da linguagem pois não seria possível relacioná-los neste trabalho.

4.3.3.2 - PREDICADOS CRIADOS

A seguir serão mostrados alguns predicados criados no SISREP. Estes predicados estão entre as principais rotinas implementadas no trabalho. Inicialmente é descrita a finalidade do predicado e em seguida o seu código e o de outros predicados diretamente relacionados ao mesmo.

definicoes - este predicado define todas as janelas utilizadas pela ferramenta. Algumas destas janelas são reutilizadas ao longo dos programas, ou seja, sobre elas são feitas operações de redimensionamento, alteração de cabeçalho e cores. Isto permite que o número de janelas seja relativamente pequeno.

```
definicoes :- define_window(wabre,'',(00,00),(24,79),(7,-47)),
              define_window(wnova,' CRIAR BASE ',(4,5),(21,75),(7,-47)),
              define_window(wnm_base,' nome ',(9,8),(11,30),(112,-47)),
              define_window(wdsc_base,'descricao',(15,8),(17,70),(112,-47)),
              define_window(watb,'',(4,5),(19,75),(7,-47)),
              define_window(wdsc_atb,'',(10,9),(12,46),(112,47)),
              define_window(wtipo,' tipo ',(10,55),(13,71),(112,47)),
              define_window(wdsc_val,'',(10,29),(12,51),(112,47)),
              define_window(winf,'',(00,00),(24,79),(112,-47)),
              define_window(wregra,'',(1,0),(20,79),(7,47)),
              define_window(wresp,'',(5,6),(15,44),(7,47)),
              define_window(wacp,' PROCESSANDO ',(5,50),(8,72),(7,47)).
```

confirma(L,C) - criado para ser utilizado em todas as operações em que foi incluído o pedido de confirmação ao usuário. L e C determinam, respectivamente, o número da linha e da coluna onde serão colocados os botões "confirma" e "cancela". O predicado *sim_nao(X,Y,Z)* é chamado para gerenciar o movimento do cursor. Quando o botão "cancela" é selecionado o predicado *confirma* falha.

```
confirma(L,C) :- Ll is L + 6, Cc is C + 33,
                 create_popup('',(L,C),(Ll,Cc),(7,47)),tmove(2,4),
                 wa(10,40),write(' CONFIRMA'),tmove(2,18),
                 wa(10,47),write(' CANCELA'),tmove(1,4),wa(10,40),
                 tmove(3,4),wa(10,40),tmove(1,18),wa(10,47),
                 tmove(3,18),wa(10,47),tmove(2,19),keyb(X,Y),
                 sim_nao(19,X,Y).

sim_nao(5,13,_)  :- exit_popup,!.
sim_nao(19,13,X) :- exit_popup,!,sim_nao(111,99,99).
sim_nao(_,27,_)  :- exit_popup,!,sim_nao(111,99,99).
sim_nao(5,9,15)  :- !,sim_nao(5,0,77).
sim_nao(5,0,77)  :- tmove(2,5),wa(09,40),tmove(2,19),wa(09,47),!,
                    keyb(X,Y),sim_nao(19,X,Y).
sim_nao(19,0,15) :- !,sim_nao(19,0,75).
sim_nao(19,0,75) :- tmove(2,19),wa(09,40),tmove(2,5),wa(09,47),!,
                    keyb(X,Y),sim_nao(5,X,Y).
sim_nao(C,S,Z)   :- tmove(2,C),!,keyb(X,Y),sim_nao(C,X,Y).
```

base_nova - criado para executar as tarefas relacionadas à criação de uma nova base de conhecimentos. Inicialmente ele coloca a janela apropriada na tela e depois chama o predicado *recebe_nova* que faz a leitura dos campos preenchidos, pede a confirmação e inclui esta base no SISREP. Caso não aconteça a confirmação a segunda cláusula do predicado fará o controle voltar para o menu inicial.

```
base_nova :- current_window(_,wnova),tmove(2,30),wa(35,47),
             relabel_window(' CRIAR BASE '),
             write('Para criar uma nova base, informe:'),
             tmove(4,34),wa(31,47),write('. O nome da base'),
             tmove(5,34),wa(31,47),write('. Uma breve descrição da base'),
             tmove(6,34),wa(31,47),write('. Preenchimento obrigat'),
             put(149),write('rio'),recebe_nova.
```

```
recebe_nova :- current_window(_,wdsc_base),cls,
               current_window(_,wnm_base),
               cls,set_cursor(6,7),tmove(0,0),pega(20),
               region_c((0,0),(0,19),Nsujo),limpa(Nsujo,Nlimpo),
               current_window(_,wdsc_base),tmove(0,0),pega(60),
               region_c((0,0),(0,59),Desc),confirma(05,40),limpa(Desc,Dl),
               atom_string(Anome,Nlimpo),atom_string(Adesc,Dl),
               not base_existe(bases,Anome,'BASE'),
               recordz(bases,Anome,_),recordz(Anome,Adesc,_),
               recordz(Anome,0,_),concat(Nlimpo,$+vlr$,Cx),
               recordz(Cx,1,_),recordz(Cx,'desconhecido',_),
               recordz(Anome,1,_),recordz(Anome,0,_),
            mostra_msg('BASE CRIADA - CADASTRE ATRIBUTOS/VALORES/REGRAS'),
               recorda(falha,ok,Ref),fail.
recebe_nova :- recorded(falha,I,R),eraseall(falha),
               current_window(_,wabre),entra_menu1.
recebe_nova :- recebe_nova.
```

mostra_msg(X) - predicado criado para mostrar uma mensagem ao usuário. Sua utilização é bastante grande pois ele é chamado por quase todas as rotinas do SISREP. X é um string que define a mensagem que deve ser mostrada.

```
mostra_msg(X) :- create_popup('',(19,10),(23,70),(7,-47)),
                 tmove(1,2),write(X),put(7),keyb(_,_),exit_popup.
```

pega(C) - utilizado para gerenciar e receber campos alfabéticos minúsculos. É chamado para receber o nome e a descrição de uma base, a descrição de atributos e a descrição de valores. C define o tamanho máximo de caracteres aceito pelo campo a ser preenchido. O predicado *veja(X,Y,Z)* gerência a digitação tecla-a-tecla do usuário.

```
pega(C) :- keyb(A,S),veja(C,A,S).
```

```
veja(_,27,1)   :- recorda(falha,esc,_),!,fail.
veja(_,0,15)   :- !,fail.
veja(_,9,15)   :- tget(X,Y),Y > 0,!.
veja(_,13,28)  :- tget(X,Y),Y > 0,!.
veja(C,_,75)   :- tget(I,J),X is J - 1, X > -1,tmove(I,X),put(32),
                  tmove(I,X),!,pega(C).
veja(C,_,_)    :- tget(I,J),J == C,!,pega(C).
veja(C,K,_)    :- K > 64, K < 91,Ks is K + 32,put(Ks),!,pega(C).
veja(C,K,_)    :- K > 96, K < 123,put(K),!,pega(C).
veja(C,32,_)   :- tget(I,J),J > 0,J < C,put(32),!,pega(C).
veja(C,K,_)    :- !,pega(C).
```

limpa(S,L) - utilizado para truncar um string no ponto em que forem encontrados dois brancos consecutivos. A criação deste predicado deveu-se ao fato de que a leitura de campos feita através do predicado interno *region_c* devolve um string que pode incluir espaços brancos que não foram utilizados pelo digitador. O predicado deve ser chamado com o primeiro argumento instanciado pois este conterá o string a ser truncado; no segundo argumento será devolvido o string final.

```
limpa(Sujo,Limpo) :- string_search($  $,Sujo,Local),
                     substring(Sujo,0,Local,Limpo),!.
limpa(Limpo,Limpo).
```

recebe_val(B,N) - utilizado para o cadastramento dos valores. O predicado para cadastramento de atributos é bastante semelhante.

```
recebe_val(Bb,N) :- Nn is N + 1,string_term(Sr,Nn),atom_string(At,Sr),
                    concat($ descricao: V$,At,Nwl),relabel_window(Nwl),
                    pega(20),region_c((0,0),(0,19),Vl),limpa(Vl,Lvl),
                    atom_string(Avl,Lvl),concat(Bb,$+vlr$,Chv),
                    not base_existe(Chv,Avl,'VALOR'),recordz(Chv,Nn,_),
                    recordz(Chv,Avl,_),nth_ref(Bb,3,Ref),replace(Ref,Nn),
                    mostra_msg('  CADASTRAMENTO DO VALOR EFETUADO'),fail.
recebe_val(Bb,N) :- recorded(falha,I,R),eraseall(falha),expunge,save,
                    current_window(_,wabre),adiante(Bb,2).
```

```
recebe_val(Bb,N) :- nth_ref(Bb,3,Ref),instance(Ref,Nn),
                    current_window(_,wdsc_val),cls,tmove(0,0),
```

muda_base(B) - criado para executar a alteração do nome de uma base. Inicialmente é mostrada a tela adequada e posteriormente é feita a leitura do novo nome. Na seqüência é feita a alteração do nome e das chaves sob as quais estão os atributos, valores e regras desta base. B é o argumento que contém o nome da base que será renomeada.

```
muda_base(Bb) :- current_window(_,wnova),cls,
                 relabel_window(' MUDAR NOME DA BASE '),
                 current_window(_,wnm_base),cls,set_cursor(6,7),
                 tmove(0,0),write(Bb),pega(20),
                 region_c((0,0),(0,19),Nsujo),
                 limpa(Nsujo,Nlimpo),atom_string(Anome,Nlimpo),
                 recorded(bases,Bb,R),
                 not mesmo_atb(bases,R,Anome,'BASE'),
                 Anome \== Bb,confirma(5,40),muda_tudo(Bb,Anome),
                 concat(Bb,$+atb$,Ch),concat(Anome,$+atb$,Chv),
                 concat(Bb,$+regras$,Cha),concat(Anome,$+regras$,Cha2),
                 muda_tudo(Ch,Chv),concat(Bb,$+vlr$,C),
                 concat(Anome,$+vlr$,Cc),muda_tudo(C,Cc),
                 replace(R,Anome),eraseall(Bb),eraseall(Ch),eraseall(C),
                 muda_tudo(Cha,Cha2),eraseall(Cha),
                 expunge,mostra_msg(' ALTERACAO EFETUADA '),
                 current_window(_,wabre),adiante(Anome,2).
muda_base(Bb) :- recorded(falha,I,R),eraseall(falha),
                 current_window(_,wabre),adiante(Bb,2).
muda_base(Bb) :- muda_base(Bb).
```

tirar_base(B) - elimina uma base de conhecimentos. Antes de executar a operação o predicado avisa das implicações relacionadas à eliminação de uma base e pede a confirmação ao usuário. B é o argumento que contém o nome da base a ser eliminada.

```
tirar_base(Bb) :- tmove(8,7),wa(12,240),write('ATENC'),put(142),
                  write('O !!'),tmove(10,3),write('TODOS OS ATRIBUTOS,'),
                  tmove(11,4),write('VALORES E REGRAS'),tmove(12,9),
                  write ('DA BASE'),string_length(Bb,Cp),N is 26 - Cp,
                  Nn is N // 2,tmove(13,Nn),write(Bb),tmove(14,4),
                  write ('SER'),put(142),write('O ELIMINADOS.'),
                  put(7) ,put(7),confirma(9,27),concat(Bb,$+atb$,Ch),
                  concat(Bb,$+vlr$,Chv),recorded(bases,Bb,R),erase(R),
                  eraseall(Ch),eraseall(Chv),eraseall(Bb),tmove(8,1),
                  wc (25,32),nl,nl,wc(25,32),nl,concat(Bb,$+regras$,Ky),
                  eraseall(Ky),wc(25,32),nl,wc(25,32),nl,wc(25,32),nl,
                  wc (25,32),expunge,mostra_msg(' BASE ELIMINADA '),cls,
                  escreve_menu1,entra_menu1.
tirar_base(Bb) :- tmove(8,1),wc(25,32),nl,nl,wc(25,32),nl,wc(25,32),
                  nl,wc(25,32),nl,wc(25,32),nl,wc(25,32),adiante(Bb,2).
```

tira_val(B,N) - elimina um valor da base. B determina o nome da base e N o número do valor a ser excluído. Os predicado *apoio* e *mostra_tudo* permitem mostrar todos os valores existentes na base, auxiliando nas situações em que o usuário não lembra o número de um valor. O predicado *exrg* fará a pesquisa no arquivo de regras para que aquelas que utilizavam este valor sejam eliminadas.

```
tira_val(Bb,$1$) :- mostra_msg(' IMPOSSIVEL EXCLUIR ESTE VALOR '),!.
tira_val(Bb,$$)  :- concat(Bb,$+vlr$,Chv),apoio(Chv,$V$,1,2,prov_),
                    mostra_tudo(mostratd,Op,K),
                    write(Op),eraseall(prov_),!,tira_val(Bb,Op).
```

```
tira_val(Bb,Nlp) :- int_text(Num,Nlp),concat(Bb,$+vlr$;Chv),
                    recorded(Chv,Num,R),nref(R,Rff),instance(Rff,Desc),
                    current_window(_,wdsc_val),cls,tmove(0,0),
                    set_cursor(6,7),relabel_window($ descricao $),
                    write(Desc),!,confirma(16,24),erase(R),erase(Rff),
                    exrg(Bb,Num),expunge,mostra_msg(' VALOR EXCLUIDO ').
tira_val(Bb,Nlp) :- int_text(Num,Nlp),concat(Bb,$+vlr$,Chv),
                    not recorded(Chv,Num,R),
                    mostra_msg(' VALOR INEXISTENTE').
```

trata_val(B,N) - altera a descrição de um valor. B determina o nome da base e N o número do valor a ser alterado. O predicado para alteração da descrição dos atributos é bastante semelhante.

```
trata_val(Bb,$$)  :- concat(Bb,$+vlr$,Ch),apoio(Ch,$V$,1,2,prov_),
                     mostra_tudo(mostratd,Op,K),write(Op),
                     eraseall(prov_),!,trata_val(Bb,Op).
trata_val(Bb,$1$) :- mostra_msg(' IMPOSSIVEL ALTERAR ESTE VALOR '),!.
trata_val(Bb,Nlp) :- int_text(Num,Nlp),concat(Bb,$+vlr$,Chv),
                     recorded(Chv,Num,R),nref(R,Rff),instance(Rff,Desc),
                     current_window(_,wdsc_val),cls,tmove(0,0),
                     set_cursor(6,7),relabel_window($descricao$),
                     write(Desc),pega(20),region_c((0,0),(0,19),Novo),
                     limpa(Novo,Novlp),atom_string(Anovo,Novlp),
                     not mesmo_atb(Chv,Rff,Anovo,'VALOR'),
                     replace(Rff,Anovo),!,
                     mostra_msg(' ALTERACAO DO VALOR EFETUADA').
trata_val(Bb,Nlp) :- int_text(Num,Nlp),concat(Bb,$+vlr$,Chv),
                     not recorded(Chv,Num,R),
                     mostra_msg(' VALOR INEXISTENTE').
```

recebe_regra(Bb,N) - predicado utilizado para fazer o cadastramento das regras. O predicado *regradepura* é chamado para verificar se a regra está digitada corretamente e os predicados *mesmaregra* e *meio* verificam se uma regra não está sendo duplicada, ou seja, se esta regra já não existe. Bb contém o nome da base e N o número da última regra cadastrada nesta base.

```
recebe_regra(Bb,N) :- escreve_miolo,int_text(N,Tn),
                      concat($ INCLUIR REGRA $,Tn,Rot),
                      atom_string(Lb,Rot),
                      relabel_window(Lb),tmove(2,3),keyb(X,Y),
                      informa_num(X,Y),confirma(17,25),
                      busca_cls(2,A1,P1,V1),busca_cls(5,A2,P2,V2),
                      busca_cls(8,A3,P3,V3),busca_cls(11,A4,P4,V4),
                      busca_ctz(Cz),busca_vld(Vd),
                      regradepura([A1,A2,A3,A4]),concat([A1,P1,V1],C11),
                      concat([A2,P2,V2],C12),concat([A3,P3,V3],C13),
                      concat([A4,P4,V4],C14),concat(Bb,$+regras$,Ky),
                      not mesmaregra(C11,C12,C13,C14,Bb,0),
                      recordz(Ky,N,_),recordz(Ky,sitpode,_),
                      recordz(Ky,C11,_),recordz(Ky,C12,_),
                      recordz(Ky,C13,_),recordz(Ky,C14,_),
                      recordz(Ky,Cz,_),recordz(Ky,Vd,_),Nss is N + 1,
                      nth_ref(Bb,4,Ref),replace(Ref,N),
                      mostra_msg(' CADASTRAMENTO DA REGRA EFETUADO'),
                      recebe_regra(Bb,Nss).
recebe_regra(Bb,N) :- recorded(falha,saiu,K),eraseall(falha),
                      eraseall(pra_),eraseall(prv_),eraseall(pas_),
                      current_window(_,wabre),expunge,save,adiante(Bb,2).
recebe_regra(Bb,N) :- recebe_regra(Bb,N).

mesmaregra(C1,C2,C3,C4,Bb,Rf) :- concat(Bb,$+regras$,Ch),
                                 recorded(Ch,N,R),nref(R,Rs),nref(Rs,Rc1),
                                 instance(Rc1,C1),nref(Rc1,Rc2),
                                 nref(Rc2,Rc3),nref(Rc3,Rc4),
                                 instance(Rc4,C4),instance(Rc2,Cm2),
                                 instance(Rc3,Cm3),Rf \== R,
                                 meio(C2,C3,Cm2,Cm3),int_text(N,Nt),
                                 concat($REGRA JA EXISTENTE - N°$,Nt,Msg),
                                 mostra_msg(Msg),!.
```

```
meio(C2,C3,C2,C3).
meio(C2,C3,C3,C2).
```

excluir_regra(B) - criado para eliminar uma regra da base de conhecimentos B. O predicado *qual_regra* é chamado para receber o número da regra a ser eliminada.

```
excluir_regra(Bb) :- qual_regra(Bb,Rr),escreve_miolo,mostra_regra(Rr,Bb),
                     confirma(17,25),tira(Rr),
                     mostra_msg('  REGRA EXCLUIDA'),fail.
excluir_regra(Bb) :- recorded(falha,nada,_),eraseall(falha),expunge,
                     current_window(_,wabre),adiante(Bb,2).
excluir_regra(Bb) :- trata_ret(Bb,excluirre).

qual_regra(Bb,Rr) :- keyb(X,Y),recebe_num(5,X,Y),
                     region_c((2,5),(2,8),Num),
                     limpa(Num,Nlp),int_text(N,Nlp),
                     concat(Bb,$+regras$,B),tira_regra(B,N,1,Rr).
```

vencidas(Bb) - predicado utilizado para marcar as regras que estejam com o prazo de validade ultrapassado e portanto não podem ser consideradas no processo de inferência. Este predicado é chamado todas as vezes que uma inferência for executada e ainda quando o usuário utiliza a opção de verificação de regras vencidas. O predicado *risca* faz a marca na regra.

```
vencidas(Bb) :- tudook(Bb),date(date(Y,Mt,Dy)),time(time(H,M,S,D)),
                concat(Bb,$+regras$,Ch),recorded(Ch,sitpode,Rs),
                nref(Rs,Rc1),nref(Rc1,Rc2),nref(Rc2,Rc3),nref(Rc3,Rc4),
                nref(Rc4,Rcz),nref(Rcz,Rvd),instance(Rvd,Vd),
                substring(Vd,0,8,Dt),substring(Vd,8,8,Hr),
                desmembra(Dt,Dr,Mr,Ar),desmembra(Hr,Hrr,Mir,Sr),
                Q is Y - 1900,
                risca(Rs,[Q,Ar,Mt,Mr,Dy,Dr,H,Hrr,M,Mir,S,Sr]),fail.
vencidas(Bb).
```

infere_normal(Bb,X) - este predicado executa o processo de inferir sobre uma base de conhecimentos. O predicado associado *menor_pode* procura por uma regra de partida quando do início ou quando uma busca não levou a qualquer solução e deve ser reiniciada. O predicado *ligado* faz a seleção do conjunto de regras que estão encadeadas com a regra em questão incluindo-as na fila abertos (chamada internamente de aglh_). O predicado *segue* verifica se uma conclusão foi alcançada ou se o processo deve seguir. *montasoluc* faz a apresentação da solução encontrada para que a mesma seja apresentada ao usuário.

```
infere_normal(Bb,X) :- menor_pode(Bb,Rl,X),recorded(aglh_,C,Y),
                       segue(Bb,Rl,K),ligado(Bb,C,Rr,K,Rl,X),
                       cconclusivo(Bb,Rr,Z),montasoluc(Bb,Rl),
                       put(7),put(7),msfim(Bb,Rr),infextra(Rr,Bb).
infere_normal(Bb,X) :- cls,tmove(2,1),relabel_window(''),
                       write('IMPOSSIVEL CONCLUIR SOBRE A BASE '),
                       tmove(3,1),put(7),write(Bb),tmove(8,0),
                       write('<digite qualquer tecla p/ continuar>'),
                       keyb(_,_).

menor_pode(Bb,Rr,X) :- eraseall(aglh_),concat(Bb,$+regras$,B),
                         recorded(B,sitpode,Rf),pref(Rf,Rr),
                         responde(Bb,Rr,C,X),expunge,recordz(aglh_,C,_),
                         replace(Rf,npode).

segue(Bb,Rl,1) :- cconclusivo(Bb,Rl,X).
segue(Bb,Rl,0).

ligado(Bb,C,Rl,1,Rl,G).
ligado(Bb,C,Rr,0,_,G) :- concat(Bb,$+regras$,B),recorded(B,sitpode,Rs),
                         pref(Rs,Rr),cconclusivo(Bb,Rr,1),esta_c(Rs,C),
                         responde(Bb,Rr,X,G),recordz(aglh_,X,_),
                         replace(Rs,npode).
ligado(Bb,C,Rr,0,_,G) :- concat(Bb,$+regras$,B),recorded(B,sitpode,Rs),
                         esta_c(Rs,C),pref(Rs,Rr),responde(Bb,Rr,X,G),
                         recordz(aglh_,X,_),replace(Rs,npode).
```

begin_menu - mostra um exemplo da definição de um dos menus do SISREP.

```
begin_menu(menu1,78,colors((112,7),(112,7),(112,7),(112,47))).
     item($a~Tributos$,[item($~Incluir  $,incluirat),
                        item($~Alterar  $,alterarat),
                        item($~Excluir  $,excluirat)]).
     item($~Valores$,[item($~Incluir  $,incluirva),
                      item($~Alterar  $,alterarva),
                      item($~Excluir  $,excluirva)]).
     item($~Regras$,[item($~Incluir  $,incluirre),
                     item($~Alterar  $,alterarre),
                     item($~Excluir  $,excluirre)]).
     item($~Parametros$,[item($~Nome da base  $,nomebase),
                         item($~Descricao  $,descricao),
                         item($~Eliminar base  $,eliminar)]).
     item($v~Oltar$,[item($con~Firma  $,confirma),
                     item($~Cancela  $,cancela)]).
                     item(right($~Ajuda$),[item($~Editor  $,editor),
aend_menu(menu1).
```

CAPÍTULO 5

5 - UTILIZAÇÃO DO SISREP

Conforme descrito no capítulo quatro, o SISREP - SISTEMA PARA REGRAS DE PRODUÇÃO é uma ferramenta para a criação de sistemas especialistas baseados em regras de produção. O sistema é de uso genérico, ou seja, ele não é orientado para nenhuma área específica. Ele foi construído de forma a permitir que mesmo usuários que não possuam conhecimentos na área de informática possam utilizá-lo. O sistema é inteiramente estruturado por menus o que permite uma fácil operação.

O SISREP deve ser instalado em computadores da família PC que possuam pelo menos 640 Kb de memória real, disponham de pelo menos 400 Kb de espaço livre em disco rígido e possuam uma unidade de disco flexível. Para se utilizar o sistema basta inserir o disquete com o programa na unidade A e digitar copy A:*.* . O sistema será então copiado para o disco rígido e estará disponível para utilização.

Depois de copiado o sistema deve ser digitada, inicialmente, a palavra SISREP para que o programa seja carregado. Serão então colocadas na tela duas janelas de abertura do sistema, conforme mostram as figuras 5 e 6. Na seqüência serão apresentados automaticamente os menus através dos quais se tem acesso a todas as opções oferecidas pelo sistema.

figura 5 - Tela Inicial do SISREP

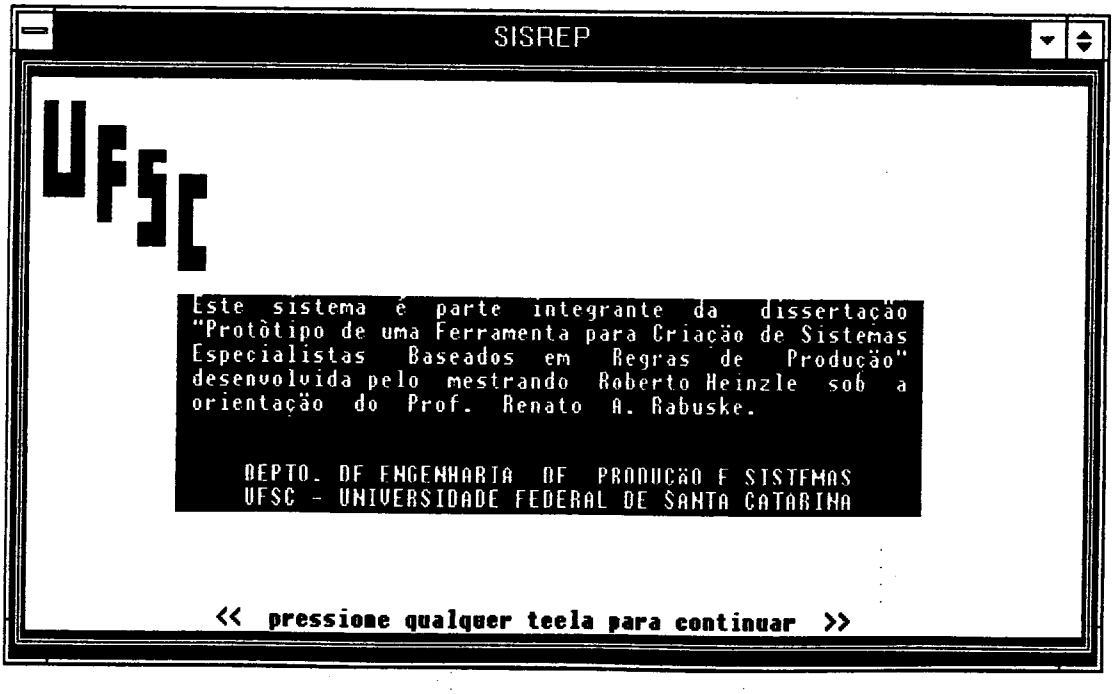

figura 6 - Tela de Abertura do SISREP

5.1 - MENUS DO SISTEMA

O SISREP é formado por três menus básicos. O primeiro deles é o chamado menu principal que é mostrado na figura 7. Neste menu o usuário pode optar entre utilizar uma base de conhecimentos já existente ou então criar uma base nova. São oferecidas ainda as opções SAIR E AJUDA. SAIR serve para terminar a seção de utilização do sistema e AJUDA fará com que um conjunto de informações relacionadas a operação do SISREP sejam mostradas na tela.

figura 7 - Menu Principal

Se for selecionada a opção do menu principal "criar nova base" o sistema mostrará, na seqüência, a tela apresentada na figura 8. Nela é solicitado ao usuário que defina um nome para esta nova base de conhecimentos e também uma breve descrição relacionada ao seu conteúdo ou finalidade. Depois de preenchidos estes campos será mostrado um lembrete para que sejam cadastrados os atributos, valores e regras da base criada e, então, volta a ser apresentado o menu principal.

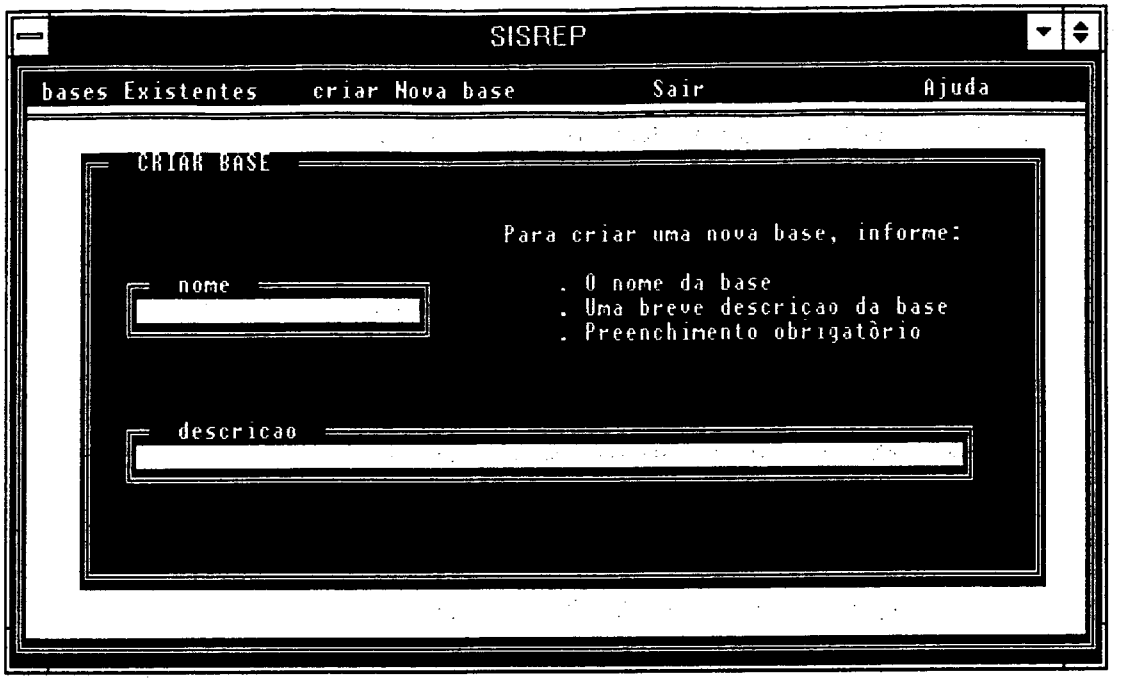

figura 8 - Tela para Criação de Base de Conhecimentos

Se a opção selecionada no menu principal for a de bases já existentes o sistema mostrará o segundo menu básico. Nele estarão relacionadas todas as bases de conhecimentos existentes e portanto disponíveis para utilização. Neste mesmo menu o usuário deve fazer ainda uma segunda opção relacionada ao tipo de operação que pretende efetuar sobre a base selecionada. As operações oferecidas no menu podem ser verificadas na figura 9.

Se a opção "processar base" for selecionada o sistema solicitará também o tipo de inferência a ser executada. Estão disponíveis as opções progressivo passo-a-passo, progressivo direto, regressivo passo-a-passo, e regressivo direto. As opções progressivas são aquelas descritas na seção 3.2.1 deste trabalho. A opção passo-a-passo fará com que o sistema mostre o número de cada regra que está sendo avaliada permitindo ao usuário acompanhar o processo de raciocínio. As opções de inferências regressivas são aquelas descritas na seção 3.2.2 deste trabalho.

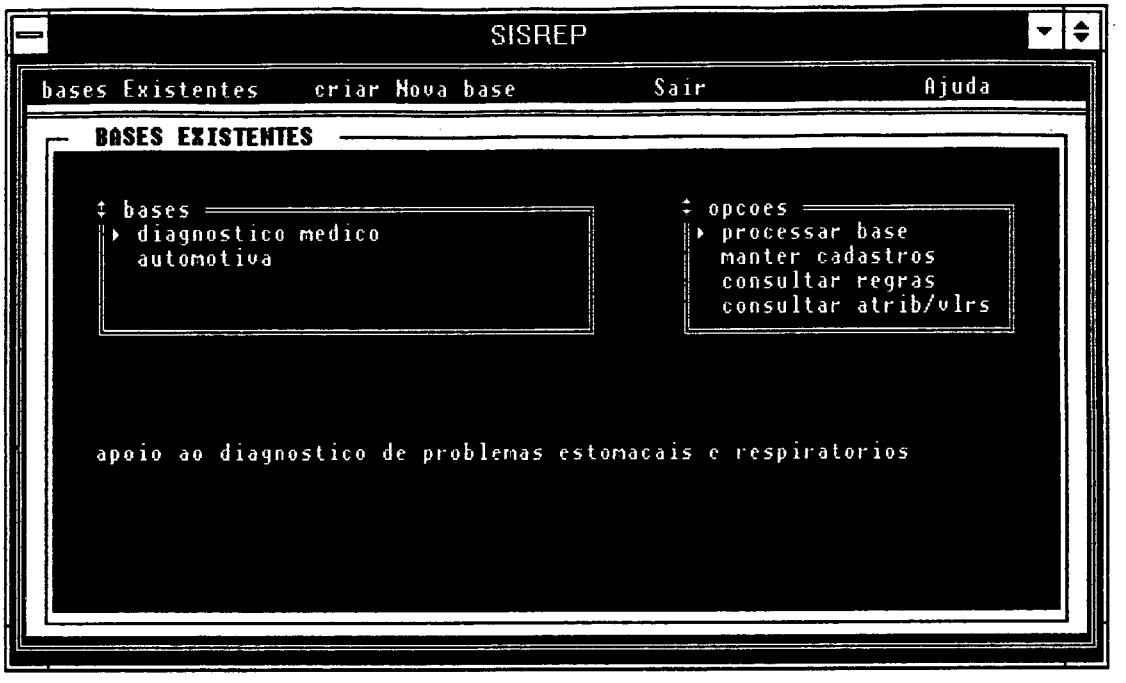

figura 9 - Menu das Bases Existentes

A opção "manter cadastros" do segundo menu básico (figura 9) permite ao usuário fazer a manutenção dos arquivos relacionados à base de conhecimentos em uso. Entende-se por manutenção de cadastros a opções de alterar o arquivo de atributos, valores, regras, descrição e nome da base e, ainda, a exclusão de uma base. Para que o usuário defina em qual dos arquivos se dará a manutenção, o sistema mostrará o terceiro menu básico. Este menu é mostrado na figura 10.

Cada uma das opções oferecidas pelo terceiro menu básico está relacionada ao próprio arquivo que se pretende alterar. Assim, por exemplo, a opção "atributos" permitiria alterar, incluir e excluir atributos. Já a opção "parâmetros" permitiria alterar a descrição de uma base, o nome de uma base e, ainda, a eliminação de uma base.

figura 10 - Menu para Manutenção de Cadastros

A opção "consultar regras", oferecida no segundo menu básico, permite ao usuário ter acesso as regras existentes numa base de conhecimentos. Já a opção "consultar atrib/vlrs" permite visualizar todos os possíveis valores de um atributo. Os possíveis valores de um atributo são definidos pelas regras existentes na base.

5.2 - CRIAÇÃO DE BASES DE CONHECIMENTO

5.2.1 - CADASTRAMENTO DA BASE

O primeiro passo a ser dado para a criação de uma base de conhecimentos no SISREP é o cadastramento desta base. Para este procedimento ser executado deve-se selecionar no menu principal (figura 7) a opção correspondente. O usuário deve, então, informar o nome e uma descrição para esta base. O nome de uma base pode possuir no máximo vinte letras e a sua descrição no máximo sessenta letras. Não são aceitos caracteres especiais e nem números.

A figura 8 mostra a tela que é apresentada pelo sistema para o cadastramento de uma base. Após o preenchimento dos campos desta tela é solicitada pelo sistema a confirmação da criação da base. Caso o usuário confirme a criação será mostrado na linha de mensagens, no rodapé do vídeo, um lembrete para que sejam cadastrados os atributos, valores e regras da base. Após esta mensagem o controle do programa volta ao menu principal.

5.2.2 - CADASTRAMENTO DE ATRIBUTOS

Todo atributo a ser utilizado em regras deve ser previamente cadastrado. Para efetuar esta operação o usuário deve selecionar a opção de bases existentes do menu principal (figura 7). No menu seguinte (figura 9) deve ser, então, selecionada a base de conhecimentos para a qual o atributo será criado e também a opção de manter cadastros. Na seqüência serão mostrados no vídeo os cadastros que podem ser acessados (figura 10). Deve-se selecionar a opção "atributos/incluir". Após estes procedimentos estarão colocados na tela os campos que permitem incluir um atributo no sistema. A figura 11 mostra a tela utilizada para esta operação.

O campo "descrição" deve ser então preenchido. O tamanho máximo de um atributo é de trinta e cinco letras. Não são permitidos números ou caracteres especiais. Sobre o campo em que será digitado o atributo aparece o número que será associado a ele pelo sistema. Também é solicitado a definição do tipo do atributo, se intermediário ou conclusivo. Inicialmente o sistema assume que o atributo é do tipo intermediário. Para que ele seja cadastrado como conclusivo, basta que o usuário utilize as teclas de movimentação do cursor para fazer com a palavra conclusivo seja ativada.

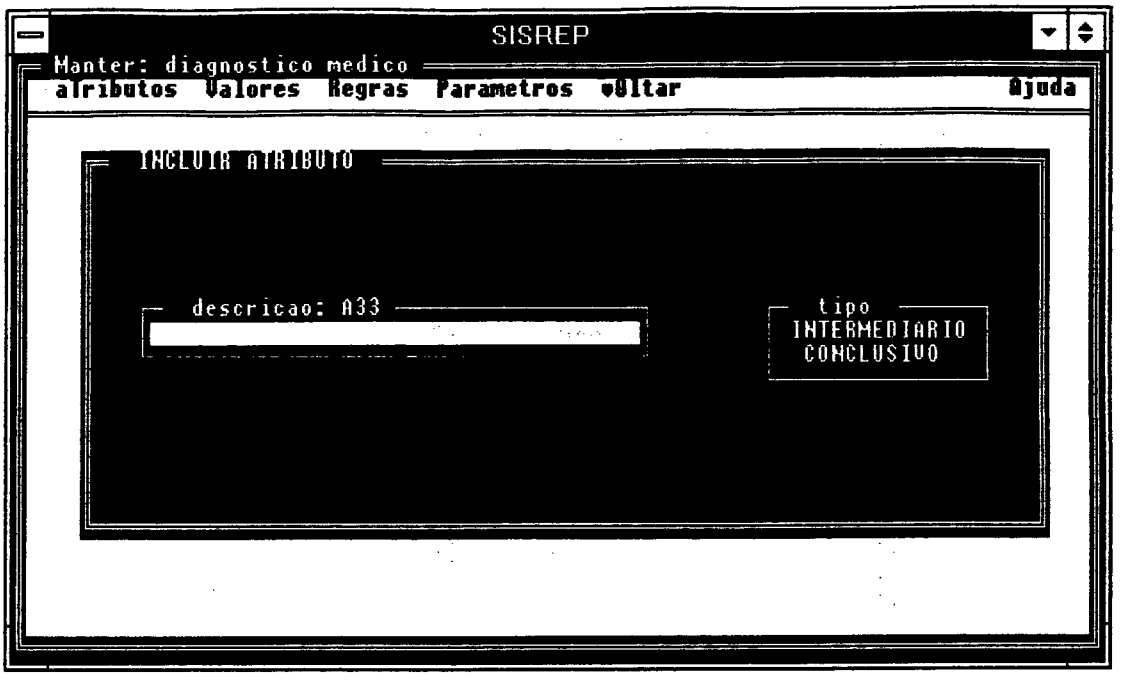

figura 11 - Tela para Cadastramento de Atributos

5.2.3 - CADASTRAMENTO DE VALORES

Todo valor a ser utilizado em regras deve ser previamente cadastrado. Para efetuar esta operação o usuário deve selecionar a opção de bases existentes do menu principal (figura 7). No menu seguinte (figura 9) deve ser, então, selecionada a base de conhecimentos para a qual o valor será criado e também a opção de manter cadastros. Na seqüência serão mostrados no vídeo os cadastros que podem ser acessados (figura 10). Deve-se selecionar a opção valor/incluir. Após estes procedimentos estarão colocados na tela os campos que permitem incluir um valor no sistema. A figura 12 mostra a tela utilizada para esta operação.

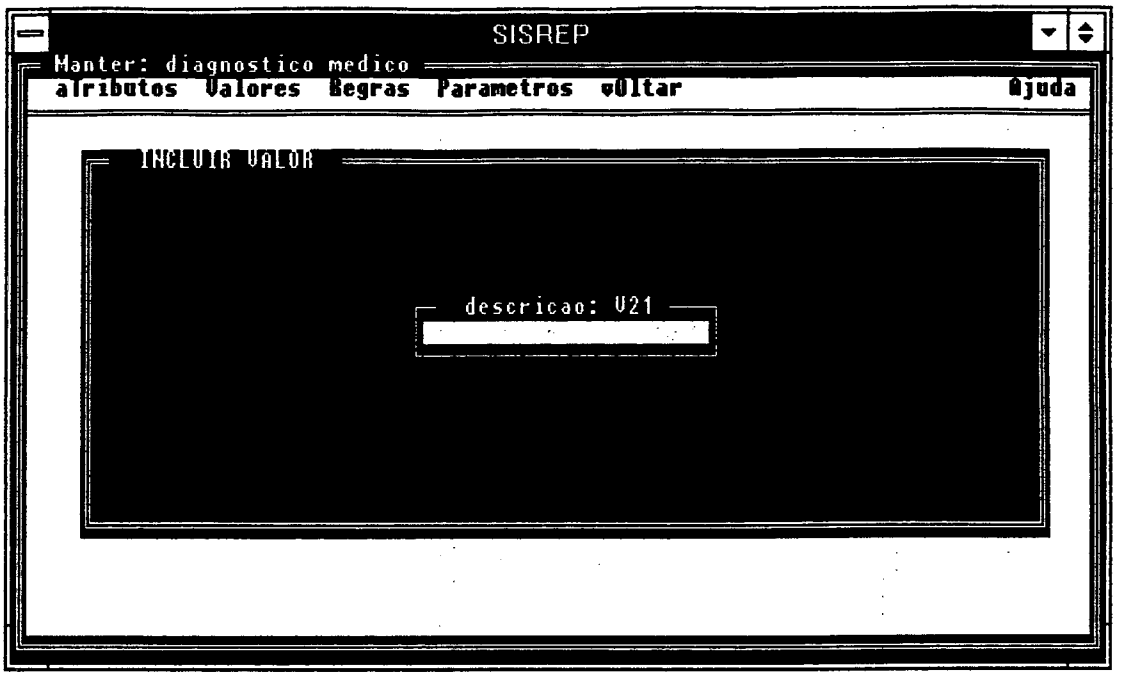

figura 12 - Tela para Cadastramento de Valores

O campo descrição deve ser preenchido. O tamanho máximo de um valor é de vinte letras. Não são permitidos números ou caracteres especiais. Sobre o campo em que será digitado o valor aparece o número que será associado a ele pelo sistema. Sempre que uma base de conhecimentos é criada o SISREP faz automaticamente o cadastramento do valor de número um cuja descrição é "desconhecido".

5.2.4 - CADASTRAMENTO DE REGRAS

Para efetuar o cadastramento ou inclusão das regras no SISREP o usuário deve selecionar a opção de bases existentes do menu principal (figura 7). No menu seguinte (figura 9) deve ser, então, selecionada a base de conhecimentos para a qual a regra será criada e também a opção de manter cadastros. Na seqüência serão mostrados no vídeo os cadastros que podem ser acessados (figura 10). Deve-se selecionar a opção "regras/incluir". Após estes procedimentos estarão colocados na tela os campos que compõem uma regra. A figura 13 mostra a tela utilizada para esta operação.

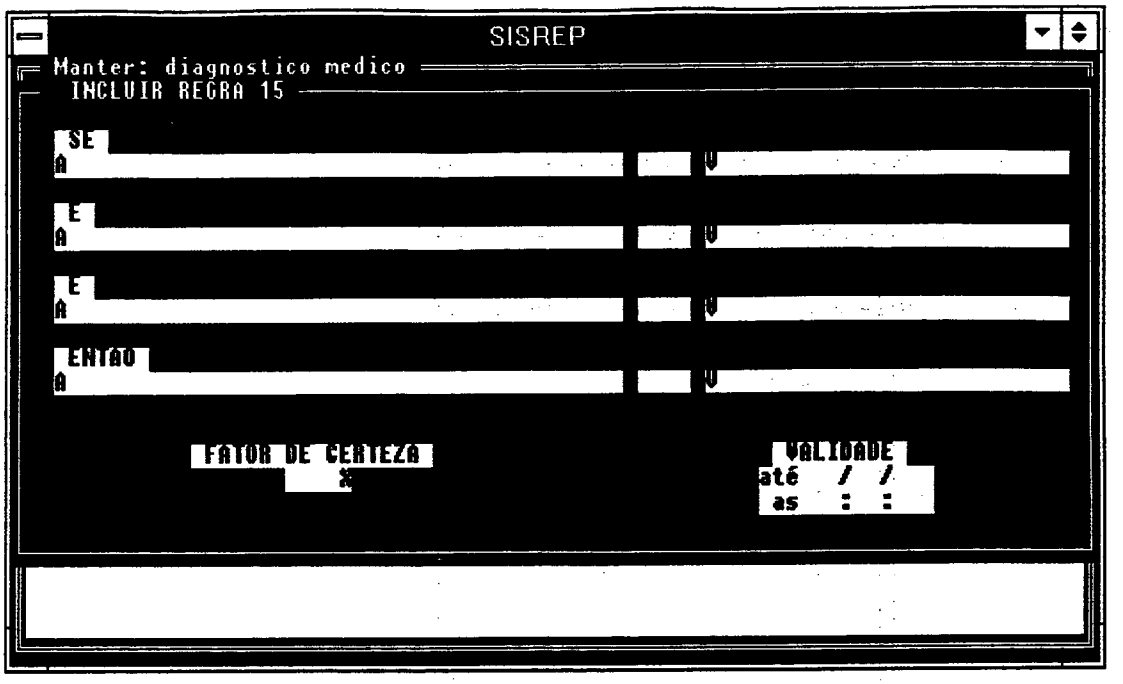

figura 13 - Tela para Cadastramento de Regras

Para cada uma das cláusulas deve ser informado o número do atributo, o predicado e o número do valor. Caso o usuário tenha esquecido o número de um atributo ou valor que pretende usar na regra, ele deve posicionar o cursor no campo a ser preenchido e teclar ENTER. Assim, será aberta uma janela onde serão mostrados todos os atributos e valores que podem ser usados naquele campo. O usuário faz então a sua escolha utilizando as teclas de movimentação de cursor para posicioná-lo na opção correta e digita ENTER. O SISREP preencherá automaticamente o campo da regra com a opção selecionada. Os atributos do tipo conclusivo não podem ser utilizados nas cláusulas antecedentes das regras.

Depois de preenchidas as cláusulas componentes da regra deve ser informado o fator de certeza e o prazo de validade. O fator de certeza é um número, entre um e cem, que representa o grau de certeza relacionado àquela regra enquanto o prazo de validade determina uma data na qual a regra deixa de ser verdadeira. Estes dois campos são de preenchimento opcional e se não utilizados fará com que o sistema assuma o número cem, para o fator de certeza, e validade permanente, para o prazo de validade.

5.3 - MANUTENÇÃO DE BASES EXISTENTES

5.3.1 - ALTERAÇÃO DO CADASTRO DE ATRIBUTOS E VALORES

O SISREP permite que o usuário altere e elimine atributos e valores cadastrados no sistema. Para efetuar esta operação deve-se selecionar a opção de bases existentes do menu principal. No menu que é então mostrado, deve-se selecionar a base com a qual se pretende trabalhar e também a opção "manter cadastros". Na seqüência deve-se, então, selecionar o item "atributos" ou "valores", conforme o arquivo que se pretende acessar.

Para a alteração ou eliminação de um atributo ou valor será solicitado o respectivo número deste atributo ou valor. Caso o usuário não lembre deste número, ele poderá simplesmente digitar ENTER com o campo vazio para fazer com que o sistema abra uma janela de apoio. Nela são mostrados todos os atributos e valores cadastrados, permitindo que um deles seja selecionado.

Quando for selecionado o número, o SISREP mostrará a descrição deste atributo ou valor nas telas apresentadas nas figuras 11 e 12, permitindo que a operação seja executada. Nos casos de tentativa de eliminação de atributos ou valores que são integrantes de regras, e portanto não deveriam ser eliminados, o sistema lembrará este fato ao usuário e pedirá a confirmação da operação. Caso haja a confirmação, as regras que usam o atributo ou valor que está sendo eliminado também serão automaticamente eliminadas pelo SISREP.

O valor número um, que é automaticamente cadastrado, não pode ser alterado nem eliminado. Já um atributo que é usado numa cláusula antecedente de qualquer uma das regras não pode ter o seu tipo alterado de intermediário para conclusivo. Se esta alteração for tentada o sistema avisará este fato e pedirá a confirmação da

operação. Se houver a confirmação as regras que utilizavam este atributo nas suas cláusulas antecedentes serão eliminadas automaticamente da base de conhecimentos.

5.3.2 - ALTERAÇÃO DE REGRAS

O SISREP permite que o usuário altere e elimine qualquer uma das regras cadastradas no sistema. Para efetuar esta operação deve-se selecionar a opção "bases existentes" do menu principal. No menu seguinte deve-se selecionar a base com a qual se pretende trabalhar e também a opção de manter cadastros. Na seqüência deve-se, então, selecionar o item regras (figura 10).

Para a alteração ou eliminação de uma regra será solicitado o respectivo número desta regra. Caso o usuário não lembre deste número ele poderá simplesmente digitar ENTER com o campo vazio para fazer com que o sistema abra uma janela de apoio. Nela são mostrados todas as regras cadastradas, permitindo que uma delas seja selecionada.

Depois de definido o número, o SISREP mostrará a regra correspondente na tela apresentada na figura 13, permitindo que a operação seja executada. Nos casos de alteração, qualquer um dos campos componentes da regra podem ser revistos. Nos casos de eliminação de uma regra o sistema exigirá a confirmação da operação ao usuário.

5.3.3 - ALTERAÇÃO DE NOME E ELIMINAÇÃO DE BASES

O SISREP permite que o usuário altere o nome de uma base, a descrição da finalidade da base e ainda a elimine. Para efetuar estas operações deve-se selecionar a opção de bases existentes do menu principal. No menu seguinte deve-se selecionar a base com a qual se pretende trabalhar e também a opção de manter cadastros. Na seqüência deve-se, então, selecionar o item parâmetros (figura 10).

Quando o usuário fizer a seleção do item correspondente à eliminação de uma base, o sistema exigirá a confirmação da operação. O SISREP avisará que este procedimento eliminará todo o conjunto de atributos, valores e regras da base.

5.4 - EXECUÇÃO DE INFERÊNCIAS

Para disparar o processo de raciocínio sobre uma base de conhecimentos o usuário deve selecionar a opção de bases existentes do menu principal. No menu seguinte deve-se selecionar a base com a qual se pretende trabalhar e também a opção "processar". Na seqüência será mostrado na tela o menu apresentado na figura 14. Neste menu define-se a forma de encadeamento a ser adotada na inferência.

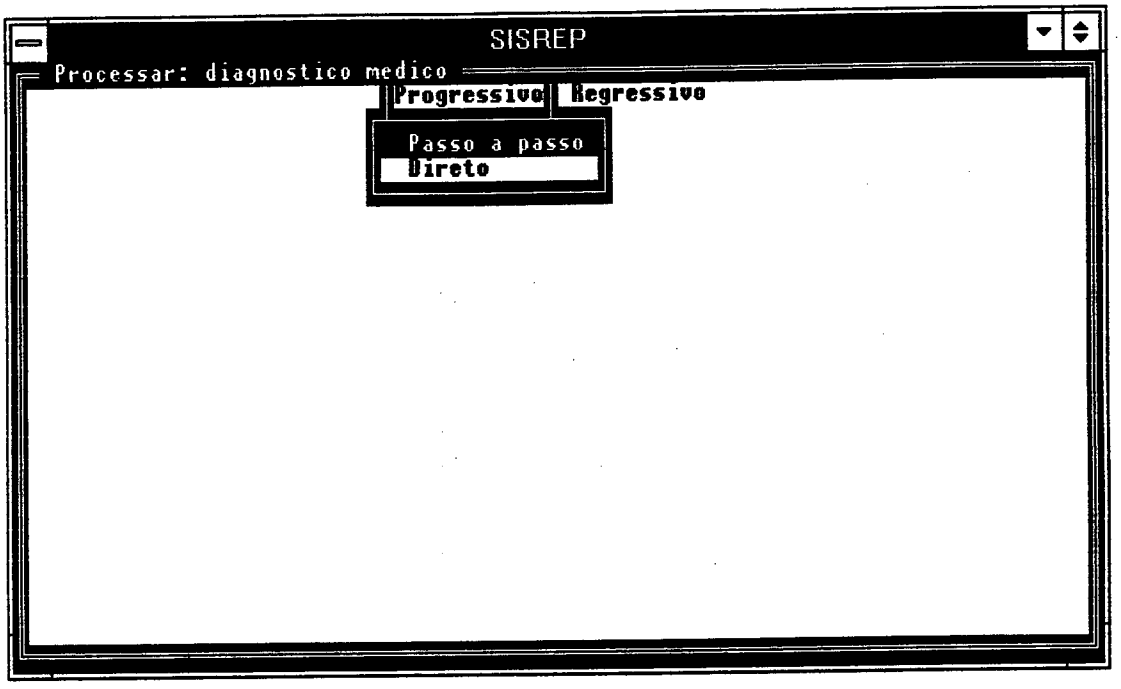

figura 14 - Menu de inferências

5.4.1 - RACIOCÍNIO PROGRESSIVO

As opções "progressivo passo-a-passo" e "progressivo direto" do menu, mostradas na figura 14, utilizam a técnica de encadeamento apresentada na seção 3.2.1 deste trabalho. No caso da opção passo-a-passo ser selecionada o usuário poderá acompanhar a ordem de seqüência de avaliação das regras. A figura 15 mostra a tela que será colocada no vídeo neste caso.

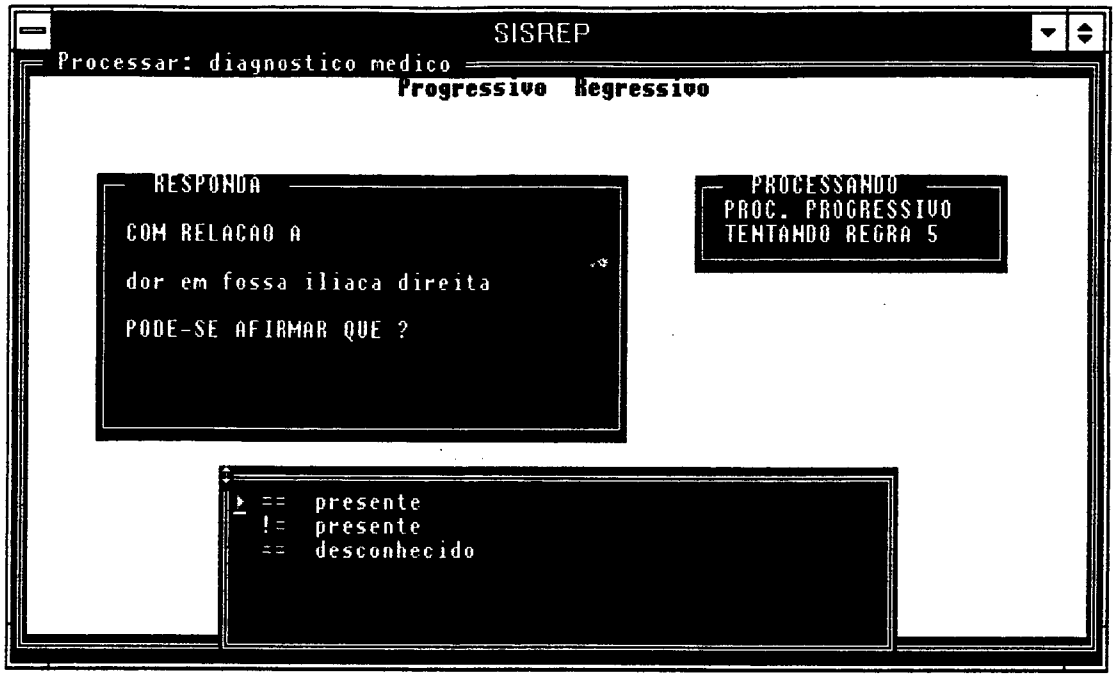

figura 15 - Tela de Inferência Passo-a-passo

A opção "progressivo direto" utilizará a mesma técnica de encadeamento da opção passo-a-passo sem, entretanto, mostrar ao usuário a ordem de avaliação das regras. A figura 16 mostra a tela que será colocada no vídeo neste caso.

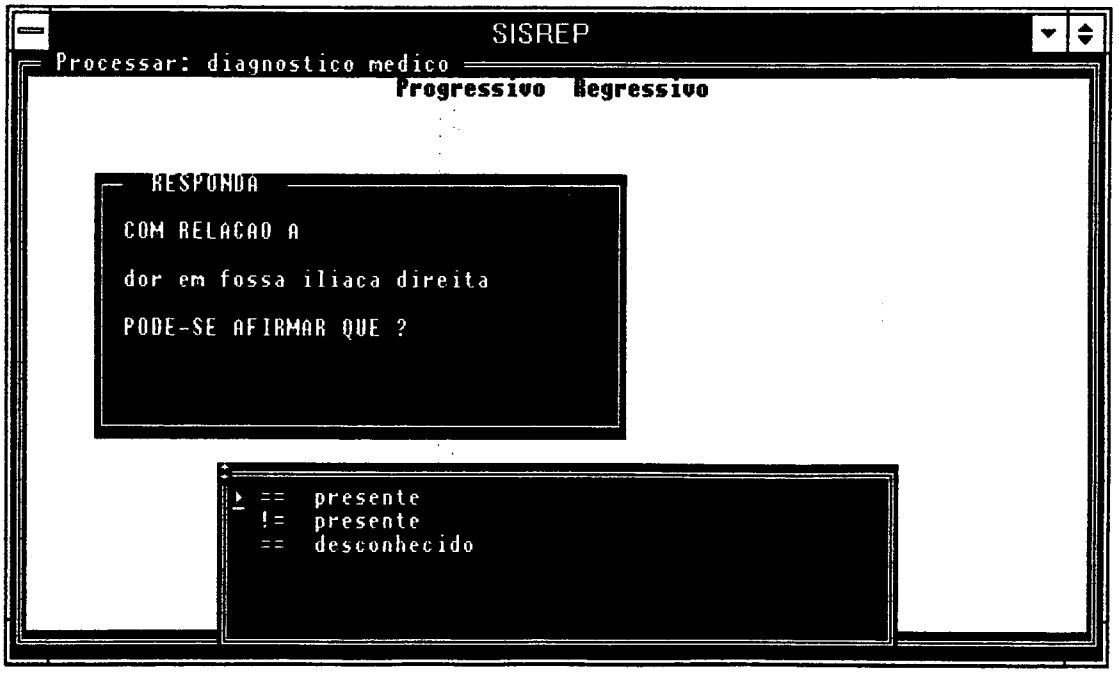

figura 16 - Tela de Inferência Direta

As figuras 15 e 16 mostram, também, o mecanismo de questionamento ao usuário durante o processo de raciocínio. Observa-se que as possíveis respostas a uma pergunta estão relacionadas em uma janela, fazendo com que o usuário tenha apenas que selecionar uma delas. Estas respostas são, na verdade, os possíveis valores que um atributo pode assumir baseado no próprio conjunto de regras da base de conhecimentos. Para todas as perguntas, no entanto, é adicionado o valor "desconhecido" como opção de resposta. O usuário pode selecionar esta opção quando não dispor da resposta à uma questão apresentada.

Adicionalmente, após a conclusão de uma inferência, será oferecido ainda um conjunto de informações (justificador) que podem ser acessadas através do menu mostrado na figura 17.

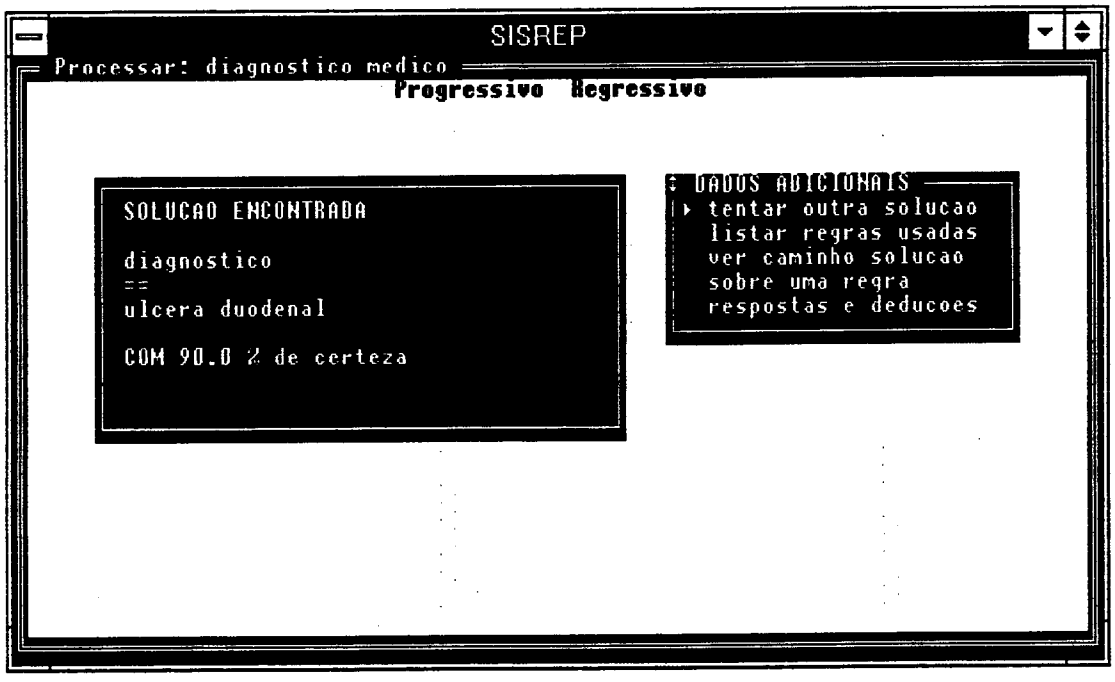

figura 17 - Menu do Justificador

A primeira opção deste menu, "tentar outra solução", fará com que o SISREP busque uma solução alternativa partindo-se da solução já alcançada. Neste caso o algoritmo de busca é novamente acionado. A lista de verdades formada até então é reutilizada e a situação de cada uma das regras definida na pesquisa anterior é respeitada.

A segunda opção do menu permite o acesso ao conjunto de regras que foram utilizadas no processo de raciocínio. Neste caso são mostradas todas as regras que chegaram a ser utilizadas, mesmo aquelas que estavam em caminhos que não levaram à solução. Se o usuário estiver interessado somente no conjunto de regras que compuseram o caminho solução ele deve selecionar a terceira opção do menu que é "ver caminho solução".

A quarta opção do menu, "sobre uma regra", permite solicitar informações relacionadas ao comportamento de uma regra específica ao longo da busca. O SISREP responderá, então, se a regra chegou a ser avaliada no processo; se a regra foi avaliada mas os questionamentos diretos ao usuário apresentaram respostas que negaram a regra, ou ainda, se a lista de verdades inviabilizou a utilização da regra com base em deduções decorrentes de outras regras.

A última opção do menu fará com que o SISREP apresente a lista de verdades formada ao longo da busca. São mostradas as cláusulas (atributos, predicados e valores) acumuladas através dos questionamentos ao usuário e das deduções feitas pelo sistema durante o processo de raciocínio.

5.4.2 - RACIOCÍNIO REGRESSIVO

As opções "regressivo passo-a-passo" e "regressivo direto" utilizam a técnica de encadeamento apresentada na seção 3.2.2 deste trabalho. No caso da opção "passo-a-passo" ser selecionada o usuário poderá acompanhar a ordem de seqüência de avaliação das regras.

A opção regressivo direto utilizará a mesma técnica de encadeamento da opção "passo-a-passo" sem, entretanto, mostrar ao usuário a ordem de avaliação das regras. A figura 16 mostra a tela que será colocada no vídeo neste caso. Também nas opções

de encadeamento regressivo é apresentado o menu de informações adicionais, relacionadas ao justificador, mostrado na figura 17 e descritas anteriormente.

5.5 - EXEMPLO DE UM SISTEMA ESPECIALISTA

5.5.1 - APRESENTAÇÃO DA APLICAÇÃO EXPERIMENTAL

Como forma de testar a viabilidade da ferramenta na construção de sistemas especialistas foi elaborada e implementada uma base de conhecimentos na área médica. Esta base permite o diagnóstico de algumas doenças comuns do aparelho digestivo e respiratório.

Para a elaboração da base de conhecimentos utilizou-se um total de trinta e um atributos intermediários, três atributos conclusivos, vinte valores e catorze regras em que estavam contidas trinta e oito cláusulas antecedentes. As conclusões possíveis de serem alcançadas, a partir desta base, permitem ao usuário determinar um diagnóstico, identificar a necessidade de tratamento ou intervenção cirúrgica e indicar a realização de exames suplementares como endoscopia digestiva, hemograma, R-X de tórax e ultrassom de abdômen.

5.5.2 - ALGUMAS REGRAS

A seguir são mostradas, como ilustração, quatro das regras que compõem a base de conhecimentos da aplicação experimental implementada. No APÊNDICE 1 é listado o conjunto completo dos atributos, valores e regras utilizadas.

Regra 1

SE dor epigastrica == existente

E azia == presente

E vômitos == freqüêntes

ENTÃO endoscopia digestiva alta == necessário

FATOR DE CERTEZA = 100

VALIDADE ATÉ 31/12/1999 AS 23:59:59

Regra 4

SE R-X de tórax == necessário

E consolidação == presente

E derrame pleural != presente

ENTÃO diagnóstico == pneumonia pneumocócica

FATOR DE CERTEZA = 80

VALIDADE ATÉ 31/12/1999 AS 23:59:59

Regra 8

SE dor cólica em hipocôndrio direito == presente

E ictericia == presente

E vômitos != ausente

ENTÃO ultrassom de abdômen superior == necessário

FATOR DE CERTEZA = 100

VALIDADE ATÉ 31/12/1999 AS 23:59:59

Regra 13

SE cefaléia == presente

E rinorréia purulenta == existente

E febre == alta

ENTÃO R-X de seios da face == necessário

FATOR DE CERTEZA = 100

VALIDADE ATÉ 31/12/1999 AS 23:59:59

5.5.3 - TESTES EFETUADOS

Para que o teste do sistema fosse o mais abrangente possível, utilizou-se todas as opções de todos os menus oferecidos pelo SISREP. Além do cadastramento, também foram verificadas as opções de alteração, exclusão e consultas de atributos, valores e regras.

Também as várias opções de inferências disponíveis no SISREP, e anteriormente descritas, foram exaustivamente testadas. As opções oferecidas no menu mostrado na figura 17 foram utilizadas em sua totalidade para cada uma das conclusões que eram mostradas pelo sistema.

5.5.4 - UMA INFERÊNCIA E SOLUÇÃO APRESENTADA

As figuras 18, 19, 20, 21 e 22 mostram seqüencialmente algumas das telas apresentadas pelo SISREP ao longo de um processo de raciocínio efetuado durante os testes. Todas as telas apresentadas são resultantes da seleção da opção de inferência progressiva passo-a-passo.

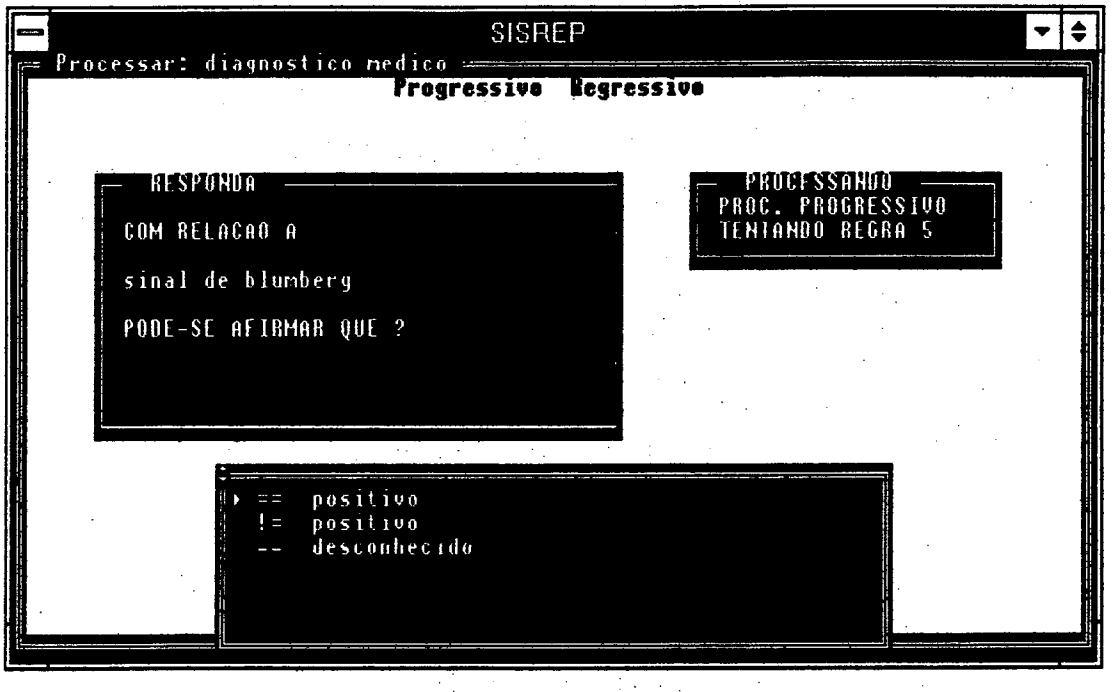

figura 18 - Exemplo de Questionamento do SISREP

As figuras 18 e 19 mostram duas das perguntas efetuadas ao usuário durante o processo de raciocínio. Observa-se que na parte inferior da tela estão listadas as possíveis respostas para as questões. Para a pergunta mostrada na figura 18 foi selecionada a resposta "== positivo" , e para a pergunta mostrada na figura 19 foi selecionada a resposta "== apendicite aguda".

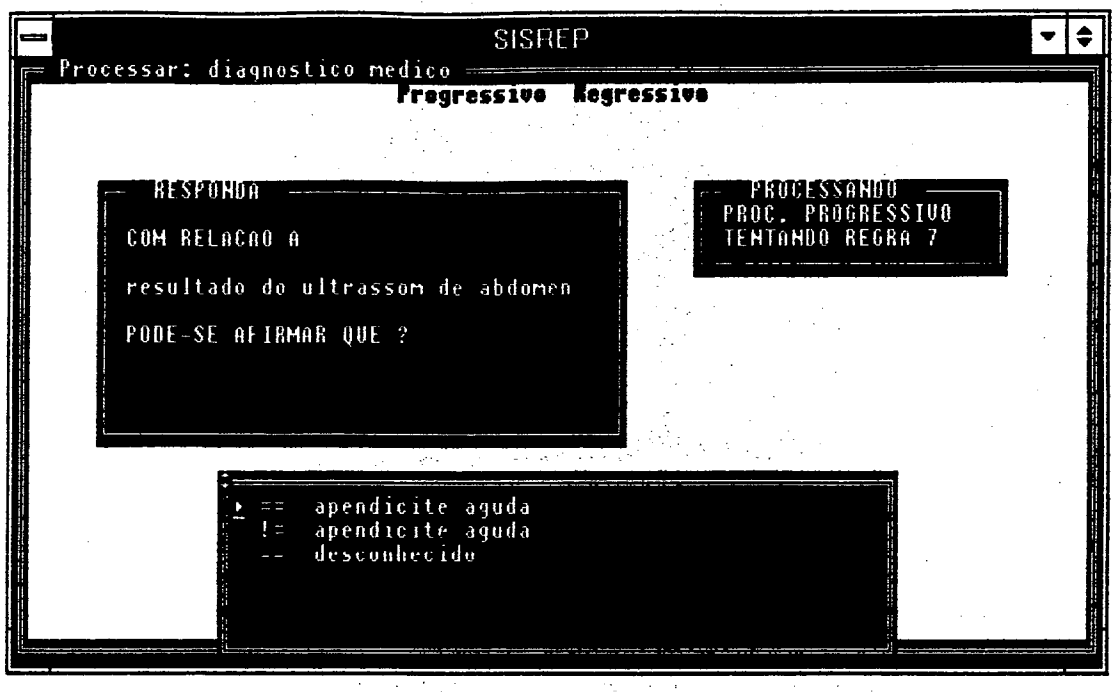

figura 19 - Uma Pergunta do Sistema

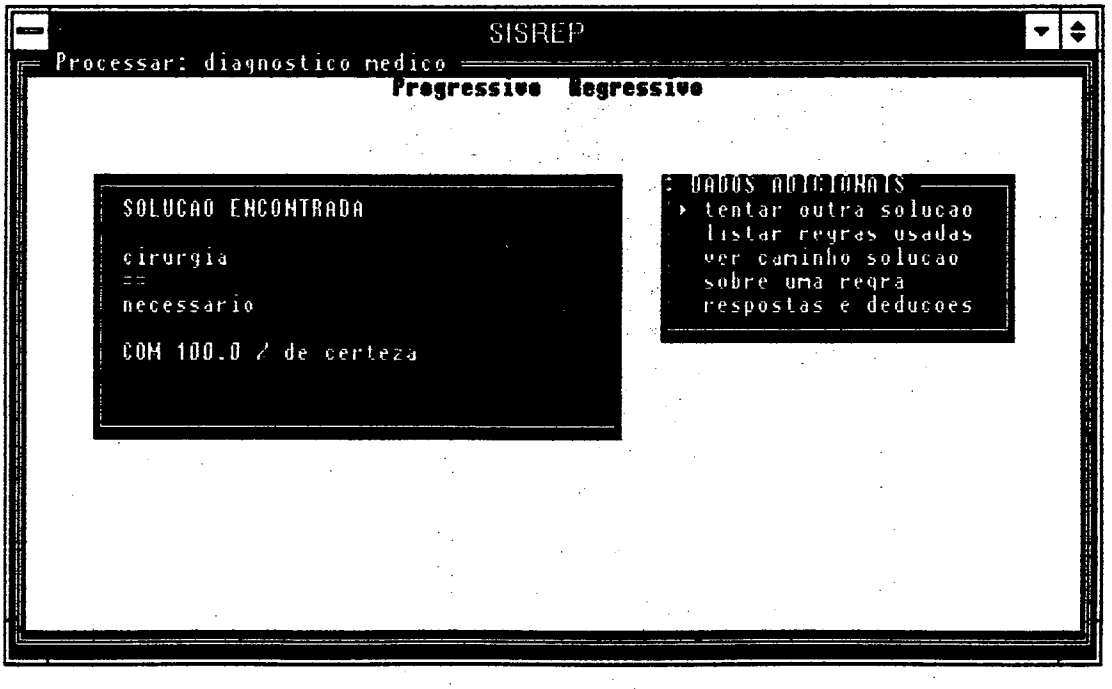

figura 20 - Apresentação de Solução

A figura 20 mostra a solução alcançada pelo SISREP. Observa-se que, paralelamente com a solução apresentada, o sistema coloca, à direita do vídeo, um menu com as opções de justificação desta solução. O SISREP mostra também o grau de certeza associada à solução encontrada.

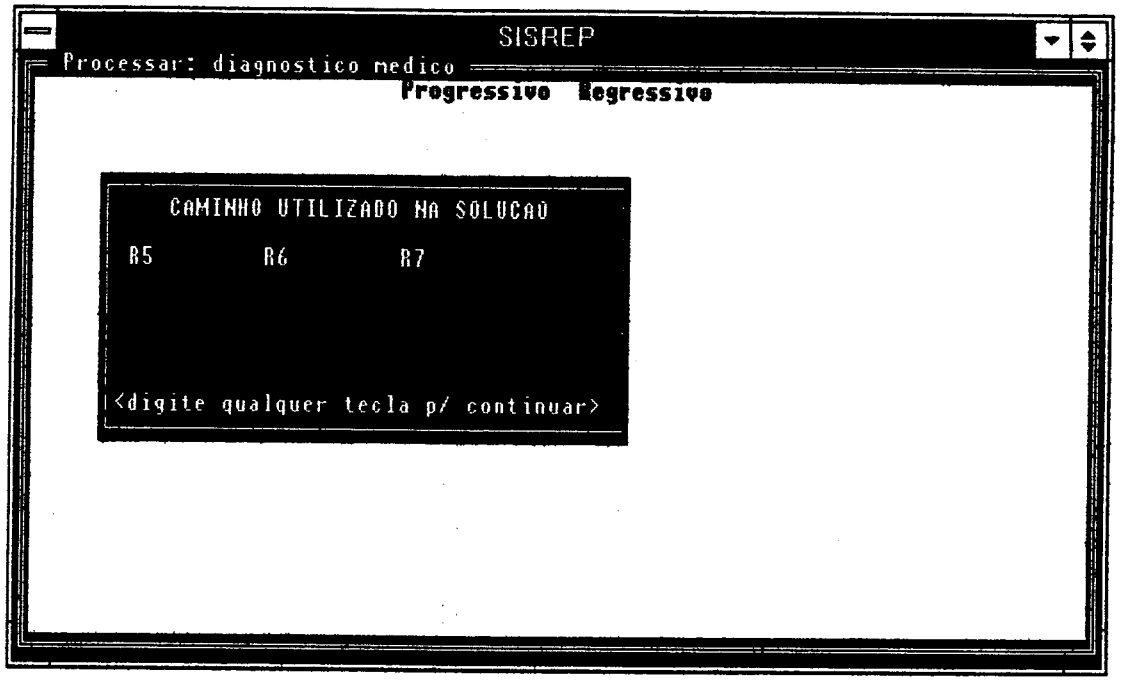

figura 21 - Apresentação do Caminho Solução

A figura 21 mostra a resposta para a opção "ver caminho solução" do menu de justificação. Neste caso são listados os números das regras que foram avaliadas e que estavam no caminho que levou à solução apresentada pelo SISREP.

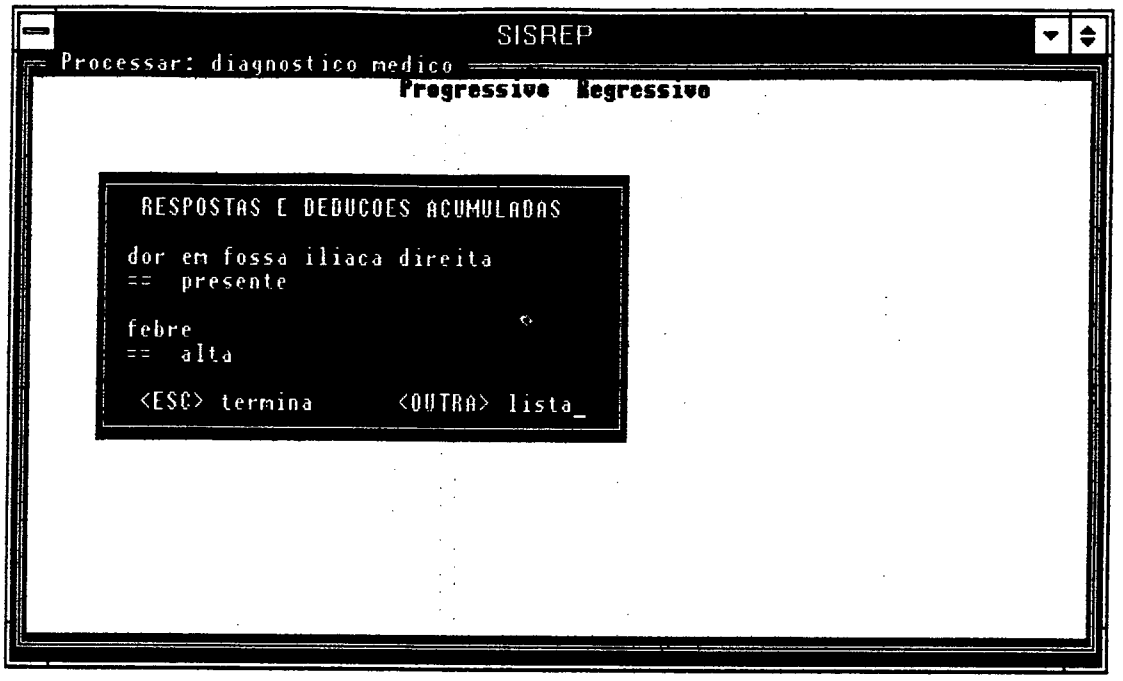

figura 22 - Respostas e Deduções Acumuladas

A figura 22 mostra a resposta para a opção "respostas e deduções" do menu de justificação. O SISREP apresenta, neste caso, uma seqüência de janelas onde são listadas as cláusulas que formaram a lista de verdades durante o processo de busca da solução apresentada na figura 20.

CAPÍTULO 6

6 - CONCLUSÕES E RECOMENDAÇÕES

6.1 - CONCLUSÕES

O presente trabalho permitiu um estudo dos aspectos relacionados à criação e utilização de sistemas especialistas. As formas de representação do conhecimento adotadas pelos sistemas e os mecanismos de inferência sobre este conhecimento, preconizados pela maioria dos autores, foram abordados detalhadamente.

A utilização das regras de produção para a representação do conhecimento, apontado pela literatura especializada como a mais comum e viável, foi testada na implementação do protótipo. Ela mostrou-se adequada pois permitiu representar o conhecimento necessário e foi facilmente assimilada pelo especialista na elaboração da base de conhecimentos da aplicação experimental.

Entre as ferramentas de apoio ao desenvolvimento de sistemas especialistas a linguagem de programação ARITY/PROLOG mostrou-se bastante poderosa. Os algoritmos de busca, que são naturalmente não triviais, foram programados sem maiores dificuldades. Apenas algumas deficiências da linguagem foram observadas

nas operações de entrada e saída. Neste aspecto a linguagem pode ser melhorada com a incorporação de novos predicados internos para facilitar este tipo de operação.

A linguagem de programação adotada mostrou-se eficiente, ainda, pela boa qualidade do código gerado pelo seu compilador. Os programas executáveis são pequenos e sua execução é bastante rápida. O mecanismo de paginação da memória auxiliar, e o seu tratamento como uma extensão da memória real do computador, facilitam o trabalho de programação.

Através da aplicação experimental verificou-se a viabilidade da utilização desta tecnologia para a solução de problemas complexos. O especialista envolvido no trabalho mostrou entusiasmo com o comportamento e os resultados obtidos do sistema.

O atual estágio de desenvolvimento da tecnologia dos sistemas especialistas faz da mesma uma alternativa real para a implementação de sistemas computacionais. A tendência atual de sua popularização justifica-se no desenvolvimento recente dos recursos de hardware, software e técnicas de representação e recuperação do conhecimento.

6.2 - RECOMENDAÇÕES

Com a conclusão do presente trabalho, abrem-se novas possibilidades de pesquisas:

- realizar várias aplicações do sistema protótipo desenvolvido, a fim de se identificar novas facilidades que poderiam ser acrescentadas.

- desenvolver novos softwares a partir de soluções adotadas e implementadas no SISREP.

- implementar no protótipo desenvolvido a possibilidade de introduzir-se fórmulas matemáticas na sua base de conhecimentos.

- acrescentar no protótipo a possibilidade de relacionar um fator de certeza a cada uma das cláusulas de uma regra.

- transformar o SISREP num software-produto para que o mesmo possa ser oferecido comercialmente.

7 - BIBLIOGRAFIA

01 - [ANAIS,93] Anais do X Simpósio Brasileiro de Inteligência Artificial. Universidade Federal do Rio Grande do Sul. Porto Alegre, outubro, 1993.

02 - [ARITY,88] ARITY CORPORATION, The Arity/Prolog Language Reference Manual,1988.

03 - [BARR,82] BARR, AVRON e FEIGENBAUM, EDWARD A. The Handbook of Artificial Intelligence. Addison-Wesley Publishing Company. 1982.

04 - [CLOCKSIN,84] CLOCKSIN W. F. e MELLISH C.S. Programming in Prolog. Springer Verlag, 1984.

05 - [DAVIS,89] DAVIS, Randall, Expert Systems: How Far Can They Go?. AI magazine. v.10, n.2, Summer, 1989.

06 - [FEIGENBAUM,83] FEIGENBAUM, E. A. e McCORDUCK, P. The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World. Addison-Wesley Publishing Company, 1983.

07 - [FISCHLER,87] FISCHLER, Martin A. e FIRSCHEIN, Oscar. Intelligence: The Eye, The Brain, and the Computer. Addison-Wesley Publishing Company. 1987.

08 - [GAUTHIER,90] GAUTHIER, Fernando Ostuni et all. Introdução à Inteligência Artificial. Apostila do Curso. NTS - Núcleo de Tecnologia de Software, 1990.

09 - [GERO,88] GERO, J. S. e STANTON, R. Artificial Intelligence Developments and Applications. North-Holland. 1988.

10 - [KELLER,91] KELLER, Robert. Tecnologia de Sistemas Especialistas. Tradução: Reinaldo Castello. Makron Books do Brasil. São Paulo, 1991.

11 - [LORENZONI,87] LORENZONI, Evandro Jorge Freire. Ferramenta para Desenvolvimento de Sistemas Especialistas. Instituto Militar de Engenharia, 1987. Tese de Mestrado.

12 - [LUGER,89] LUGER, George F. e STUBBLEFIELD, William A. Artificial Intelligence and the Design of Expert Systems. The Benjamin/Cummings Publishing Company. 1989.

13 - [MCCARTHY,92] MCCARTHY, John. Epistemological Problems of Artificial Intelligence. Computer Science Department-Stanford University. 1992.

14 - [MARCUS,86] MARCUS, Claudia. Prolog Programming. Addison-Wesley Publishing Company. 1986.

15 - [MINSKY,74] MINSKY, Marvin. A Framework for Representing Knowledge. Memo n. 306 - MIT, Junho, 1974.

16 - [PROLOG,87] ARITY CORPORATION, Using the Arity/Prolog Compiler and Interpreter, 1987.

17 - [RABUSKE,95] RABUSKE, Renato Antônio. Inteligência Artificial. Ed. da UFSC. Florianópolis. 1995.

18 - [RIBEIRO,87] RIBEIRO, Horácio da Cunha e Sousa. Introdução aos Sistemas Especialistas. Livros Técnicos e Científicos Ed. Rio de Janeiro. 1987.

19 - [RICH,93] RICH, Elaine e KNIGHT, Kevin. Artificial Intelligence. McGraw-Hill, 1993.

20 - [STERLING,87] STERLING, Leon e SHAPIRO, Ehud. The Art of Prolog. The MIT Press, 1987.

21 - [THOMPSON,87] THOMPSON, B. Taxing the Expert System Shells. AI Expert, Junho, 1987.

22 - [TURNER,85] TURNER, Raymond. Logics for Artificial Intelligence, Ellis Horwood, 1985.

23 - [WATERMAN,86] WATERMAN, D. A. A Guide to Expert Systems, Addison Wesley, 1986.

24 - [WINSTON,87] WINSTON, P. H. Artificial Intelligence, Addison Wesley, 1987.

APÊNDICE 1

CADASTROS DA APLICAÇÃO EXPERIMENTAL

CADASTRO: ATRIBUTOS

001	dor epigástrica	intermediário
002	azia	intermediário
003	vômitos	intermediário
004	endoscopia digestiva alta	intermediário
005	hematemese	intermediário
006	pirose	intermediário
007	tosse produtiva	intermediário
008	febre	intermediário
009	dor toráxica ventilatório-dependent	intermediário
010	diagnóstico	conclusivo
011	rx de torax	intermediário
012	consolidação	intermediário
013	derrame pleural	intermediário
014	dor em fossa ilíaca direita	intermediário
015	sinal de blumberg	intermediário
016	hemograma	intermediário
017	resultado de hemograma	intermediário
018	ultrassom de abdomen	intermediário
019	cirurgia	conclusivo
020	dor cólica em hipocôndrio direito	intermediário
021	icterícia	intermediário

022	ultrassom de abdomen superior	intermediário
023	colelitíase	intermediário
024	colecistite aguda	intermediário
025	tratamento clínico	intermediário
026	tratamento cirúrgico	conclusivo
027	odinofagia	intermediário
028	amigdalas com placas purulentas	intermediário
029	cefaleia	intermediário
030	rinorreia purulenta	intermediário
031	rx de seios da face	intermediário
032	resultado de ultrassom de abdomen	intermediário
033	resultado ultrassom abdomen superior	intermediário
034	resultado de rx de seios da face	intermediário

CADASTRO: VALORES

001	desconhecido
002	existente
003	presente
004	freqüentes
005	necessário
006	úlcera duodenal
007	pneumonia pneumocócica
008	positivo
009	infecção
010	apendicite aguda
011	cálculos de vesícula

012 colelitíase

013 colecistite aguda

014 ineficaz

015 visíveis

016 amigdalite aguda

017 opacific seios face

018 sinusite aguda

019 alta

020 ausente

CADASTRO: REGRAS

Regra 1

SE A1 dor epigástrica == V2 existente

E A2 azia == V3 presente

E A3 vômitos == V4 freqüentes

ENTÃO A4 endoscopia digestiva alta == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 2

SE A4 endoscopia digestiva alta == V5 necessário

E A5 hematemese == V3 presente

E A6 pirose == V3 presente

ENTÃO A10 diagnóstico == V6 úlcera duodenal

FATOR DE CERTEZA: 90%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 3

SE A7 tosse produtiva == V3 presente

E A8 febre == V19 alta

E A9 dor toráxica ventilatório dependent == V2 existente

ENTÃO A11 rx de torax == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 4

SE A11 rx de torax == V5 necessário

E A12 consolidação == V3 presente

E A13 derrame pleural != V3 presente

ENTÃO A10 diagnóstico == V7 pneumonia pneumocócica

FATOR DE CERTEZA: 80%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 5

SE A14 dor em fossa ilíaca direita == V3 presente

E A8 febre == V19 alta

E A15 sinal de blumberg == V8 positivo

ENTÃO A16 hemograma == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 6

SE A16 hemograma == V5 necessário

E A17 resultado do hemograma == V9 infecção

E A3 vômitos != V20 ausente

ENTÃO A18 ultrassom de abdômen == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 7

SE A18 ultrassom de abdômen == V5 necessário

E A32 resultado de ultrassom de abdômen == V10 apendicite aguda

ENTÃO A19 cirurgia == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 8

SE A20 dor cólica hipocôndrio direito == V3 presente

E A21 icterícia == V3 presente

E A3 vômitos != V20 ausente

ENTÃO A22 ultrassom de abdômen superior == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 9

SE A22 ultrassom de abdômen superior == V5 necessário

E A33 resultado ultrassom abdômen superior == V11 cálculos de vesícula

ENTÃO A10 diagnóstico == V12 colelitíase

FATOR DE CERTEZA: 85%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 10

SE A23 colelitíase == V3 presente

E A16 hemograma == V9 infecção

E A8 febre == V19 alta

ENTÃO A10 diagnóstico == V13 colecistite aguda

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 11

SE A24 colecistite aguda == V3 presente

E A25 tratamento clínico == V14 ineficaz

ENTÃO A25 tratamento cirúrgico == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 12

SE A27 odinofagia == V3 presente

E A8 febre == V19 alta

E A28 amigdalas com placas purulentas == V15 visíveis

ENTÃO A10 diagnóstico == V16 amigdalite aguda

FATOR DE CERTEZA: 75%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 13

SE A29 cefaléia == V3 presente

E A30 rinorréia purulenta == V2 existente

E A8 febre == V19 alta

ENTÃO A31 rx de seios da face == V5 necessário

FATOR DE CERTEZA: 100%

VALIDADE ATÉ: 31/12/99 as 23:59:59

Regra 14

SE A31 rx de seios da face == V5 necessário

E A34 resultado de rx dos seios da face == V17 opacific seios face

ENTÃO A10 diagnóstico == V18 sinusite aguda

FATOR DE CERTEZA: 90%

VALIDADE ATÉ: 31/12/99 as 23:59:59