UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(BACHARELADO)

ESTUDO DA FERRAMENTA PARA SISTEMAS ESPECIALISTAS CLIPS APLICADO NO DIAGNÓSTICO DE TRANSTORNOS MENTAIS

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA OBTENÇÃO DOS CRÉDITOS DE DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

CLÉVERSON TAMBOSI

BLUMENAU (SC), NOVEMBRO/2001

2001/2-10

ESTUDO DA FERRAMENTA PARA SISTEMAS ESPECIALISTAS CLIPS APLICADO NO DIAGNÓSTICO DE TRANSTORNOS MENTAIS

CLÉVERSON TAMBOSI

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Profª. Marilda Maria de Souza — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Profª. Marilda Maria de Souza

_____

Prof. Roberto Heinzle

_____

Prof. Oscar Dalfovo

DEDICATÓRIA

Dedico este trabalho aos meus pais
Olímpio e Irene Tambosi
por toda a força e apoio recebido
durantes todos esses anos.

AGRADECIMENTOS

À professora e orientadora Marilda M. de Souza pela orientação deste trabalho. Aos professores do Departamento de Sistemas de Informação pelo conhecimento transmitido.

Aos verdadeiros amigos pelos momentos de descontração e pelas boas risadas.

SUMÁRIO

DEDICATÓRIA III

AGRADECIMENTOS IV

LISTA DE FIGURAS X

LISTA DE QUADROS XI

RESUMO XII

ABSTRACT XIII

1 INTRODUÇÃO 1

1.1 OBJETIVOS DO TRABALHO 3

1.2 JUSTIFICATIVA 3

1.3 ESTRUTURA DO TRABALHO 4

2 INTELIGÊNCIA ARTIFICIAL 5

2.1 HISTÓRICO 6

2.2 CAMPOS DE APLICAÇÃO 7

2.3 TÉCNICAS DA INTELIGÊNCIA ARTIFICIAL 8

2.3.1 ALGORITMOS GENÉTICOS 8

2.3.1.1 HISTÓRICO 9

2.3.1.2 CARACTERÍSTICAS 10

2.3.2 REDES NEURAIS 10

2.3.2.1 HISTÓRICO 11

2.3.2.2 CARACTERÍSTICAS 12

2.3.3 RACIOCÍNIO BASEADO EM CASO 12

2.3.3.1 HISTÓRICO 13

2.3.3.2 CARACTERÍSTICAS 13

2.3.4 AGENTES 13
2.3.4.1 HISTÓRICO 14
2.3.4.2 CARACTERÍSTICAS 15
3 SISTEMAS ESPECIALISTAS 17
3.1 HISTÓRICO 18
3.2 CARACTERÍSTICAS 19
3.3 ARQUITETURA DOS SISTEMAS ESPECIALISTAS 20
3.3.1 BASE DE CONHECIMENTO 20
3.3.2 MEMÓRIA DE TRABALHO (QUADRO NEGRO) 21
3.3.3 MOTOR OU MECANISMO DE INFERÊNCIA 21
3.3.4 MECANISMO DE APRENDIZAGEM E AQUISIÇÃO DO CONHECIMENTO 22
3.3.5 SISTEMA DE JUSTIFICAÇÃO 23
3.3.6 SISTEMA DE CONSULTA 23
3.4 FORMAS DE ENCADEAMENTO 23
3.4.1 ENCADEAMENTO PARA FRENTE 24
3.4.2 ENCADEAMENTO PARA TRÁS 24
3.5 REPRESENTAÇÃO DO CONHECIMENTO 24
3.5.1 REDES SEMÂNTICAS 25
3.5.2 QUADROS OU *FRAMES* 25
3.5.3 LÓGICA DAS PROPOSIÇÕES E DOS PREDICADOS 26
3.5.4 REGRAS DE PRODUÇÃO 28
4 FERRAMENTAS E LINGUAGENS 30
4.1 M.4VB 30
4.1.1 HISTÓRICO 31
4.1.2 CARACTERÍSTICAS 31

4.1.3 ARQUITETURA 31
4.1.4 RECURSOS E FACILIDADES 32
4.2 JESS 33
4.2.1 HISTÓRICO 34
4.2.2 CARACTERÍSTICAS 34
4.2.3 ARQUITETURA 34
4.2.3.1 FORMATOS 34
4.2.3.2 FUNÇÕES 35
4.2.3.3 VARIÁVEIS 36
4.2.4 RECURSOS E FACILIDADES 36
4.3 EXPERT SINTA 36
4.3.1 HISTÓRICO 36
4.3.2 CARACTERÍSTICAS 37
4.3.3 ARQUITETURA 38
4.3.4 FORMA DE REPRESENTAÇÃO DO CONHECIMENTO 39
4.3.5 RECURSOS E FACILIDADES 40
4.4 PROLOG 40
4.4.1 HISTÓRICO 40
4.4.2 CARACTERÍSTICAS 40
4.4.3 FORMA DE REPRESENTAÇÃO DO CONHECIMENTO 41
4.4.4 RECURSOS E FACILIDADES 41
5 CLIPS 43
5.1 HISTÓRICO 43
5.2 CARACTERÍSTICAS E ARQUITETURA 45
5.3 FORMA REPRESENTAÇÃO DO CONHECIMENTO 45

5.3.1 CONHECIMENTO HEURÍSTICO (REGRAS) 45
5.3.2 CONHECIMENTO PROCEDURAL 47
5.4 AMBIENTE E INTERFACE 47
5.5 MÁQUINA DE INFERÊNCIA 48
5.6 WXCLIPS 48
5.6.1 AMBIENTE DE DESENVOLVIMENTO WXCLIPS 48
6 ENGENHARIA DE SOFTWARE E FERRAMENTAS 51
6.1 QUALIDADE DE SOFTWARE 51
6.2 ISO/IEC 9126 53
6.2.1 OBJETIVOS 54
7 O PROTÓTIPO 56
7.1 AVALIAÇÃO DO PROBLEMA 56
7.2 ESPECIFICAÇÃO 69
8 ANÁLISE DA FERRAMENTA CLIPS 74
8.1 CARACTERÍSTICAS DA QUALIDADE E MÉTRICA ISO/IEC9126 74
8.2 ASPECTOS RELEVANTES PARA SISTEMAS ESPECIALISTAS 77
8.2.1 INTERFACE COM O USUÁRIO 77
8.2.2 INTERFACE DE DESENVOLVIMENTO 78
8.2.3 INTERFACE COM O SISTEMA OPERACIONAL 79
8.2.4 MOTOR DE INFERÊNCIA 80
8.2.4.1 MÉTODO DE RACIOCÍNIO 80
8.2.4.2 REPRESENTAÇÃO DE INCERTEZA 81
8.2.5 REPRESENTAÇÃO DO CONHECIMENTO 82
8.2.6 TESTES EFETUADOS E RESULTADOS 82
9 CONCLUSÃO 83

9.1 EXTENSÕES 83

REFERÊNCIAS BIBLIOGRÁFICAS 85

LISTA DE FIGURAS

FIGURA 1 – ESTRUTURA DE UM SISTEMA ESPECIALISTA 20
FIGURA 2 – EXEMPLO DE REDE SEMÂNTICA COM CONCEITOS DE GEOMETRIA DESCRITIVA 25
FIGURA 3 – COMPONENTES DE UM *FRAMES* 26
FIGURA 4 - ARQUITETURA DE UM SISTEMA ESPECIALISTA NO EXPERT SINTA 39
FIGURA 5 – EXEMPLO DE REGRA DE PRODUÇÃO NO EXPERT SINTA 40
FIGURA 6 – AMBIENTE CLIPS 47
FIGURA 7 - EXEMPLO DE *INTERFACES* DO WXCLIPS 49
FIGURA 8 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DEVIDO A UMA CONDIÇÃO MÉDICA GERAL 58
FIGURA 9 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DEVIDO A UMA CONDIÇÃO MÉDICA GERAL (CONT.) 59
FIGURA 10 – ÁRVORE DE DECISÃO DOS TRANSTORNOS INDUZIDOS POR SUBSTÂNCIAS 60
FIGURA 11 - TRANSTORNOS INDUZIDOS POR SUBSTÂNCIAS (CONT.) 61
FIGURA 12 – ÁRVORE DE DECISÃO DOS TRANSTORNOS PSICÓTICOS 62
FIGURA 13 – ÁRVORE DE DECISÃO DOS TRANSTORNOS PSICÓTICOS (CONT.) 63
FIGURA 14 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DO HUMOR 64
FIGURA 15 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DO HUMOR (CONT.) 65
FIGURA 16 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DE ANSIEDADE 66
FIGURA 17 – ÁRVORE DE DECISÃO DOS TRANSTORNOS DE ANSIEDADE (CONT.) 67
FIGURA 18 – ÁRVORE DE DECISÃO DOS TRANSTORNOS SOMATOFORMES 68
FIGURA 19 – ÁRVORE DE DECISÃO DOS TRANSTORNOS SOMATOFORMES (CONT.) 69
FIGURA 20 – PROCEDIMENTOS APÓS RODAR O PROGRAMA 73

LISTA DE QUADROS

QUADRO 1 – DESCRIÇÃO DOS COMPONENTES DE UM *FRAME* 26
QUADRO 2 – EXEMPLO DE REGRA DE PRODUÇÃO 29
QUADRO 3 – EXEMPLO DE REGRA M.4 32
QUADRO 4 – EXEMPLO DE PROCEDIMENTO NO M.4 33
QUADRO 5 – EXEMPLO DE REGRAS NO CLIPS 46
QUADRO 6 - CONJUNTO DE CARACTERÍSTICAS E SUBCARACTERÍSTICAS DA ISO/IEC 9126 53
QUADRO 7 – FUNÇÕES PARA AUXILIAR A FAZER PERGUNTAS AO USUÁRIO 69
QUADRO 8 – FUNÇÕES PARA SEPARAR OS GRUPOS GERAIS DE SINTOMAS 70
QUADRO 9 – EXEMPLOS DE REGRAS PARA TRANSTORNOS DEVIDO A UMA CONDIÇÃO MÉDICA GERAL 71
QUADRO 10 – EXEMPLOS DE REGRAS PARA TRANSTORNOS INDUZIDOS POR SUBSTÂNCIA 71
QUADRO 11 – FUNÇÃO PARA IMPRIMIR A DOENÇA 72
QUADRO 12 – CARACTERÍSTICAS DA QUALIDADE SOFTWARE 74
QUADRO 13 – EXEMPLO DE DECLARAÇÃO DE GRAU DE CONFIANÇA 81

RESUMO

O presente trabalho apresenta um estudo das características e particularidades da ferramenta para sistemas especialistas CLIPS aplicado em um problema prático, sendo este, o diagnóstico de transtornos mentais. Para isto, foi implementado um protótipo de um sistema especialista que auxiliará o profissional da área psicológica que possui dificuldades em manusear e/ou interpretar o Manual de Diagnóstico de Transtornos Mentais (DSM).

ABSTRACT

This work presents a study of characteristics and particularities of expert systems CLIPS tool in a practical problem, being this, the diagnosis of mental disorders. For this, it was established a prototype of an expert system that will help the professional of psychological area who has difficulties in handing and/or interpreting the "Diagnostic and statistical manual of mental disorder".

1 INTRODUÇÃO

A Inteligência Artificial é a área da computação voltada para o estudo das faculdades mentais, que busca o desenvolvimento de sistemas inteligentes. Suas características estão voltadas para a inteligência humana, por exemplo: compreensão da linguagem, aprendizado, raciocínio, resolução de problemas e outros. O seu objetivo é desenvolver programas capazes de resolver problemas de maneira inteligente, como quando resolvido por pessoas (Souza, 1998, p. 2).

O Sistema Especialista (SE) aplica técnicas de Inteligência Artificial e conhecimento em problemas específicos de um dado domínio para simular a atuação de peritos humanos. O SE procura solucionar problemas do mesmo modo que um especialista humano, através de programas e computador. A eficácia deste sistema depende diretamente de sua quantidade de conhecimento.

Desenvolvido a partir da necessidade de processar informações, um sistema especialista é capaz de apresentar conclusões sobre um determinado tema, desde que devidamente orientado e alimentado.

Existe uma série de ferramentas próprias para o uso de técnicas de Inteligência Artificial, porém pode-se usar qualquer boa linguagem de programação para se construir sistemas inteligentes. Alternadamente, existem as *shells*, que visam simplificar a construção e gerência da base de conhecimento dos sistemas especialistas.

A razão para a elaboração deste trabalho deu-se ao perceber as dificuldades encontradas pelo desenvolvedor de *software* em escolher a ferramenta adequada para o desenvolvimento de seus sistemas.

O presente trabalho propõe-se, justamente, a auxiliar numa análise do desempenho de uma destas ferramentas, sendo esta, a ferramenta para sistemas especialistas CLIPS. Fundamentado em pesquisas e ensaios, procurar-se-á estudar as facilidades, os recursos e a qualidade desta ferramenta, através da implementação de uma aplicação experimental, auxiliando assim na tomada de decisão para futuras aplicações.

Para a implementação de uma aplicação para testar a ferramenta, optou-se por desenvolver um sistema especialista para diagnóstico de transtornos mentais, que auxiliará o

psicólogo a entender melhor o funcionamento do manual Diagnóstico e Estatístico dos Transtornos Mentais (DSM), utilizado para diagnosticar seus pacientes (Jorge, 1995).

O diagnóstico é baseado no conjunto de sintomas apresentados pelo paciente. Muitas vezes, um mesmo sintoma poderá fazer parte de mais de uma patologia. Em outros casos, poderá haver cruzamento de sintomas, o que dificulta o diagnóstico da doença. Todas as doenças mentais catalogadas encontram-se descritas no DSM. Essas doenças estão dividias em seis grupos gerais de transtornos: transtornos mentais devido a uma condição médica geral, transtornos mentais induzidos por substâncias, transtornos psicóticos, transtornos do humor, transtornos de ansiedade e transtornos somatoformes. Dentro de cada grupo existem os vários sintomas que caracterizam cada uma das doenças catalogadas.

O livro Manual Diagnóstico e Estatístico de Transtornos Mentais é fundamental para o tratamento dos pacientes. A partir dele, juntamente com o histórico de vida do paciente, serão determinados quais os métodos que serão utilizados no tratamento, a medicação que será ministrada e o prognóstico. Mas, às vezes, é necessário um trabalho multidisciplinar e, conseqüentemente, uma linguagem técnica em comum entre os especialistas. Para isso, foi desenvolvido o manual (Jorge, 1995, p. 45).

O DSM-I foi publicado em 1952. “O DSM-I continha um glossário de descrições de categorias diagnósticas e foi o primeiro manual oficial de transtornos mentais a focalizar a utilidade clínica” (Jorge, 1995, p XVII). “A finalidade do DSM-I é oferecer descrições claras de categorias diagnósticas, a fim de permitir que clínicos e investigadores diagnostiquem, comuniquem, estudem e tratem pessoas com vários transtornos mentais.” (Jorge, 1995, p XXV). Desta forma, torna-se possível os profissionais terem certeza de que estarão trabalhando com uma mesma doença.

Este trabalho propõe-se a descrever e implementar o protótipo deste sistema especialista, bem como a análise do desempenho e potencialidade da ferramenta CLIPS através do uso da norma ISO/IEC 9126, que lista o conjunto de características que devem ser verificadas em um *software*, como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade, as quais são divididas em um conjunto de subcaracterísticas. Serão analisados também os aspectos relevantes da ferramenta, tais como, interface com o usuário, interface de desenvolvimento, interface com o sistema operacional, motor de inferência e representação do conhecimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é fazer um estudo das características e particularidades da ferramenta para sistemas especialistas CLIPS, desenvolvendo um protótipo para diagnóstico de pacientes com transtornos mentais, o qual auxiliará na análise da ferramenta.

Os objetivos específicos do trabalho são:

a) analisar o desempenho da ferramenta CLIPS;

b) avaliar a qualidade através de aspectos da engenharia de *software* propostos pela ISO/IEC 9126;

c) identificar os aspectos relevantes para esta ferramenta;

d) ampliar o estudo da ferramenta em uma situação prática, sendo esta, o diagnóstico de transtornos mentais.

1.2 JUSTIFICATIVA

Hoje em dia, já se sabe que os benefícios que a informática traz podem ser úteis à maioria das outras áreas profissionais. Apesar disso, há incontáveis situações, em diversas áreas, que ainda não fazem uso da ajuda que a informática pode proporcionar.

Uma das razões para a elaboração deste trabalho deu-se ao perceber as dificuldades encontradas pelos profissionais de informática em escolher a ferramenta adequada para o desenvolvimento de seus sistemas.

Outro motivo para o desenvolvimento do presente trabalho foi, também, dificuldades que os profissionais da área psicológica têm em interpretar o livro Manual Diagnóstico e Estatístico de Transtornos Mentais. Essa dificuldade deve-se pelo fato de o manual ser muito extenso, e, muitas vezes, o profissional não sabe como manuseá-lo. É dentro deste contexto, que é proposto o desenvolvimento de um protótipo de *software* para o auxílio a profissionais que têm dificuldades para lidar com o manual DSM-IV.

Este trabalho apresenta um estudo da ferramenta CLIPS aplicado em uma situação real, sendo esta, o Diagnóstico de Transtornos Mentais.

1.3 ESTRUTURA DO TRABALHO

O trabalho encontra-se estruturado da seguinte maneira:

O capítulo 1, inicia com uma breve introdução sobre os assuntos que vão ser vistos ao decorrer deste trabalho, seguido de seus objetivos e justificava deste.

No capítulo 2, faz-se um levantamento sobre os conceitos da Inteligência Artificial.

No capítulo 3, estuda-se sobre a sub-área da Inteligência Artificial, Sistemas Especialistas.

O capítulo 4 faz uma abordagem sobre algumas linguagens e ferramentas para sistemas especialistas.

O capítulo 5 trata especificamente sobre a ferramenta para sistemas especialistas CLIPS.

No capítulo 6 são abordados os critérios da Engenharia de *Software* para avaliar a qualidade da ferramenta.

O capítulo 7, principal foco do trabalho, trata sobre o protótipo construído.

O capítulo 8 aborda as considerações para a realização da avaliação da ferramenta.

No capítulo 9 trata-se sobre as conclusões deste trabalho e sugestões para projetos futuros.

2 INTELIGÊNCIA ARTIFICIAL

Até onde sabemos, o termo Inteligência Artificial (IA) foi utilizado pela primeira vez, por John McCarthy, em agosto de 1956, no convite para o encontro que ficou conhecido por Conferência de Dartmouth. Esse encontro tinha por objetivo "estudar as bases da conjectura de que cada aspecto do aprendizado e da inteligência pode, em princípio, ser tão precisamente descrito que o torne passível de ser simulado por uma máquina", e reuniu 15 pioneiros do que viria a constituir a área de Inteligência Artificial (IA, 2000).

Uma definição para Inteligência Artificial diz que IA é o estudo de como fazer os computadores realizarem tarefas que, no momento, são melhor executadas pelas pessoas. Essa definição não inclui aqueles problemas nos quais os computadores já são melhores que os humanos, nem as áreas nas quais os problemas não são bem solucionadas nem pelos humanos nem pelos computadores; no entanto, é uma definição útil, pois limita, de forma razoavelmente precisa, o escopo da Inteligência Artificial.

Já Fernandes (1996) diz que inteligência artificial é um ramo da ciência da computação construída a partir de idéias filosóficas, científicas e tecnológicas herdadas de outras ciências, algumas tão antigas quanto a lógica, a física e a química, e seu objetivo é o estudo do mundo físico, de maneira que as teorias e modelos desenvolvidos nestas disciplinas se aplicam a um domínio onde a existência ou não de pessoas ou inteligência não é relevante para a validade científica das teorias.

Outra definição amplamente aceita é a de Minsky (1968): IA é a ciência de fazer máquinas fazerem coisas que requereriam inteligência, caso fossem feitas por homens.

Para simular o raciocínio humano e implementar aspectos da inteligência, a IA precisa responder a perguntas como estas:

- Como ocorre o pensar?
- Como o homem extrai conhecimentos do mundo?
- Como a memória, os sentidos e a linguagem ajudam no desenvolvimento da inteligência?
- Como surgem idéias?

- Como a mente processa informações e tira conclusões decidindo por uma coisa ao invés de outra?

A IA fornece métodos e técnicas para o desenvolvimento de programas que simulam nas máquinas, comportamentos inteligentes, isto é, tornam os computadores capazes de "pensar e tomar decisões". Por isso, as técnicas de IA necessitam de uma grande quantidade de conhecimentos e de mecanismos de manipulação de símbolos (IA, 2000).

2.1 HISTÓRICO

Após a II Guerra Mundial o computador não ficou restrito ao âmbito militar e científico, começou a ser gradualmente utilizado em empresas, indústrias, universidades. etc. A diversidade de aplicações estimulou pesquisas de *software*, *hardware* e linguagens de programação (IA, 2000).

O desenvolvimento do computador, primeiramente impulsionado pela aplicabilidade militar e posteriormente comercial, mostrou-se viável. Seu rápido progresso, desde o surgimento dos primeiros computadores eletrônicos (1943 - Collossus, na Inglaterra e 1946 - ENIAC, nos Estados Unidos) até o surgimento dos microcomputadores (na década de 70) demonstra que essa área recebeu grandes investimentos.

O segundo grande passo foi dado nos Estados Unidos, em 1956, quando John McCarthy reuniu em uma conferência proferida ao Darmouth College, na Universidade de New Hampshire, vários pesquisadores de renome para estudar o que foi denominado por McCarthy, Minsky, Newell e Simon, de Inteligência Artificial (IA), expressão utilizada para designar um tipo de inteligência construída pelo homem para dotar a máquina de comportamentos inteligentes.

A partir da estruturação desse novo campo do conhecimento o fenômeno da inteligência começou a ser pesquisado de forma intensa. Vários esforços foram e têm sido feitos no sentido se simular os tipos de raciocínios utilizados pelo ser humano e implementá-los no computador por meio da IA.

A inteligência artificial é amplamente utilizada como um auxiliar que expande a capacidade de inteligência do homem e até mesmo o substitui em diversas funções. Isso se tornou possível em grande parte graças ao desenvolvimento dos sistemas especialistas, da lógica *fuzzy* e das redes neurais.

Atualmente, criar máquinas inteligentes não pode ser considerado uma ficção. A IA transformou essa ficção em um campo de estudo movido por uma meta que consome bilhões de dólares em projetos, os quais envolvem pesquisadores de instituições governamentais, militares, industriais e universitárias de todo o mundo.

2.2 CAMPOS DE APLICAÇÃO

Existem vários campos de estudo dentro da IA com o propósito de dotar a máquina de capacidade de raciocínio, aprendizado e autoaperfeiçoamento (IA, 2000).

O Processamento de Linguagem Natural é o estudo voltado para a construção de programas capazes de compreender a linguagem natural (interpretação) e gerar textos. A Geração de linguagem Natural é a produção de textos por um programa a partir de um conteúdo semântico representado internamente no próprio programa. Objetiva aperfeiçoar a comunicação entre as pessoas e os computadores.

O Reconhecimento de Padrões consiste em uma das áreas de pesquisa bem avançadas da IA. A capacidade de reconhecimento de padrões permite ao programa reconhecer a fala em linguagem natural, os caracteres digitados e a escrita (ex.: assinatura). Os scanners, por exemplo, utilizam programas de reconhecimento óptico desenvolvidos pelas pesquisas em IA.

O desenvolvimento da Visão de Computador busca encontrar maneiras de o micro trabalhar com a visão bidimensional e tridimensional.

A parte de Programação de Jogos estuda a construção de programas de jogos envolvendo raciocínio. Os jogos computadorizados são um grande sucesso, ainda mais quando exibem um tipo de inteligência capaz de desafiar as habilidades do jogador. O jogo de xadrez, por exemplo, foi utilizado para as primeiras experiências em programação do raciocínio artificial, onde o computador se tornou capaz de analisar milhões de jogadas por segundo para tentar derrotar o adversário. Além de analisar as jogadas, os programas utilizam um método heurístico que consiste na utilização de uma árvore de busca, onde a mesma possui ramificações a partir de certos nós, que representam pontos de decisão no caminho a tomar, com um certo número de etapas, para chegar a um objetivo. Deste modo, ele pode analisar vários nós, de acordo com a situação atual do jogo, e escolher o melhor caminho (o mais curto ou menos arriscado).

Na Robótica, procura-se encontrar meios de construir máquinas que possam interagir com o meio (ver, ouvir e reagir aos estímulos sensoriais).

Existem também programas de IA que conseguem aprender certos fatos por meio da experiência, desde que esse conhecimento possa ser representado de acordo com o formalismo adotado pelo programa.

2.3 TÉCNICAS DA INTELIGÊNCIA ARTIFICIAL

Segundo Rich (1993), uma técnica de IA é um método que explora o conhecimento, e deve ser representado de tal forma que o conhecimento: (i) capture generalizações, as situações que compartilham propriedades importantes são agrupadas, caso contrário haverá grande necessidade de memória e atualização; (ii) seja compreendido pelas pessoas que o fornecem, pois a grande parte de conhecimento que um programa possui precisa basicamente ser fornecido pelas pessoas em termos que elas compreendam; (iii) seja facilmente modificado para corrigir erros e refletir mudanças do mundo e da visão do mundo que o usuário possui, (iv) seja usado em inúmeras situações, mesmo que não seja totalmente preciso nem esteja completo; (v) sirva de ajuda para superar seu próprio volume, auxiliando a limitar as várias possibilidades que, em geral, têm de ser consideradas.

2.3.1 ALGORITMOS GENÉTICOS

Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais (Carvalho, 1997).

Além de ser uma estratégia de gerar-e-testar muito elegante, por serem baseados na evolução biológica, são capazes de identificar e explorar fatores ambientais e convergir para soluções ótimas, ou aproximadamente ótimas em níveis globais.

Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes. Este é o conceito básico da evolução genética biológica. A área biológica mais proximamente ligada aos Algoritmos Genéticos (AGs) é a Genética Populacional.

Os pesquisadores referem-se a "algoritmos genéticos" ou a "um algoritmo genético" e não "ao algoritmo genético", pois AGs são uma classe de procedimentos com muitos passos separados, e cada um destes passos possui muitas variações possíveis.

Funcionamento básico dos AGs: Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é avaliada: para cada indivíduo é dada uma nota, ou índice, refletindo sua habilidade de adaptação a determinado ambiente. Uma porcentagem dos mais adaptados são mantidos, enquanto os outros são descartados (*darwinismo*). Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais através de mutações e cruzamento (*crossover*) ou recombinação genética gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada.

2.3.1.1 HISTÓRICO

Até meados do século 19, os naturalistas acreditavam que cada espécie havia sido criada separadamente por um ser supremo ou através de geração espontânea. O trabalho do naturalista Carolus Linnaeus sobre a classificação biológica de organismos despertou o interesse pela similaridade entre certas espécies, levando a acreditar na existência de uma certa relação entre elas.

Depois de mais de 20 anos de observações e experimentos, Charles Darwin apresentou em 1858 sua teoria de evolução através de seleção natural, simultaneamente com outro naturalista inglês Alfred Russel Wallace. No ano seguinte, Darwin publica o seu On the Origin of Species by Means of Natural Selection com a sua teoria completa, sustentada por muitas evidências colhidas durante suas viagens a bordo do Beagle.

Este trabalho influenciou muito o futuro não apenas da Biologia, Botânica e Zoologia, mas também teve grande influência sobre o pensamento religioso, filosófico, político e econômico da época. A teoria da evolução e a computação nasceram praticamente na mesma época: Charles Babbage, um dos fundadores da computação moderna, e amigo pessoal de Darwin desenvolveu sua máquina analítica em 1833.

Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética, foi redescoberto pelos cientistas e teve grande

influência sobre os futuros trabalhos relacionados à evolução. A moderna teoria da evolução combina a genética e as idéias de Darwin e Wallace sobre a seleção natural, criando o princípio básico de Genética Populacional: a variabilidade entre indivíduos em uma população de organismos que se reproduzem sexualmente é produzida pela mutação e pela recombinação genética.

Este princípio foi desenvolvido durante os anos 30 e 40, por biólogos e matemáticos de importantes centros de pesquisa. Nos anos 50 e 60, muitos biólogos começaram a desenvolver simulações computacionais de sistemas genéticos. Entretanto, foi John Holland quem começou, seriamente, a desenvolver as primeiras pesquisas no tema. Holland foi gradualmente refinando suas idéias e em 1975 publicou o seu livro *Adaptation in Natural and Artificial Systems*, hoje considerado a Bíblia de Algoritmos Genéticos. Desde então, estes algoritmos vêm sendo aplicados com sucesso nos mais diversos problemas de otimização e aprendizado de máquina.

2.3.1.2 CARACTERÍSTICAS

Os AGs empregam uma estratégia de busca paralela e estruturada, mas aleatória, que é voltada em direção ao reforço da busca de pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Apesar de aleatórios, não são caminhadas aleatórias não direcionadas, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos.

Durante cada iteração, os princípios de seleção e reprodução são aplicados a uma população de candidatos que pode variar, dependendo da complexidade do problema e dos recursos computacionais disponíveis.

A maioria das representações são genotípicas, utilizam vetores de tamanho finito em um alfabeto finito.

2.3.2 REDES NEURAIS

O cérebro humano é considerado o mais fascinante processador baseado em carbono existente, sendo composto por aproximadamente 10 bilhões neurônios. Todas as funções e movimentos do organismo estão relacionados ao funcionamento destas pequenas células. Os

neurônios estão conectados uns aos outros através de sinapses, e juntos formam uma grande rede, chamada REDE NEURAL. Esta grande rede proporciona uma fabulosa capacidade de processamento e armazenamento de informação (Tatibana, 2000).

Os principais componentes dos neurônios são:

a) Os dendritos, que têm por função, receber os estímulos transmitidos pelos outros neurônios;

b) O corpo de neurônio, também chamado de somma, que é responsável por coletar e combinar informações vindas de outros neurônios;

c) E finalmente o axônio, que é constituído de uma fibra tubular que pode alcançar até alguns metros, e é responsável por transmitir os estímulos para outras células.

2.3.2.1 HISTÓRICO

As primeiras informações mencionadas sobre a neuro computação datam de 1943, em artigos de McCulloch e Pitts, em que sugeriam a construção de uma máquina baseada ou inspirada no cérebro humano. Em 1949, Donald Hebb escreveu um livro entitulado "*The Organization of Behavior*" (A Organização do Comportamento) que perseguia a idéia de que o condicionamento psicológico clássico está presente em qualquer parte dos animais pelo fato de que esta é uma propriedade de neurônios individuais. Suas idéias não eram completamente novas, mas Hebb foi o primeiro a propor uma lei de aprendizagem específica para as sinapses dos neurônios. Este primeiro e corajoso passo serviu de inspiração para que muitos outros pesquisadores perseguissem a mesma idéia. Embora muito tenha sido estudado e publicado nos anos que seguiram (1940-1950), estes serviram mais como base para desenvolvimento posterior que para o próprio desenvolvimento (Tatibana, 2000).

Em 1951, foi construído por Marvin Minsky o primeiro neurocomputador, denominado Snark. Este operava com sucesso a partir de um ponto de partida técnico, ajustando seus pesos automaticamente. Entretanto, ele nunca executou qualquer função de processamento de informação interessante, mas serviu de inspiração para as idéias de estruturas que o sucederam.

O primeiro neuro computador a obter sucesso foi o Mark I Perceptron, surgido por 1957 e 1958, criado por Frank Rosenblatt, Charles Wightman e outros.

Infelizmente, o desenvolvimento em torno das Redes Neurais passou por um período ocioso. Após esse período, Ira Skurnick, um administrador de programas da DARPA (Defense Advanced Research Projects Agency) decidiu ouvir os argumentos da neuro computação e seus projetistas, e divergindo dos caminhos tradicionais dos conhecimentos convencionais, fundou em 1983 pesquisas em neuro computação. Este ato não só abriu as portas para a neuro computação, como também deu à DARPA o status de uma das líderes mundiais em se tratando de "moda" tecnológica.

Em 1987, ocorreu em São Francisco a primeira conferência de redes neurais em tempos modernos, a IEEE International Conference on Neural Networks, e também foi formada a International Neural Networks Society (INNS). A partir destes acontecimentos, decorreram: a fundação do INNS journal em 1989, seguido do Neural Computation e do IEEE Transactions on Neural Networks em 1990.

Desde 1987, muitas universidades anunciaram a formação de institutos de pesquisa e programas de educação em neuro computação

2.3.2.2 CARACTERÍSTICAS

Resumidamente, quando um processo é criado visando a utilizar aspectos de redes neurais começam com o desenvolvimento de um neurônio artificial ou computacional baseado no entendimento de estruturas biológicas neurais, seguidas do aprendizado de mecanismos voltados para um determinado conjunto de aplicações. Ou em outras palavras, seguindo as três etapas:

- O desenvolvimento de modelos neurais motivado por neurônios biológicos;
- Modelos de estruturas e conexões sinápticas;
- O aprendizado das regras (um método de ajuste de pesos ou forças de conexões internodais).

2.3.3 RACIOCÍNIO BASEADO EM CASO

Raciocínio Baseado em Caso (RBC) pode ser definido como um modelo de raciocínio que envolve o entendimento de situações, a solução de problemas e o aprendizado, integrando esses aspectos com processos de memória (Reginaldo, 2000).

Pode-se justificar a construção de sistemas de RBC com base na referência de que psicólogos verificam que as pessoas sentem-se confortáveis usando casos passados para tomar decisões, entretanto elas têm dificuldades para lembrar-se dos melhores casos.

2.3.3.1 HISTÓRICO

A origem do RBC em Inteligência Artificial é encontrada nos trabalhos de Roger Schank, sendo que o primeiro sistema que pode ser considerado de RBC foi o CYRUS, desenvolvido por Janet Kolodner. Este sistema foi baseado no modelo de memória dinâmica de Roger Schank e na teoria de MOPs (*memory organization packets* – pacotes de organização de memória) para aprendizagem e solução de problemas (Reginaldo, 2000).

Outras bases para RBC e outros conjuntos de modelos foram desenvolvidos por Bruce Porter e seu grupo.

Atualmente, trabalhos envolvendo RBC tanto nos Estados Unidos quanto na Europa têm se expandido e o número de publicações vem crescendo consideravelmente em atividades relacionadas a IA.

2.3.3.2 CARACTERÍSTICAS

A utilização de Raciocínio Baseado em Casos envolve quatro passos que devem ser observados:

- Construir modelos que representam de forma satisfatória a abstração dos problemas ou objetos envolvidos;
- Selecionar um problema ou objeto conhecido a fim de compará-lo a um desconhecido;
- Mapear os atributos do problema ou objeto conhecido, com o desconhecido;
- Estender o mapeamento com o objetivo de gerar uma solução que seja considerada válida para ser aplicada ao desconhecido.

2.3.4 AGENTES

O termo agente é utilizado na literatura computacional para determinar diversos tipos de programas. Estes programas não precisam, necessariamente, apresentar um comportamento "inteligente" (Costa, 1999).

Um agente pode ser qualquer sistema autônomo que percebe e age para alcançar um estreito conjunto de metas dentro de um específico ambiente virtual ou real. Os sistemas tradicionais de IA têm sido projetados para operar sob o controle do usuário e sobre um olho cuidadoso, enquanto agentes são lançados no mundo para agir autonomamente, geralmente sob suas próprias crenças.

2.3.4.1 HISTÓRICO

A idéia de agentes foi originada em meados da década de 50 com John McCarthy e Oliver Selfridge. Eles tinham em mente um sistema onde, dada uma meta, uma seqüência de ações seria executada na tentativa de satisfazer aquela meta.

Os primeiros agentes trabalhavam na resolução de problemas usando heurísticas ou métodos baseados em conhecimento. Alguns destes agentes poderiam ainda planejar, aprender e modificar suas percepções sobre o ambiente no qual estavam inseridos.

Na década de 70, Hewitt propôs um agente com o conceito de um objeto auto-suficiente, interativo e com execução concorrente, o qual ele denominava ator. Este objeto teria alguns estados internos encapsulados e responderia a mensagens de objetos similares.

O desenvolvimento de poderosos computadores, a proliferação de redes de computadores e a consideração sobre o uso de grupos de indivíduos para resolver problemas, fizeram com que a Inteligência Artificial Distribuída (IAD) se tornasse uma área emergente.

Esse crescimento da IAD pode ser separado em duas linhas distintas: a primeira linha, do período de 1977 até 1990, trabalha com agentes ativos, estando mais relacionados a tipo de agente deliberativo com modelos simbólicos internos. Um agente deliberativo é aquele que possui uma representação explícita, um modelo simbólico do mundo, e cujas decisões (ações a executar) são realizadas através de um raciocínio simbólico; a segunda linha de pesquisa, partindo de 1990, em rápido desenvolvimento, enfatiza a diversificação das tipologias ou classes de agentes, onde os agentes apresentam uma evolução em sua inteligência e autonomia. Esta linha demonstra a evolução da deliberação para a execução, do raciocínio para a ação remota.

2.3.4.2 CARACTERÍSTICAS

Um agente não precisa possuir todas as características listadas a seguir, embora suas capacidades estejam diretamente associadas a presença delas.

- Autonomia - refere-se ao princípio de que os agentes podem agir baseados em seus próprios princípios, sem a necessidade de serem guiados por humanos. No contexto da Internet, autonomia é a habilidade de operar, mesmo quando o usuário final está desconectado da rede. Neste caso, esta propriedade normalmente está associada à outra: mobilidade.
- Mobilidade - a capacidade de poder se mover através de uma rede de computadores parece ser interessante para agentes que auxiliam seus usuários na busca de informações, principalmente dentro da Internet.
- Cooperação - pode ser entendida como a capacidade que os agentes têm de trabalharem em conjunto de forma a concluírem tarefas de interesse comum.
- Comunicabilidade - quando existe mais de um agente envolvido, há uma necessidade óbvia por um modelo de comunicação. Entretanto, o conceito de comunicabilidade não estabelece apenas a troca de informações entre agentes.
- Aprendizagem - um dos atributos que mais caracterizam agentes inteligentes é a capacidade de aprender. Uma real autonomia só pode estar presente quando um agente possui a habilidade de avaliar as variações de seu ambiente externo e escolher qual a ação mais correta. Entretanto, mesmo quando um agente não reconhece nenhuma ação a ser executada, é esperado que ele procure encontrar uma saída. A questão não é acertar sempre, mas aprender continuamente por experiência, seja através de sucessos ou de fracassos.
- Reatividade - é a habilidade que um agente tem de reagir a mudanças no seu ambiente. Para tal, o agente deve ser capaz de perceber seu ambiente e atuar sobre ele.
- Habilidade Social - este atributo está diretamente associado com a característica de comunicabilidade, uma vez que representa a habilidade de interagir com outros agentes.

- Pró-atividade – este atributo pode ser também denominado iniciativa, uma vez que representa um comportamento independente. As ações são dirigidas pelo objetivo e não simplesmente por mudanças no seu ambiente. O agente que implementa este atributo possui maior flexibilidade, pois é capaz de resolver problemas causados por situações inesperadas.

3 SISTEMAS ESPECIALISTAS

Sistemas Especialistas podem ser definidos como sistemas de inteligência artificial, criados para resolver problemas em determinado domínio cujo conhecimento utilizado é fornecido por pessoas que são especialistas naquele domínio (IA, 2000).

Sistema - "Conjunto de elementos, materiais ou idéias, entre os quais se possa encontrar ou definir alguma relação" (Ferreira, 1988).

Especialista - "Pessoa que se consagra com particular interesse e cuidado a certo estudo. Conhecedor, perito".

Definir um sistema especialista pode não ser tarefa fácil. O que você definiria como um Sistema especialista, depois de ler as definições independentes dessas duas palavras? Alguns estudiosos deram suas opiniões:

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 (Heinzle, 1995).

Rich (1993) diz: “Os sistemas especialistas solucionam problemas que normalmente são solucionados por especialistas humanos. Para solucionar tais problemas, os sistemas especialistas precisam acessar uma substancial base de conhecimentos do domínio da aplicação, que precisa ser criada do modo mais eficiente possível”.

Para Crippa (1999), “Um sistema especialista é um programa inteligente de computador que se utiliza de métodos inferenciais para a resolução de problemas técnicos e altamente especializados. Por utilizar-se da Inteligência Artificial, um ramo da computação que estuda a capacidade de uma máquina raciocinar e aprender como um ser humano, os sistemas especialistas interagem com seu usuário numa linguagem natural de perguntas e respostas, sugerindo e auxiliando na solução de problemas complexos”.

Segundo Friedman-Hill (2001), um sistema especialista é um conjunto de regras que podem ser repetidamente aplicadas a uma coleção de fatos sobre o mundo.

Sendo assim, um Sistema Especialista é aquele projetado e desenvolvido para atender a uma aplicação determinada e limitada do conhecimento humano. É capaz de emitir uma decisão, apoiado em conhecimento justificado, a partir de uma base de informações, da mesma forma que um especialista de determinada área do conhecimento humano (IA, 2000).

Para tomar uma decisão sobre um determinado assunto, baseado nos fatos que encontra, um sistema especialista formula suas hipóteses. Durante o processo de raciocínio, ele verifica qual a importância dos fatos que encontra, comparando-os com as informações já adquiridas sobre esses fatos e hipóteses. Neste processo, vai formulando novas hipóteses e verificando novos fatos; e esses novos fatos vão influenciar no processo de raciocínio. Este raciocínio é sempre baseado no conhecimento prévio acumulado.

Um sistema especialista com esse processo de raciocínio pode não chegar a uma decisão se os fatos de que dispõe para aplicar o seu conhecimento prévio não forem suficientes. Pode, inclusive, chegar a uma conclusão errada. Porém, este erro é justificado em função dos fatos que encontrou e do seu conhecimento acumulado previamente.

Um Sistema Especialista deve, além de inferir conclusões, ter capacidade de aprender e, desse modo, melhorar o seu desempenho de raciocínio e a qualidade de suas decisões.

De um modo geral, sempre que um problema não pode ser algoritmizado, ou sua solução conduz a um processamento muito demorado, os Sistemas Especialistas podem ser uma saída, pois possuem o seu mecanismo apoiado em processos heurísticos. Além disso, pode ser uma boa maneira de preservar e transmitir o conhecimento de um especialista humano em uma determinada área.

Vale ainda lembrar que um Sistema Especialista não é influenciado por elementos externos a ele, como ocorre com o especialista humano. Dessa forma, para as mesmas condições, deverá fornecer sempre o mesmo conjunto de decisões.

3.1 HISTÓRICO

Segundo Heinzle (1995), os primeiros trabalhos em sistemas que hoje são chamados de especialistas surgiram na década de 60. A finalidade era 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 criar-se-iam sistemas de capacidade superiores à capacidade humana. No entanto, não tardou para que os pesquisadores observassem o engano e verificassem as reais dimensões do trabalho.

Em 1964, segundo o autor, surgiu o programa algorítmico DENDRAL. Construído por Joshua Lederberg da Universidade de Stanford, o DENRAL, a partir de um determinado

conjunto de dados como massa espectrográfica e ressonância magnética, deduz a possível estrutura de um determinado composto químico.

Já em 1965, Joshua, juntamente com Edward Feigenbaum e Bruce Buchnan, tentaram construir um programa que usasse regras heurísticas para resolver os mesmos problemas do DENDRAL. Surgiu então um novo DENDRAL que mostrou a viabilidade dos sistemas especialistas e levou pesquisadores de outras universidades a trabalharem no assunto.

Por 1968, surge o MACSYMA, destinado a auxiliar matemáticos na resolução de problemas complexos. Este, é um sistema ainda hoje amplamente utilizado em universidades e laboratórios de pesquisa.

Já na década de 70, concentraram esforços em técnicas de representação, isto é, modo de formular o problema de maneira a tornar sua solução mais fácil. Surgiram então, importantes e complexos sistemas especialistas, dentre eles, o MYCIN e o PROSPECTOR. O MYCIN é um sistema na área médica para detectar e diagnosticar doenças infecciosas, enquanto que o PROSPECTOR é um sistema para dar suporte a geólogos na exploração mineral.

Na década de 80 houve o grande crescimento de aplicações. Este acelerado processo de desenvolvimento de aplicações, deve-se em parte ao avanço dos recursos de equipamentos, ocorrido paralelamente neste período.

Hoje em dia, acontece um crescimento de Sistemas Especialistas em todas as áreas, mas nota-se forte desenvolvimento para a área médica em geral.

3.2 CARACTERÍSTICAS

Algumas características dos Sistemas Especialistas, segundo Fávero (2000):

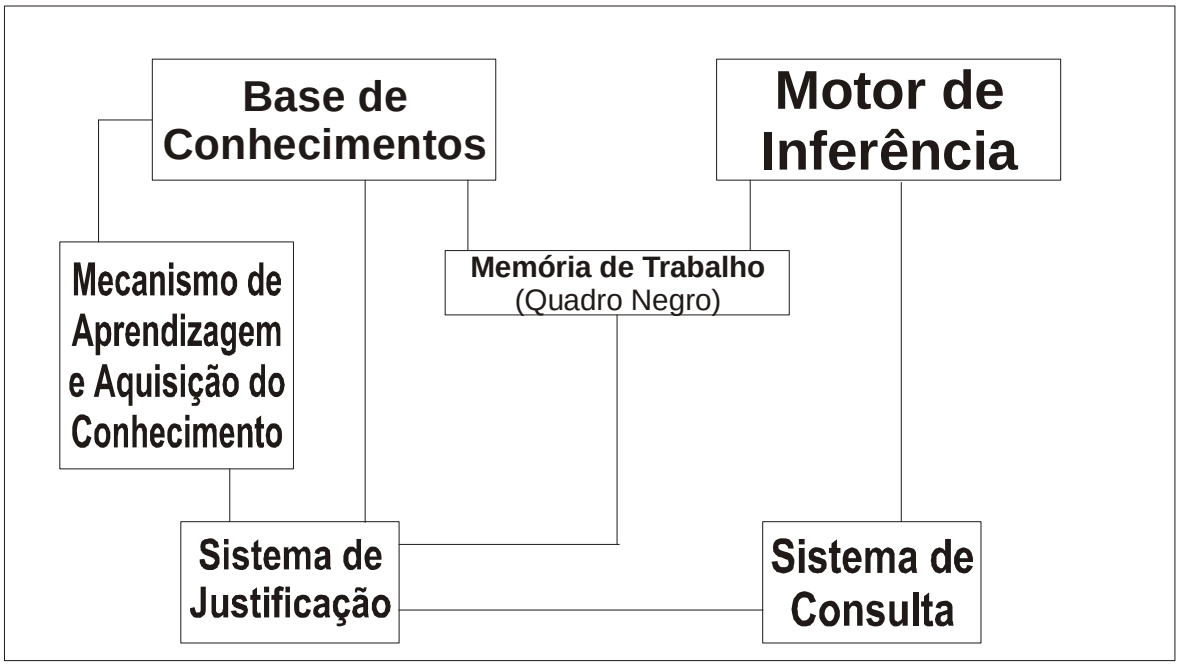
- Resolvem problemas muito complexos tão bem quanto ou melhor que especialistas humanos;
- Raciocinam heuristicamente, usando o que os peritos consideram regras práticas;
- Interagem com usuários humanos utilizando, inclusive, linguagem natural;
- Manipulam e raciocinam sobre descrições simbólicas;
- Funcionam com dados errados e regras incertas de julgamento;

- Contemplam hipóteses múltiplas simultaneamente;
- Explicam porque estão fazendo determinada pergunta;
- Justificam suas conclusões.

3.3 ARQUITETURA DOS SISTEMAS ESPECIALISTAS

Conforme Heinzle (1995), 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, a forma mais simples de compreender e a mais difundida, compõe-se de alguns elementos básicos: base de conhecimentos, máquina ou motor de inferência, quadro negro, sistema de justificação, mecanismo de aprendizagem e aquisição de conhecimento e sistema de consulta, representada na **Figura 1.**

Figura 1 – Estrutura de um Sistema Especialista



Fonte: (Heinzle, 1995)

3.3.1 BASE DE CONHECIMENTO

Segundo Fávero (2000), é o local onde ficam armazenadas as informações de um Sistema Especialista. 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 conhecimento 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 a memória de trabalho.

Para Manchini (2000), a Base de Conhecimento “é um elemento permanente, mas específico de um sistema especialista. É onde estão armazenadas as informações, ou seja, os fatos e as regras. As informações armazenadas de um determinado domínio fazem do sistema um especialista neste domínio”.

3.3.2 MEMÓRIA DE TRABALHO (QUADRO NEGRO)

Segundo Manchini (2000), é um local dentro da memória do computador no qual as informações armazenadas em um sistema especialista são "afixadas" para que qualquer outro sistema especialista possa acessá-las, caso precise delas para alcançar seus objetivos.

Já Heinzle (1995) diz: “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”.

3.3.3 MOTOR OU MECANISMO 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 motor de inferência (Heinzle, 1995).

Para Manchini (2000), mecanismo de inferência ou motor de inferência é um elemento permanente, que pode ser reutilizado por vários sistemas especialistas. É a parte responsável pela busca das regras da base de conhecimento para serem avaliadas, direcionando o processo de inferência. O conhecimento deve estar preparado para uma boa interpretação e os objetos devem estar em uma determinada ordem representados por uma árvore de contexto.

3.3.4 MECANISMO DE APRENDIZAGEM E AQUISIÇÃO DO CONHECIMENTO

Conforme Manchini (2000), a aquisição do conhecimento é a etapa do desenvolvimento que busca obter o conhecimento necessário à construção do sistema, e não acontece de uma só vez, mas distribuída por todo o processo.

A aquisição do conhecimento deve ser realizada de acordo com as seguintes tarefas:

a) Coleta inicial de conhecimento;

b) Redução do conhecimento incorreto;

c) Aumento do conhecimento adquirido.

As características do especialista terão grande impacto na aquisição do conhecimento. A experiência deste influencia a organização, o armazenamento e a recuperação do conhecimento. Quanto maior for a organização e mais refinado o processo de raciocínio, mais difícil decodificar o conhecimento e expressá-lo de forma básica. O conhecimento do especialista não se encontra organizado de forma que possa ser extraído diretamente para a base de conhecimento. Muitos pesquisadores e engenheiros do conhecimento têm tentado trabalhar na construção de mecanismos que facilitem este processo.

Outros fatores contribuem para que a aquisição do conhecimento seja a parte mais crítica no processo de desenvolvimento de um sistema baseado em conhecimento. São eles:

- Falta de gerência e organização;
- Treinamento incompleto de engenheiros do conhecimento;
- Dificuldade da tradução do conhecimento para uma linguagem compreensiva pelo computador;
- Necessidade da flexibilidade da base de conhecimento;
- Tratamento de conflitos e informações incertas.

Vários pontos devem ser considerados no processo de aquisição do conhecimento, independente do método de desenvolvimento utilizado. A primeira delas, e talvez a mais complexa, é a identificação dos especialistas do domínio da organização que irão participar do desenvolvimento. O segundo ponto é a motivação do especialista do domínio, que deve ser reforçada pelo engenheiro do conhecimento. Também devemos considerar a disponibilidade do especialista, que na maioria das vezes é um profissional muito ocupado. Por último, deve

preocupar com o que se refere à seleção e ao seqüenciamento das técnicas de elicitação do conhecimento. O engenheiro do conhecimento desempenha um papel fundamental no processo de aquisição do conhecimento, fornecendo meios ao especialista para que ele consiga transmitir a sua maneira de solucionar os problemas.

3.3.5 SISTEMA DE JUSTIFICAÇÃO

Segundo Heinzle (1995), o módulo de Justificação tem a função de esclarecer o usuário a respeito de uma conclusão apresentada pelo sistema ou ainda explicar porque uma pergunta está sendo feita. Ele é, na verdade, um recurso de questionamentos fornecido ao usuário.

De uma forma geral, os sistemas são implementados para responder às seguintes perguntas:

- Como chegou a esta conclusão?
- Por que chegou a esta conclusão?
- Por que não chegou a tal conclusão?

3.3.6 SISTEMA DE CONSULTA

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

Aspectos internos dos sistemas, terminologia computacional, entre outros, devem ser evitados e detalhes técnicos relativos à 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 (Heinzle, 1995).

3.4 FORMAS DE ENCADEAMENTO

O encadeamento de regras é a forma mais comum de utilizar as regras para chegar a uma conclusão. Existem dois tipos mais comuns de protocolos de encadeamento de regras o encadeamento para frente e para trás, os quais estão descritos a seguir (Keller, 1991).

3.4.1 ENCADEAMENTO PARA FRENTE

Ao usar a técnica de encadeamento para frente, ou *data driven*, para chegar a uma conclusão, é como se não conhecesse nada sobre a base de dados e precisasse descobrir alguma coisa. São fornecidos alguns fatos sobre uma dada situação, embora possamos ter de solicitá-los explicitamente.

O método elementar do encadeamento para frente começa arbitrariamente com a primeira regra da base de conhecimento e tenta usá-la. Pode-se usar qualquer regra em que todas as suas premissas são conhecidas como verdadeiras.

Após usar de fato uma regra, adiciona-se todas as suas conclusões e procura-se pela próxima regra na base de conhecimento que utiliza a conclusão da primeira como uma de suas premissas. Tenta-se então, utilizar esta nova regra. Este movimento que vai da conclusão de uma regra à premissa de outra é o chamado encadeamento para frente.

3.4.2 ENCADEAMENTO PARA TRÁS

O encadeamento para trás é, algumas vezes, chamado de encadeamento *goal-driven*. Ele difere do encadeamento para frente principalmente pelo fato de iniciar assumindo que uma conclusão é verdadeira e, então, usar as regras para provar essa conclusão.

O método do encadeamento para trás começa com uma conclusão e prova-a como verdadeira pela demonstração de suas premissas como verdadeiras numa ordem da esquerda para a direita ou de cima para baixo. Para provarmos a verdade de uma premissa, procuramos por uma regra que tenha essa premissa como uma de suas conclusões. Se tal regra é encontrada, encadeamos para trás até ela e tentamos prová-la demonstrando a verdade de cada uma de suas premissas da mesma maneira. O processo termina quando não houver mais regras a serem provadas.

3.5 REPRESENTAÇÃO DO CONHECIMENTO

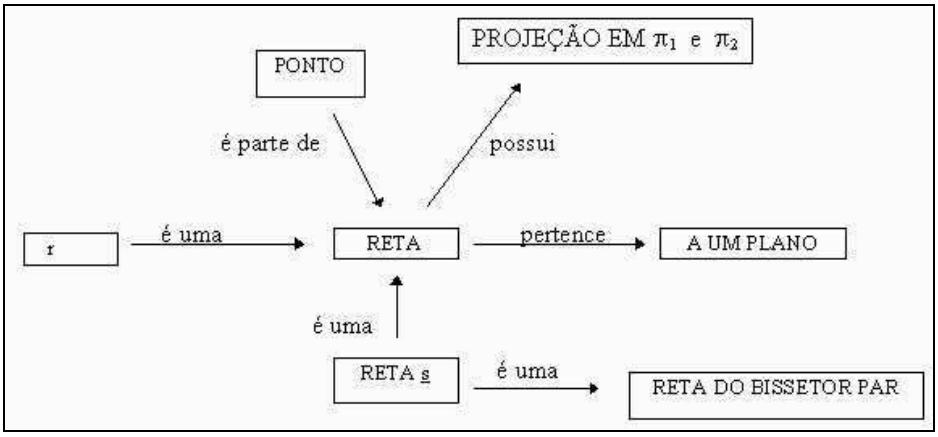
De acordo com Heinzle (1995), o conhecimento deve ser transformado em organizadas estruturas de dados para que possa ser posteriormente recuperado, para raciocinar com ele e até mesmo para adquirir mais conhecimento. A representação do conhecimento é a formalização do conhecimento do sistema. Para isto, existem técnicas que facilitam o acesso.

Ainda segundo o autor, existem várias formas para representar o conhecimento. Dentre elas, citam-se:

3.5.1 REDES SEMÂNTICAS

As redes semânticas foram originariamente desenvolvidas para modelagem psicológica da memória humana, constituindo-se agora num método de representação padrão. Podem ser ilustradas por diagramas que contêm nós e arcos. Os nós representam objetos, ações ou eventos. Os arcos que os ligam representam relações sobre os nós. Uma ligação pode significar que um objeto de um lado é um atributo de um objeto, do outro, pode significar que um implica outro, ou ainda alguma coisa definida pelo usuário (Souza 1998).

Figura 2 – Exemplo de Rede Semântica com conceitos de Geometria Descritiva

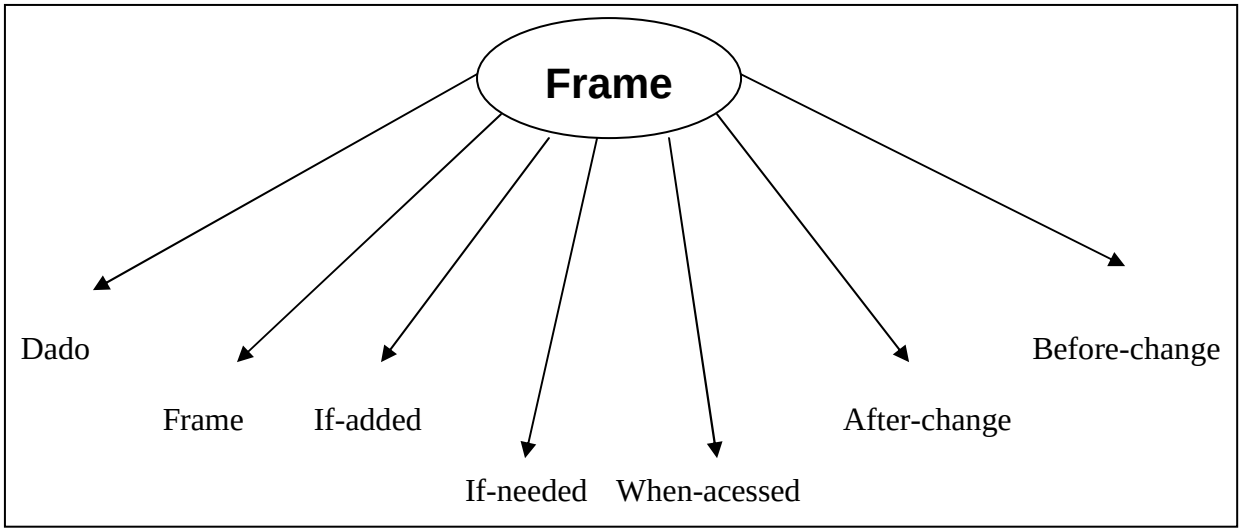


Fonte: (Ulbricht, 2000)

3.5.2 QUADROS OU *FRAMES*

Segundo Souza (1998), este tipo de representação, organiza o conhecimento de maneira a tornar evidente a compreensão de como a inferência pode ser feita. Ajuda o sistema a interpretar o significado de acordo com o contexto e provê o armazenamento de detalhes associados com ele para inferir objetos. Um quadro consiste de uma coleção de indicadores (*slots*) que descrevem aspectos da situação, ação, objeto ou evento. Tais indicadores são preenchidos com valores, procedimentos ou ponteiros para outros quadros.

Figura 3 – Componentes de um *Frame*



Fonte: (Ramos, 1995)

Quadro 1 – Descrição dos componentes de um *Frame*

if-added	São invocados cada vez que um valor é adicionado a um *slot*.
if-needed	São invocados cada vez que se necessita de uma determinada informação associada a um *slot*, a qual não está explicitada ainda, gerando dúvida.
when-acessed	São invocados cada vez que se realiza um acesso a uma determinada informação associada a um *slot*.
after-change	São invocados após alguma mudança de determinada informação associada a um *slot*.
Before change	São invocados antes de uma mudança de determinada informação associada a um *slot*.

3.5.3 LÓGICA DAS PROPOSIÇÕES E DOS PREDICADOS

Essas duas técnicas utilizam símbolos para representar conhecimentos e operadores aplicados aos símbolos para produzir raciocínio lógico. Também representam métodos formais bem fundamentados para representação do conhecimento e raciocínio.

A lógica proposicional representa e raciocina com proposições (declarações que são verdadeiras ou falsas). Trabalha com operadores lógicos como *AND, OR, NOT, IMPLIES, EQUIVALENCE,* que nos permite raciocinar com várias estruturas de regras.

Segundo Heinzle (1995), 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 de predicados como também é conhecida, é uma extensão da lógica proposicional e fornece uma representação mais apurada do conhecimento, facilitando o processamento ao permitir a utilização de variáveis e funções (UFSC, 2000).

- Constantes:

  Exemplo: a constante carro pode representar o objeto Carro.

- Predicados:

  Exemplo: come (coelho, cenoura).

  Predicado: come (relação entre argumentos coelho e cenoura).

- Variáveis:

  Exemplo: come (X, Y).

  Onde a variável X representa o argumento coelho; a variável Y o argumento cenoura.

- Funções:

  Exemplo: pai (Leo) = Pedro

  mae (Maria)= Lia

  amigos (Pedro, Lia)=amigos (pai (Leo), mae (Maria)).

  Operações em Cálculo de Predicados

O cálculo de predicados utiliza as mesmas operações da lógica proposicional.

Exemplo: Joao ama Maria = ama (Joao, Maria)

Leo ama Maria = ama (Leo, Maria)

ama(X,Y) AND ama(Z,Y)

IMPLIES NOT ama(X,Z)

ou

ama(X,Y) Ù ama(Z,Y) ® Ø ama(X,Z)

O cálculo de predicados trabalha ainda com dois outros tipos de variáveis:

Variáveis quantificadoras:

quantificador universal ("): a expressão é verdadeira para todos os valores das variáveis:

Ex: " X ama (X, Maria)

quantificador existencial ($): existe ao menos um valor para o qual a expressão é verdadeira.

Ex: $ X ama (X, Maria)

3.5.4 REGRAS DE PRODUÇÃO

Conforme Heinzle (1995), o termo “regras de produção” é usado para descrever uma família de sistemas, que têm em comum o fato de serem constituídos de um conjunto de regras, que reúnem condições e ações. As regras são armazenadas como uma coleção de declarações do tipo SE <premissa> 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.

A naturalidade que as regras de produção representam para o ser humano faz com que sejam a forma mais utilizada em sistemas especialistas. Estima-se que cerca de oitenta por cento dos sistemas existentes utilizam esta forma de representação do conhecimento.

As regras de produção possuem, além da interpretação humana, vantagens como modularidade, onde as regras podem ser manipuladas como peças independentes e novas regras podem ser incluídas a qualquer tempo, e uniformidade, onde existe um padrão único utilizado para todas as regras do sistema.

Um exemplo de regra de produção:

Quadro 2 – Exemplo de regra de produção

SE	galerias nos ramos ou inflorescências
OU	inflorescências = murchas ou secas
E	brotações novas murchas
E	orifícios laterais nos ramos ou inflorescências
ENTÃO	praga = broca das pontas CNF 90%

4 FERRAMENTAS E LINGUAGENS

Para implementar sistemas especialistas com o uso de técnicas de inteligência artificial existem vários *softwares*, que se dividem entre linguagens de programação e as chamadas *Shell*s.

Depois de vários sistemas terem sido desenvolvidos, ficou claro que eles tinham muito em comum. Em particular, devido ao fato de os sistemas serem construídos como um conjunto de representações declarativas (em sua maioria, regras) combinadas com um interpretador dessas representações, era possível separar o interpretador do conhecimento específico do domínio da aplicação, e criar um sistema que podia ser usado para elaborar novos sistemas especialistas através da adição de novos conhecimentos, correspondentes ao novo domínio do problema. Os interpretadores resultantes são chamados de *shell*s (Manchini, 2000).

Comparando-se com uma linguagem de programação, uma vantagem das *shells* é a simplicidade para se construir um sistema especialista. O desenvolvedor não precisa de um conhecimento muito elevado.

Algumas linguagens foram desenvolvidas especificamente para facilitar o desenvolvimento de Sistemas Especialistas, mas para construir estes tipos de sistemas, pode-se utilizar outras linguagens de programação como: BASIC, FORTRAM, ALGOL, PASCAL, FORTH, Delphi, Visual BASIC, etc.

4.1 M.4VB

M.4VB é uma *shell* para construção de sistemas baseados em conhecimento e para criação de módulos de processamento inteligente em outros sistemas. Esta *shell* tem sido usado para o desenvolvimento de aplicações de seleção, análise, escalonamento, diagnóstico de configurações, planejamento e projeto, nos campos da manufatura, finanças, administração, processamento de dados, sistemas de informações gerenciais, vendas, marketing, treinamento e educação (Winters, 1997).

A ferramenta M.4 inclui encadeamento para frente e para trás, unificação de padrões, fatos e regras, fatores de certeza e listas, para auxiliar a representação do conhecimento.

4.1.1 HISTÓRICO

A linha de produtos M.4 iniciou com o M.1 em 1984. Desde então, ele tem sido usado para desenvolver uma série de sistemas especialistas para corporações tais como: Ford, Kodak, General Motors, Motorola; e instituições de ensino norte americanas como Stanford University, Texas Tech University, University of Britsh Columbia, University of Florida, University of Utah.

M.4 foi escrito em C com o objetivo de ser pequeno e rápido, e para facilitar a integração com *software* convencionais.

4.1.2 CARACTERÍSTICAS

- *Visual Basic Custom Control*: um controle personalizado suporta uma comunicação fácil entre as aplicações baseadas no Visual Basic e a máquina de processamento do conhecimento do M.4VB.
- Características avançadas de linguagem: tanto o controle procedural e a programação orientada a objeto são integradas com uma linguagem e um ambiente de desenvolvimento baseado em conhecimento.

4.1.3 ARQUITETURA

O motor de inferência do M.4 é uma biblioteca que pode ser *linkada* (ligada) às aplicações. Esta biblioteca tem cerca de 110 rotinas que podem ser chamadas para realizar funções que vão desde a inicialização de uma sessão com M.4 até modificações dinâmicas da base de conhecimento. A biblioteca de inferência do M.4 pode ser usada das seguintes maneiras: DLL – (*Dynamic Link Library*) pode ser criada e *linkada* com um executável; VBX – (*Visual Basic Custom Control*) pode ser criado e *linkado* com um executável; EXE - A própria biblioteca pode ser *linkada* com um executável.

Várias aplicações que embutiram o motor de inferência M.4 serviram como exemplo dentro do pacote M.4. Um servidor DDE está incluído para suportar a comunicação com aplicações que suportam o protocolo DDE no ambiente Windows. O desenvolvimento e distribuição de interfaces criadas utilizando *Asymetrise Toolbook* e *Visual Basic* estão incluídas para facilitar o desenvolvimento e depuração dos sistemas de conhecimento. A

interface *Toolbook* é suportada usando o servidor DDE e a interface do Visual Basic é suportada pelo servidor DDE ou pelo VBX do M.4.

4.1.4 RECURSOS E FACILIDADES

A seguir são descritas algumas características da capacidade de representação do conhecimento:

- Expressões Simbólicas: manipulam valores, podem ser testadas em regras, testadas ou manipuladas em procedimentos ou fornecerem valores estáticos através de fatos. Quando um valor de uma expressão simbólica é necessário, a máquina de inferência do M.4, guiada pela base de conhecimento, pode obter o valor a partir de um fato, pelo uso de regras, perguntando ao usuário, e/ou chamando uma rotina externa.
- Fatos: são usados na base de conhecimento para representar informações que são estaticamente conhecidas sobre o domínio. Fatos podem também ser armazenados em arquivos que podem ser carregados de acordo com a necessidade. Estes arquivos podem ser facilmente mantidos ou criados por um editor qualquer e depois passado para o M.4
- Regras: são usados para derivar novos fatos baseados em condições específicas. A premissa de uma regra especifica as condições sob as quais um novo fato pode ser concluído. Comparações simbólicas ou numéricas ou testes podem ser combinados com os operadores *booleanos* tradicionais para formar a premissa da regra. As funções aritméticas também estão disponíveis para a premissa de uma regra. Quando uma regra é usada para encontrar o valor de uma expressão, o encadeamento para trás é disparado, fazendo com o que o M.4 procure valores para as expressões testadas na premissa da regra. Por exemplo, se o M.4 está tentando encontrar o valor da variável vinho recomendando através da regra conforme Quadro 3:

Quadro 3 – Exemplo de Regra M.4

```
If melhor_cor_para_vinho = vermelho and Melhor_cor_para_vinho = denso

Then  vinho_recomendado = zinfandel
```

- Unificação de Padrões: M.4 pode utilizar variáveis para unificação de padrões em regras e fatos para torná-los mais gerais e poderosos, combinando-os em um fato simples que poderia, caso contrário, necessitar de várias regras ou fatos.
- Procedimentos usados no M.4: dão ao engenheiro do conhecimento uma forma compacta e eficiente de controle complexo da especificação. Utilizando uma sintaxe como A do C, pode-se direcionar o fluxo do controle, avaliar expressões e iniciar o encadeamento, interagir, exibir texto, chamar outros procedimentos, e executar os comandos do M.4. O Quadro 4 mostra um exemplo de um procedimento:

Quadro 4 – Exemplo de Procedimento no M.4

```
Procedure (determine-and_display_recs(FAULT)) = {
   Find recommendations (FAULT);
   List := listof (recommendations (FAULT));
   COUNT := 1;
While (List = = [ITEM/REST]) {
         Display ([count, ".", item, NL]);
         COUNT := COUNT +1;
         List := REST; }; }.
```

- Interação, busca e recursividade: são suportadas no M.4 em regras e, no caso de interação somente, em procedimentos. M.4 primeiramente a estrutura de regra com encadeamento para trás.
- Encadeamento para frente: é também suportado no M.4, causando a tomada de ações após valores para uma expressão são alcançados. A ação tomada pode ser condicional e pode envolver a chamada de um procedimento ou o início de uma busca.

4.2 JESS

Segundo Friedman-Hill (2000), Jess é uma ferramenta para construção de sistemas especialistas que usa um algoritmo especial chamado *Rete* para validar as regras através dos

fatos. Este algoritmo faz a ferramenta ser muito mais rápida do que um simples conjunto de regras *if... then* organizadas em cascata.

A ferramenta Jess foi originalmente concebida como um clone em Java do CLIPS, mas hoje em dia possui muitas características que a diferencia de sua linguagem mãe.

4.2.1 HISTÓRICO

A ferramenta Jess foi escrita por Ernest Friedman-Hill do Laboratório Sandia National, como parte de um projeto de pesquisa interno. A primeira versão do Jess foi escrita em 1995, quando Java era uma linguagem bastante nova. Jess se tornou considerável desde então.

Um grande número de usuários desta ferramenta tem contribuído com códigos, sugestões e atualizações. O nome de muitos deles está no Manual de Jess.

4.2.2 CARACTERÍSTICAS

- Originalmente inspirado na ferramenta CLIPS;
- Permite a construção de *applets* Java;
- Permite a criação de objetos Java e chamadas a métodos Java sem a necessidade de qualquer código compilado;
- Representa o conhecimento em forma de regras declarativas;
- Conforme a situação, chega a ser mais rápido que o CLIPS;
- Mesmo a versão mais recente ainda é compatível com CLIPS;
- Possui um mecanismo de resolução de conflitos muito eficiente.

4.2.3 ARQUITETURA

4.2.3.1 FORMATOS

- Átomo (ou símbolo): é o mais avançado conceito da linguagem Jess. Os átomos podem ser comparados aos identificadores, em outras linguagens. Um átomo, em Jess, pode conter letras, números e estes podem ser seguidos pelos sinais: (**`$*=+/<>_?#.`**). Um átomo não pode iniciar com um número; pode começar com

algumas marcas de pontuação (alguns têm significados especiais como operadores quando eles aparecem ao começo de um átomo). Os melhores átomos consistem em letras, números, *underscores* (_) e hífens.

Exemplo: foo first-value contestant#1 _abc

Existem três átomos que o Jess interpreta como especiais: *nil*, o qual é parecido com o *null* do Java; e *true* e *false*, valores *booleanos*.

- Números: o Jess analisa gramaticalmente números inteiros ou reais e não aceita notação científica. Os seguintes números são todos os números válidos: `3 4. 5.643`
- *Strings*: caracteres no Jess são denotados usando aspas (“).

  Exemplo: `"foo" "Hello, World".`
- Listas: é uma unidade de sintaxe fundamental no Jess. Uma lista sempre consiste de um conjunto de parênteses ( ) e zero ou mais átomos, números, *strings* ou outras listas. A primeira palavra da lista pode ser chamada de cabeça da lista. As seguintes listas são válidas:

```
(+ 3 2) (a b c) ("Hello, World") () (deftemplate foo (slot bar))
```

- Comentários: os comentários do programador em Jess iniciam com um ponto-e-vírgula (;) e se estende até o fim da linha.

  Exemplos:

```
; This is a list
(a b c)
```

4.2.3.2 FUNÇÕES

Todo código em Jess (estruturas de controle, chamadas de *procedures*) possui chamadas de função. Chamadas de função em Jess são, simplesmente, listas. Estas chamadas utilizam uma notação especial. Uma lista que possui em sua cabeça um átomo o qual é o nome de uma função existente, pode ser uma chamada de função. Por exemplo, uma expressão que usa a função + para somar dois números poderia ser escrita assim: `(+ 2 3)`. Quando avaliado, o valor desta expressão é o número 5 (não uma lista contendo o número 5!).

Pode-se escrever expressões no *prompt* do Jess:

```
Jess> (+ 2 3)
5
```

```
Jess> (+ (+ 2 3) (* 3 3))
14
```

4.2.3.3 VARIÁVEIS

Variáveis em Jess são átomos que iniciam com o sinal de interrogação (?). Este sinal faz parte do nome da variável. Variáveis deste tipo pode referir-se a simples átomos, números ou *strings*. Há um tipo especial de variável chamada multivariável. Esta possui o cifrão ($) antes do sinal de interrogação (por exemplo, `$?var`). Uma multivariável refere-se a um tipo especial de lista chamado multicampo. O exemplo abaixo utiliza o comando *bind*, que associa um valor a uma variável:

```
Jess> (bind $?grocery-list (eggs bread milk))
(eggs bread milk)
```

4.2.4 RECURSOS E FACILIDADES

Dentre os vários recursos do Jess, destaca-se a interface *Jess.Userfunction*. Com esta, pode-se adicionar novas funções à linguagem Jess simplesmente escrevendo a classe que implementa a interface *Jess.Userfunction*, criando uma instância simples desta classe e instalando-a no objeto *Jess.Rete* usando a função *Rete.addUserfunction().* Essa *Userfunction* criada pode manter um estado, ou seja, pode armazenar resultados após sua chamada. Esses resultados podem ser uma estrutura de dados complexa ou manter referências a objetos Java. Uma *Userfunction* simples pode ser uma porta em um subsistema Java complexo.

4.3 EXPERT SINTA

O *Expert* SINTA é uma *shell* que utiliza técnicas de Inteligência Artificial para geração automática de sistemas especialistas. Esta ferramenta utiliza um modelo de representação do conhecimento baseado em regras de produção e probabilidades, tendo como objetivo principal simplificar o trabalho de implementação de sistemas especialistas através do uso de uma máquina de inferência compartilhada, da construção automática de telas e menus, do tratamento probabilístico das regras de produção e da utilização de explicações sensíveis ao contexto da base de conhecimento modelada (Nogueira, 1995).

4.3.1 HISTÓRICO

Segundo Souza (1998), o projeto *Expert* SINTA teve início em 1995 no Laboratório de Inteligência Artificial (LIA), da Universidade Federal do Ceará. O *software* foi

desenvolvido por bolsistas ligados ao LIA. Os responsáveis pelo programa formam o grupo Sistemas INTeligentes Aplicados (SINTA), que desde então vêm desenvolvendo atividades relacionadas a aplicações práticas de técnicas de Inteligência Artificial.

O *Expert* SINTA foi implementado na linguagem de programação *Delphi*. Através de uma interface de manipulação e de utilitários criados para depuração, o SINTA permite o desenvolvimento modular de bases de conhecimento. Isto facilita para os desenvolvedores da base conhecimento, pois proporciona várias facilidades como economia de tempo.

4.3.2 CARACTERÍSTICAS

- Encadeamendo para trás (*backward chaining)*: O encadeamento para trás é o modo mais comum de utilização de um Sistema Especialista. O desenvolvedor deve incluir na definição da base quais os atributos que devem ser encontrados (ou seja, os objetivos - *goals* - do Sistema Especialista). A máquina de inferência encarrega-se de encontrar uma atribuição para o atributo desejado nas conclusões das regras (após o ENTÃO...). Obviamente, para que a regra seja aprovada, suas premissas devem ser satisfeitas, obrigando a máquina a encontrar os atributos das premissas para que possam ser julgadas, acionando um encadeamento recursivo. Caso o atributo procurado não seja encontrado em nenhuma conclusão de regra, uma pergunta direta é feita ao usuário;

- Utilização de fatores de confiança: Sabemos que o conhecimento humano não é determinístico. Não há especialista que sempre se encontre em condições de afirmar determinada conclusão com certeza absoluta. Graus de confiança são freqüentemente atribuídos às suas respostas, principalmente quando existe mais de uma. Este, sem dúvida, é um dos mais fortes pontos críticos na elaboração de uma representação computacional do saber humano. Vejamos a dificuldade em representar a confiabilidade das informações:

    - Especialistas humanos não se sentem confortáveis em pensar em termos de probabilidade. Suas estimativas não precisam corresponder àquelas definidas matematicamente;
    - Tratamentos rigorosamente matemáticos de probabilidade utilizam informações nem sempre disponíveis ou simplificações que não são claramente justificáveis em aplicações práticas.

Na verdade, existem duas correntes de pensamento: aquela que utiliza fórmulas estatísticas rigorosas e aquela que utiliza uma abordagem *ad hoc* sobre os fatores de certeza, ou seja, mais generalizada e sem uma base matemática forte. O *Expert* SINTA utiliza atualmente uma abordagem não-estatística, mas baseada na Teoria dos Conjuntos;

- Ferramentas de depuração: A consulta se desenvolve por meio de menus de múltipla (ou única escolha) ou menus onde se pode entrar com valor numérico. Pode-se ainda, entrar com o grau de confiança da resposta. Graus de confiança são utilizados quando não se possui certeza absoluta sobre um fato. Assim sendo, pode-se expressar uma dúvida por meio de um número percentual, ou seja, a resposta terá validade no intervalo de zero a cem por cento. Cada resposta assinalada pode ter o seu próprio grau de confiança. Caso realmente não se sabe responder àquela pergunta, simplesmente pode-se deixar todas as alternativas em branco. O *Expert* SINTA entenderá que não se soube como responder ao questionamento apresentado;

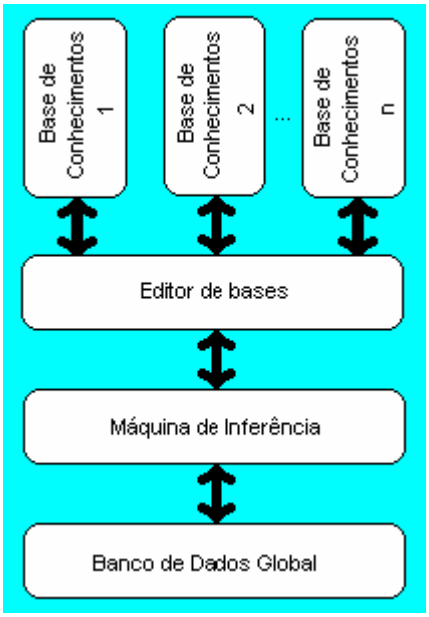
- Possibilidade de incluir ajudas *on-line* para cada base: Um bom Sistema Especialista não deve saber somente encontrar respostas, mas também fazer o usuário utilizar a resposta com fins práticos. Para isso, o *Expert* SINTA possibilita a inclusão de tópicos de ajuda para os valores possíveis de determinados atributos, associando a um arquivo no formato Ajuda do Windows a sua base. Explicações sobre a criação de arquivos de ajuda fogem ao escopo do presente documento, mas existem diversas documentações fornecidas por terceiros. O *Expert* SINTA também não distribui o compilador necessário para a criação do arquivo final. A idéia desses textos explicativos não é somente ativar uma ajuda *on-line* quando o usuário estiver utilizando o sistema, mas ir diretamente ao tópico explicativo de uma resposta, quando o *Expert* SINTA atinge um determinado objetivo.

4.3.3 ARQUITETURA

Os sistemas especialistas gerados no *Expert* SINTA seguem uma arquitetura conforme

Figura 4:

Figura 4 - Arquitetura de um Sistema Especialista no Expert SINTA



Fonte: (Nogueira, 1995)

- Base de conhecimento - É a informação que um especialista utiliza, representada computacionalmente;
- Editor de bases - Meio pelo qual a shell permite a implementação das bases desejadas;
- Máquina de inferência - Parte do sistema especialista responsável pelas deduções sobre a base de conhecimento;
- Banco de dados global - Representa as evidências apontadas pelos usuários do sistema especialista durante uma consulta.

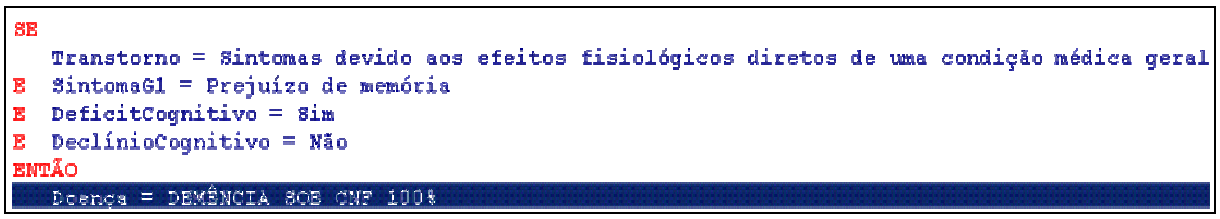
4.3.4 FORMA DE REPRESENTAÇÃO DO CONHECIMENTO

Na ferramenta *Expert* SINTA, a representação do conhecimento é feita através de regras de produção, podendo ainda, utilizar-se de probabilidades.

Sendo assim, o engenheiro do conhecimento pode se preocupar apenas com a representação do conhecimento do especialista.

Um exemplo da representação do conhecimento em regras de produção:

Figura 5 – Exemplo de regra de produção no Expert SINTA



Fonte: (Nogueira, 1995)

4.3.5 RECURSOS E FACILIDADES

Dentre muitas facilidades apresentadas pela ferramenta *Expert* SINTA, destacam-se:

- A interface apresenta grande facilidade de trabalho;
- Por trabalhar com regras de produção, possui modularidade (cada regra por si mesma) e facilidade de edição (podendo-se acrescentar ou excluir regras);
- Transparência do sistema: garante maior legibilidade da base de conhecimentos.

4.4 PROLOG

Conforme Rabuske (1995) PROLOG é uma linguagem declarativa, inicialmente concebida para o processamento de linguagem natural, orientada para o processamento simbólico, tendo bons recursos para o processamento de listas. Um programa escrito nesta linguagem consiste em um conjunto de regras que descrevem relações entre objetos. Programas em PROLOG envolve aspectos declarativos, que determinam se um certo objetivo é verdadeiro; e aspectos procedimentais que permitem melhorar a eficiência dos programas.

4.4.1 HISTÓRICO

A linguagem de programação PROLOG (*Programming in Logic)* nasceu no início da década de 70 em Edimburgo, Escócia. Primeiramente, sua utilização ficou restrita a algumas universidades e centros de pesquisa europeus e alcançou popularidade depois que o projeto japonês de computadores de quinta geração adotou-a como sua linguagem básica de desenvolvimento (Souza, 1998).

4.4.2 CARACTERÍSTICAS

Para Heinzle (1995), o Prolog é uma linguagem declarativa orientada para o processamento simbólico. Um programa escrito nessa linguagem consiste em um conjunto de

regras que descrevem relação entre objetos. Estas relações, chamadas de predicados do programa, são escritas a partir de um subconjunto do cálculo de predicados, denominado "cláusulas de *Horn*" - cláusula que tem no máximo um literal positivo.

Algumas das principais características da linguagem Prolog são:

- É uma linguagem orientada ao processamento simbólico;
- Representa uma implementação da lógica como linguagem de programação;
- Apresenta uma semântica declarativa inerente à lógica;
- Permite a definição de programas reversíveis, isto é, programas que não distinguem entre os argumentos de entrada e os de saída;
- Permite a obtenção de respostas alternativas;
- Suporta código recursivo e iterativo para a descrição de processos e problemas, dispensando os mecanismos tradicionais de controle, tais como *while*, *repeat*, etc.;
- Permite associar o processo de especificação ao processo de codificação de programas;
- Representa programas e dados através do mesmo formalismo;
- Incorpora facilidades computacionais extralógicas e metalógicas.

4.4.3 FORMA DE REPRESENTAÇÃO DO CONHECIMENTO

A forma de representação do conhecimento na linguagem Prolog é baseada na lógica dos predicados, e é definida sob os aspectos sintaxe, onde são descritas as possíveis configurações que podem constituir sentenças e a semântica, que determina os fatos do mundo aos quais as sentenças se referem.

4.4.4 RECURSOS E FACILIDADES

Arrolam-se abaixo as principais características que tornam PROLOG uma escolha incontestável:

- Facilidade de modificação de estruturas de dados grandes e complexas: Esta peculiaridade é de vital importância para o armazenamento de estruturas sintáticas,

estruturas semânticas bem como de entradas léxicas, elementos presentes em programas que manipulam qualquer linguagem natural.

- Capacidade de auto-análise e automodificação de programas: O suporte à metaprogramação é muito forte em PROLOG, o que possibilita a adoção de modelos abstratos de programação.
- Algoritmo *depth-first* embutido: Para a busca das informações concernentes a um programa PROLOG (fatos e regras), o método de busca *depth-first* é internamente utilizado em sistemas PROLOG. Este algoritmo é de extrema utilidade para a implementação de analisadores sintáticos, sendo adotado pela maciça maioria.
- Incorporação de DCG (*Definite Clause Grammar***)**: DCG é um formalismo que estende as gramáticas livres de contexto, possibilitando a identificação e utilização de inter-relacionamentos entre os componentes de uma regra gramatical. Além de cobrir a principal carência das gramáticas livres de contexto, as DCGs não oneram o processamento de sentenças de tamanho considerável, como o fazem as gramáticas sensíveis ao contexto. Deste modo, com as DCGs tem-se um formalismo de grande capacidade de expressão, aliada à eficiência. As gramáticas sensíveis ao contexto, quando empregadas no processamento de sentenças, denotam uma complexidade de tempo exponencial, em função do tamanho das entradas.

5 CLIPS

Segundo (Parpinelli, 2000), CLIPS (*C Language Integrated Production System* – ou Sistema de Produção Integrado da Linguagem C) é uma ferramenta de desenvolvimento de sistemas especialistas em que provê um ambiente completo para a construção de regras de objetos baseados em sistemas especialistas. Foi desenvolvida para facilitar o desenvolvimento de *softwares* que modelam o conhecimento humano.

O CLIPS está sendo usado por um grande número de usuários em uma comunidade que inclui usuários públicos e privados, em toda a *National Aeronautics and Space Administration* (NASA) e bases militares dos Estados Unidos, numerosas agências federais, universidades e muitas companhias.

5.1 HISTÓRICO

A origem do Sistema de Produção Integrado da Linguagem C (CLIPS , C Language Integrated Production System) data de antes de 1984 no Centro Espacial Johnson da NASA. Naquele tempo, a Seção de Inteligência Artificial havia sido desenvolvida através de uma dúzia de protótipos de aplicações de sistemas especialistas utilizando o estado-da-arte em matéria de *hardware* e *software*. Entretanto, apesar das extensas demonstrações do potencial do sistema especialista, poucas destas aplicações foram colocadas em uso regular. Esta falha no fornecimento de tecnologia para sistemas especialistas com os padrões de computação operacional da NASA poderia ser atribuído, em grande parte, ao uso do LISP como a principal linguagem para quase todas as ferramentas de sistemas especialistas naquele tempo. Em particular, três problemas dificultaram o uso do LISP como linguagem base no desenvolvimento de sistemas especialistas dentro da NASA: a baixa disponibilidade do LISP em uma grande variedade de computadores, o alto custo de ferramentas e *hardware* "estado-da-arte" para o LISP e a pobre integração do LISP com outras linguagens (tornando as aplicações "embutidas" mais difíceis) (Parpinelli, 2000).

A Seção de Inteligência Artificial propôs que o uso de uma linguagem convencional, como o C, eliminaria a maioria destes problemas. Inicialmente foram observados os sistemas especialistas comerciais para prover uma ferramenta de sistema especialista utilizando uma linguagem de programação convencional. Embora várias empresas estivessem convertendo suas ferramentas para rodar em C, o custo ainda era muito alto, sendo que a maioria

continuava restrita a uma pequena variedade de computadores, e o tempo de projeto disponível era desanimador. Conhecendo todas as necessidades de tempo e custo de maneira efetiva , ficou evidente que a Seção de Inteligência Artificial teria que desenvolver seu próprio sistema especialista baseado em C.

A versão "protótipo" do CLIPS foi desenvolvida na primavera de 1985 em pouco mais de 2 meses. Foi dada uma atenção particular para fazer com que a ferramenta continuasse compatível com os sistemas especialistas em desenvolvimento naquele tempo na Seção de Inteligência Artificial. Assim, a sintaxe do CLIPS foi feita com uma relação muito estreita para se assemelhar com a sintaxe de um subconjunto de ferramentas para sistemas especialistas "ART" desenvolvida pela *Inference Corporation*. Embora originalmente modelado a partir do ART, o CLIPS foi totalmente desenvolvido sem qualquer assistência da *Inference* ou acesso ao código-fonte do ART.

A intenção original do CLIPS era ganhar uma habilidade e conhecimento sobre a construção de ferramentas para sistemas especialistas e fixar uma base para o desenvolvimento de uma ferramenta para a substituição de ferramentas comerciais que eram usadas naquela época. A versão 1.0 demonstrou a viabilidade do conceito de projeto. Depois de um desenvolvimento adicional, ficou aparente que o CLIPS seria uma ferramenta para sistemas especialistas de baixo custo ideal para ser usado em treinamento. Após um ano de desenvolvimento e uso interno adicionaram ao CLIPS portabilidade, performance, funcionalidade, e um suporte documentado.

Por causa de sua portabilidade, extensibilidade, capacidades e baixo custo, o CLIPS teve uma grande aceitação difundida através do governo, indústria e acadêmica. O desenvolvimento de CLIPs ajudou a melhorar a habilidade para liberar tecnologia de sistemas especialistas ao longo dos setores públicos e privados para uma grande variedade de aplicações e ambientes de computação diversos. O CLIPS está sendo usado por mais de 5.000 usuários entre público e privado: todas as localidades da NASA, bases militares, numerosas agências do governo, universidades, e muitas companhias privadas.

O CLIPS é atualmente mantido como *software* de domínio público pelos seus principais autores que não trabalham mais para a NASA.

5.2 CARACTERÍSTICAS E ARQUITETURA

Portabilidade: CLIPS é escrito em C e sua portabilidade e velocidade foram mantidas mesmo instalado em diferentes computadores sem mudança em seu código-fonte. Os computadores nas quais o CLIPS foi testado incluem IBM PC (DOS e Win95), Macintosh MacOS. O CLIPS pode ser portado para qualquer máquina que possua um compilador C compatível com o padrão ANSI. O CLIPS é distribuído com todo seu código-fonte que pode ser modificado de acordo com as necessidades específicas de seus usuários.

Integração/Expansibilidade: O CLIPS pode ser embutido dentro de um código procedural, que é chamado como uma subrotina, e desta maneira integrado com linguagens como C, FORTRAN e ADA. O CLIPS pode ser facilmente expandido por regras definidas pelo usuário.

Desenvolvimento interativo: A versão padrão do CLIPS fornece um ambiente de desenvolvimento interativo, orientado-a-textos, incluindo *debugger*, ajuda on-line e um editor integrado. Em sua interface existem características como menus *pull-down*, editores integrados, e múltiplas janelas para ambientes Mac, Win95 e Xwindows.

Verificação/Validação: No CLIPS existem várias características para apoiar a verificação e validação de sistemas especialistas, incluindo suporte para desenvolvimento modular de uma base de conhecimento particionada, verificação estática e dinâmica de valores e argumentos de funções, e análise semântica de padrões e regras para determinar e ,se necessário , prevenir uma possível geração de erro.

5.3 FORMA REPRESENTAÇÃO DO CONHECIMENTO

O CLIPS fornece uma ferramenta coesiva para gerenciar uma grande variedade de conhecimento com suporte a dois diferentes paradigmas de programação: Conhecimento Heurístico (Regras), Conhecimento Procedural (Giarratano[1], 1998).

5.3.1 CONHECIMENTO HEURÍSTICO (REGRAS)

A programação Baseada em Regras permite representar o conhecimento como heurísticas, ou "*rules of thumb*" (regras do polegar) que especifica um conjunto de ações a ser executado para uma determinada situação. Como já foi visto, uma regra é composta por

premissas e conclusões. As premissas são também chamadas de *left-hand side* (LHS), enquanto que as conclusões podem ser chamadas de *right-hand side* (RHS).

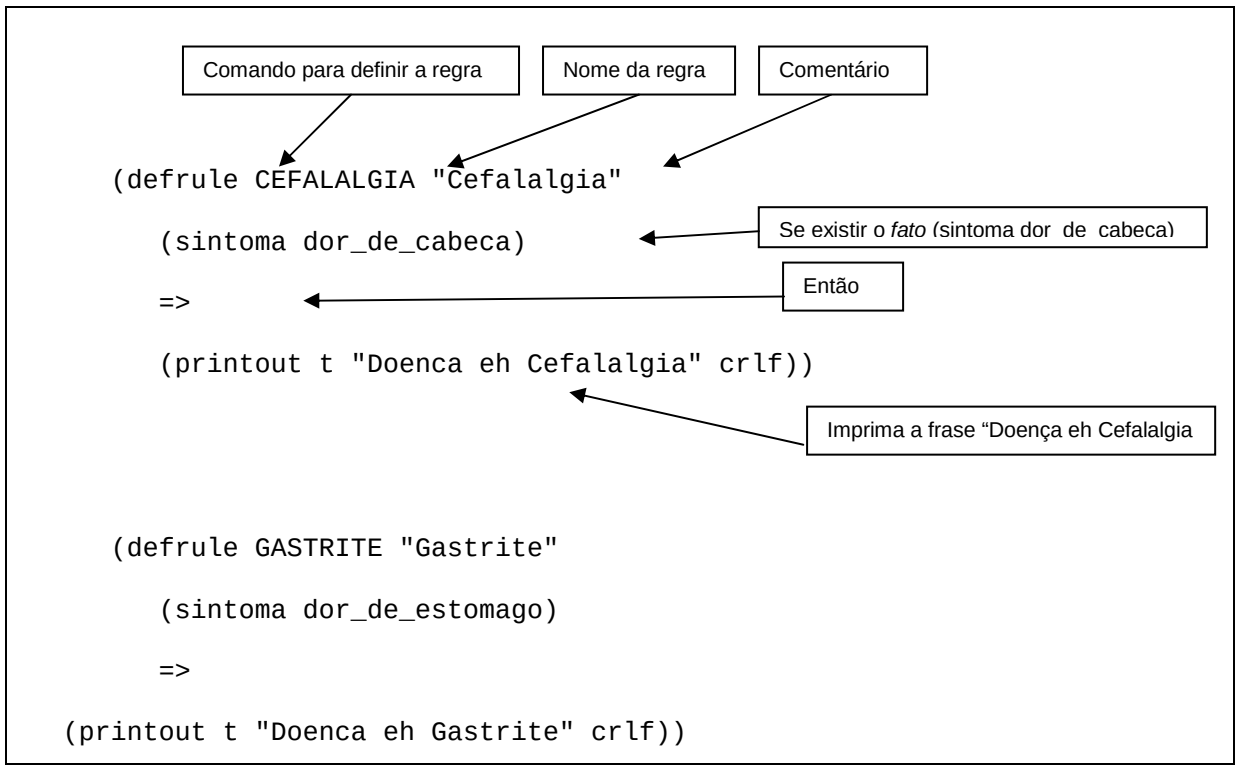
Em CLIPS, as premissas de uma regra são satisfeitas baseadas na existência ou não-existência de fatos, definidos em uma lista-de-fatos ou em instâncias especificadas pelo projetista.

Um tipo de condição que pode ser especificada, são os chamados padrões. Esses padrões consistem em um conjunto de restrições os quais são usados para determinar quais os fatos ou objetos satisfazem as condições especificadas por uma regra.

As conclusões serão o conjunto de ações a serem executadas quando uma regra for validada.

O Quadro 5 mostra um exemplo de regras de produção no CLIPS onde são definidas duas regras para verificar, através de um sintoma, qual pode ser a doença:

Quadro 5 – Exemplo de Regras no CLIPS



A partir do momento em que as regras estão definidas, basta inserir um fato dizendo que o paciente possui determinado sintoma. Isto é feito com o comando *(assert)*, conforme exemplo abaixo:

```
(assert (sintoma dor_de_estomago))
```

Após digitada a linha acima, a regra GASTRITE já é validada automaticamente. Ao rodar o programa *(run)*, todas as regras que estão validadas serão executadas. Neste caso, somente a regra GASTRITE foi validada e será impressa a mensagem “Doenca eh Gastrite”.

5.3.2 CONHECIMENTO PROCEDURAL

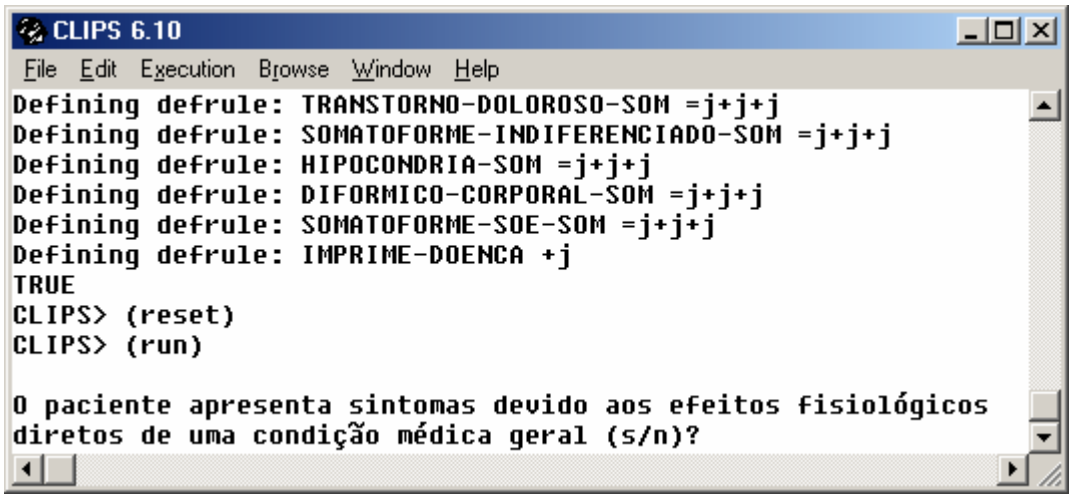
CLIPS também suporta um paradigma procedural para representar o conhecimento, da mesma forma que muitas linguagens convencionais como Pascal e C. Funções genéricas permitem ao projetista definir novos elementos executáveis para que o CLIPS possa realizar tarefas e retornar valores úteis. Essas novas funções podem ser chamadas de funções embutidas do CLIPS. Manipuladores de mensagens permitem ao projetista definir comportamentos de objetos através de respostas às mensagens.

Essas são características procedurais especificadas pelo projetista que o CLIPS executa impreterivelmente nos momentos apropriados.

5.4 AMBIENTE E INTERFACE

Para iniciar o ambiente CLIPS, deve-se executar a *shell* apropriada para cada sistema operacional. A interface é simples, baseada em comandos. Existem menus que poupam o projetista a digitar certos comandos. Confere-se na Figura 6:

Figura 6 – Ambiente CLIPS



5.5 MÁQUINA DE INFERÊNCIA

A máquina de inferência do CLIPS é o mecanismo que seleciona automaticamente os fatos existentes no estado atual e determina qual a regra que será aplicada (validada) (Giarratano[2], 1998).

As ações de uma regra validada serão executadas quando a máquina de inferência for instruída para iniciar a execução de regras validadas. Se mais de uma regra se aplica ao estado atual, a máquina de inferência utiliza um mecanismo chamado estratégia de resolução de conflitos para selecionar qual a regra deverá ter suas ações executadas. O CLIPS oferece sete modos de resolução de conflitos: *depth*, *breadth*, LEX, MEA, *complexity*, *simplicity* e *random*. As ações de uma regra executada podem afetar a lista de regras validadas. A máquina de inferência, então, seleciona outra regra e executa suas ações. Este processo continua até não existirem mais regras validadas.

5.6 WXCLIPS

Segundo Orchard (2000), wxCLIPS é uma ferramenta desenvolvida pelo AIAI - *Artificial Intelligence Applications Institute*, Universidade de Edinburgh, Reino Unido, em 1994. O desenvolvimento de tal ferramenta foi motivado pela necessidade de suprir a deficiência que os programas feitos em CLIPS ou FuzzyCLIPS apresentam quanto à criação de interfaces.

wxCLIPS pode ser considerado em dois aspectos: (i) uma biblioteca de funções CLIPS/FuzzyCLIPS, para acessar as facilidades do wxWindows; (ii) um ambiente de desenvolvimento de aplicações wxWindows, usando as funções CLIPS/FuzzyCLIPS. A biblioteca pode ser usada por qualquer programa C++; wxCLIPS é então, uma simples interface de desenvolvimento *for Windows*, que usa a biblioteca de funções CLIPS/FuzzyCLIPS.

5.6.1 AMBIENTE DE DESENVOLVIMENTO WXCLIPS

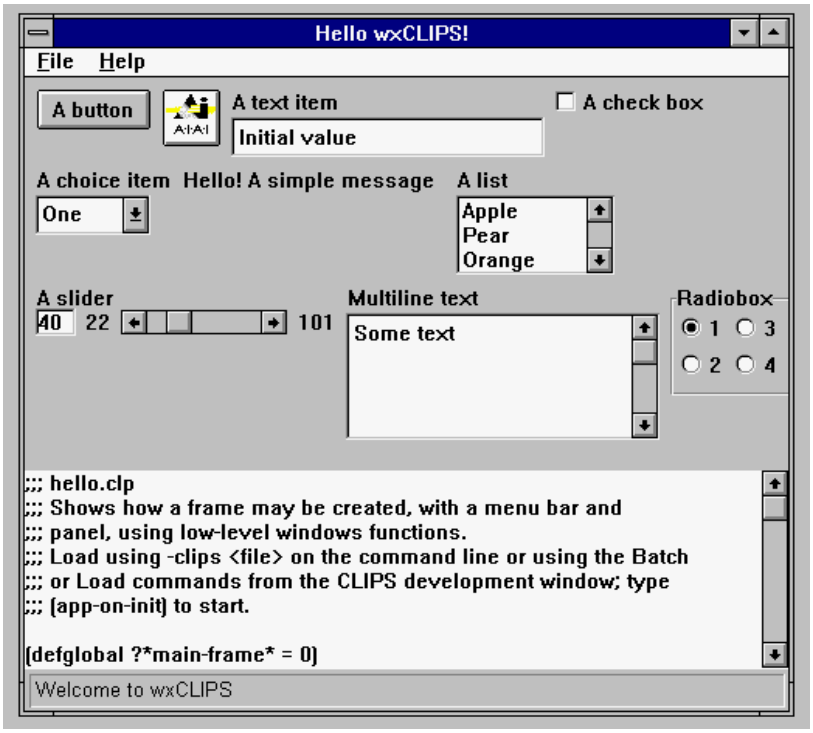
O wxCLIPS possui um ambiente de desenvolvimento básico, que consiste em uma janela com menus, uma janela para entrada de dados e uma janela para os textos de saída. Durante a compilação dos programas, as mensagens de erro são escritas na janela de saída.

Usando o wxCLIPS, pode-se criar *frames*, cada um com seus próprios menus. Dentro de um *frame* pode-se criar uma ou mais subjanelas. Estas subjanelas podem ser *panels*, *canvases* e subjanelas de texto:

- *Panels* são usados para conter os *panel items*, tais como botões, itens para entrada de texto, *box* de listas, etc. O *box* de diálogo é uma forma especial de *panel*, que contem seu próprio *frame*, sendo assim, ao invés de criar um *frame* e um *panel*, basta criar um box de diálogo e inserir os *panel items* necessários.
- *Canvases* são usados para desenhar figuras com qualquer formato.
- As subjanelas de texto são usadas para exibir arquivos texto ou editar os programas.

Não há necessidade de criar um box de diálogo ou manipular todos os botões, e outros eventos manualmente. Há um certo número de funções que desempenham todas estas tarefas, tais como: get-text-from-user, message-box, get-choice, file-selector, etc. Tais funções bloqueiam o fluxo de execução do programa no ponto onde foram chamadas, até que o usuário responda às solicitações. A Figura 7 exibe algumas das opções de interface que o wxCLIPS fornece.

Figura 7 - Exemplo de *Interfaces* do wxCLIPS



Há uma única função obrigatória a todos os programas que são executados no wxCLIPS: app-on-init. Se uma aplicação define uma função com este nome, a interface wxCLIPS do usuário pode inicializar a aplicação e estabelecer todas as variáveis de ambiente, arquivos e funções que serão usadas no decorrer da execução.

6 ENGENHARIA DE SOFTWARE E FERRAMENTAS

Este capítulo destina-se a introduzir o conceito de qualidade de *software* definida pela norma brasileira, e abordar os aspectos da Engenharia de *Software* que serão utilizados para avaliar as ferramentas para sistemas especialistas.

6.1 QUALIDADE DE SOFTWARE

Para a atual norma brasileira sobre o assunto (NBR ISO 8482), qualidade de *software* pode ser definida como: “A totalidade das características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas”. Onde entidade é o produto que pode ser um bem ou um serviço. Necessidades explícitas são as condições e objetivos propostos pelo cliente e implícitas são as necessidades que variam de cliente para cliente como implicações éticas e de segurança.

Existem instituições normalizadoras reconhecidas a nível mundial como:

- ISO – *International Organization for Standardization*;
- IEEE – Instituto de Engenharia Elétrica e Eletrônica;
- IEC – *International Eletrothcnical Comission*.

No Brasil, a entidade que se responsabiliza pela adoção de normas e padrões é a ABNT (Associação Brasileira de Normas Técnicas), que também é responsável pela pesquisa e desenvolvimento de normas, muitas vezes, em conjunto com os organismos internacionais.

Tanto a qualidade no produto final como no seu processo de produção são de grande importância para que se possa atingir da melhor maneira possível a satisfação do cliente, no entanto durante muito tempo apenas se considerou o produto final. Com a evolução das pesquisas e a crescente necessidade de elevação da qualidade dos produtos devido à forte concorrência, percebeu-se que o processo de construção do produto é um fator importantíssimo e de relevância fundamental na obtenção da qualidade esperada do produto (Costa, 2001) *apud* (Souza, 2001).

Dentre as normas de qualidade de *software*, as principais são:

- ISO 9126 - Característica da qualidade de produtos de *software*;
- NBR 13596 - Versão brasileira da ISO 9126;

- ISO 12119 - Característica de qualidade de Pacotes de *Software* (*software* de prateleira);
- ISO 9001- Modelos de qualidade em Projeto, Desenvolvimento, Instalação e Assistência Técnica (processo);
- ISO 9000-3 - Aplicação da norma ISO 9000 para desenvolvimento de *software*;
- ISO 10011 - Auditoria de sistemas de qualidade (processo);
- ISO 14598 - Guias para a avaliação da qualidade;
- ISO 15504 - SPICE, projeto da ISO para a avaliação do Processo de Desenvolvimento (*Software Process Improvement and Capability dEtermination*).

Sabendo-se o que é qualidade e como ela pode ser avaliada, pode se tentar aplicar estes conceitos aos produtos de *software* e ao processo de desenvolvimento de *software*. Inicialmente, encontra-se um grande problema: muitos desenvolvedores ainda acreditam que criar programas é uma arte que não pode seguir regras, normas ou padrões.

As instituições normalizadoras, citadas anteriormente, desenvolveram conjuntos de normas para serem aplicadas a produção de *software*. As normas foram desenvolvidas considerando-se:

a) produtos de *software* são complexos, até mais do que o hardware que o executam;

b) *software* não tem produção em série. Seu custo está no projeto e desenvolvimento;

c) *software* não se desgasta e nem se modifica com o uso;

d) *software* é invisível. Sua representação em grafos e diagramas não é precisa;

e) a Engenharia de *Software* ainda não está madura, é uma tecnologia em evolução;

f) não há um acordo entre os profissionais da área sobre o que é qualidade de *software*.

Mesmo levando-se em consideração todos esses fatores é importante ressaltar que o grande problema na definição de padrões de qualidade para a produção de *software* é o modo como os desenvolvedores têm feito isto durante vários anos, artesanalmente por isso é necessária a aplicação dos conceitos de qualidade na indústria de *software* urgentemente.

Por esse motivo, proceder-se-á nesse trabalho a aplicação de uma das principais abordagens de certificação de produtos de *software*, a ISO/IEC 9126.

6.2 ISO/IEC 9126

A qualidade de produtos de *software* está descrita na norma ISO/IEC 9126, publicada em 1991, tendo sido traduzida no Brasil como NBR 13596 e publicada em agosto de 1996. Essa norma lista o conjunto de características que devem ser verificadas em um *software* para que ele seja considerado um "*software* de qualidade".

Seu objetivo é definir as principais características que devem ser encontradas em um *software* dito de qualidade, esta norma leva em consideração o produto pronto. Ela se divide em seis grandes características que são:

- Funcionalidade – determina se o *software* satisfaz as necessidades do cliente;
- Confiabilidade – determina se o *software* é imune a falhas;
- Usabilidade – determina se o *software* é fácil de ser usado;
- Eficiência – determina se o *software* é rápido;
- Manutenibilidade – determina se é simples de fazer manutenções no *software*;
- Portabilidade – determina se o *software* é fácil de usar em outro ambiente.

Cada uma destas características gera um conjunto de subcaracterísticas sendo que cada uma destas possui uma pergunta chave, como apresentado no **Quadro 6** proposto por:

Quadro 6 - Conjunto de características e subcaracterísticas da ISO/IEC 9126

Característica	Subcaracterística	Pergunta chave para a subcaracterística
Funcionalidade (satisfaz as necessidades?)	Adequação	Propõe-se a fazer o que é apropriado?
	Acurácia	Faz o que foi proposto de forma correta?
	Interoperabilidade	Interage com os sistemas especificados?
	Conformidade	Está de acordo com as normas, leis, etc.?
	Segurança de acesso	Evita acesso não autorizado aos dados?
Confiabilidade (é imune a falhas?)	Maturidade	Com que freqüência apresenta falhas?
	Tolerância a falhas	Ocorrendo falhas, como ele reage?

	Recuperabilidade	É capaz de recuperar dados em caso de falha?
Usabilidade (é fácil de usar?)	Intelegibilidade	É fácil entender o conceito e a aplicação?
	Apreensibilidade	É fácil aprender a usar?
	Operacionalidade	É fácil de operar e controlar?
Eficiência (é rápido e "enxuto"?)	Tempo	Qual é o tempo de resposta, veloc. Execução?
	Recursos	Quanto recurso usa? Durante quanto tempo?
Manutenibilidade (é fácil de modificar?)	Analisabilidade	É fácil de encontrar uma falha, quando ocorre?
	Modificabilidade	É fácil modificar e adaptar?
	Estabilidade	Há grande risco quando se faz alterações?
	Testabilidade	É fácil testar quando se faz alterações?
Portabilidade (é fácil de usar em outro ambiente?)	Adaptabilidade	É fácil adaptar a outros ambientes?
	Capac. de instalação	É fácil instalar em outros ambientes?
	Conformidade	Está de acordo com padrões de portabilidade?
	Capac. para substituir	É fácil usar para substituir outro?

6.2.1 OBJETIVOS

Conforme Storch (2000), a norma ISO/IEC 9126 é um conjunto de padrões dirigido àqueles que estão envolvidos com aquisição, desenvolvimento, uso, suporte, manutenção e auditoria de *software*.

As organizações podem usar estes padrões em grande número de aplicações:

a) na possibilidade de determinação, pela organização, na capacidade de determinado *software* suportar um padrão internacional reconhecido;

b) para auxiliar a organização a melhorar seu próprio processo de desenvolvimento e manutenção de *software*;

c) como autojulgamento, para auxiliar uma organização a determinar sua capacidade de implementar um novo projeto de *software*.

7 O PROTÓTIPO

A idéia inicial para o tema do protótipo surgiu ao solicitar a ajuda de um especialista de conhecimento em psicologia. Dentre às várias subáreas da psicologia, o especialista sugeriu o Diagnóstico de Transtornos Mentais, embasado no fato de que, algumas vezes, pode ser complicado para o especialista chegar a uma conclusão sobre o estado do paciente.

O objetivo do protótipo proposto será diagnosticar qual a doença que determinado paciente possui, analisando os sintomas do mesmo. Antes de iniciar o protótipo propriamente dito, o mesmo foi implementado na ferramenta *Expert SINTA* para avaliar a viabilidade do problema.

Neste capítulo, tratar-se-á como este problema foi estudado e elaborado para a especificação deste protótipo.

7.1 AVALIAÇÃO DO PROBLEMA

Em diversas situações, as pessoas podem apresentar diferentes tipos de sintomatologias. Estas podem ter diversas causas sendo elas ambientais, hereditárias, neurológicas ou físicas. Algumas destas sintomatologias podem significar distúrbios psiquiátricos. Nestes casos, eles devem ter acompanhamento médico e psicológico. Mas para isso, é preciso saber qual a patologia apresentada pelo paciente – segundo Ferreira (1998), patologia são as modificações estruturais e/ou funcionais produzidas pela doença no organismo. Desta forma, o tratamento poderá ser feito de forma mais eficaz.

Em alguns casos, apenas a psicoterapia já é o suficiente para que o paciente obtenha alta, em outros, é necessário o uso de medicamentos, ou até mesmo a internação. Tudo isso dependerá do tipo de patologia e do grau da mesma.

O diagnóstico é baseado no conjunto de sintomas apresentados pelo paciente. Muitas vezes, um mesmo sintoma poderá fazer parte de mais de uma patologia. Em outros casos, poderá haver cruzamento de sintomas, o que dificulta o diagnóstico da doença. Para que este seja feito de forma eficaz, é preciso avaliar todos os sintomas e o contexto destes. Isto poderá ser feito através da conversa com o paciente e anamnese psiquiátrica - segundo Ferreira (1998), anamnese é a informação acerca do princípio e evolução de uma doença até a primeira observação do médico. Isso é feito através de entrevista, na qual avalia-se antecedentes familiares, início e evolução da doença, humor, tenacidade, consciência.

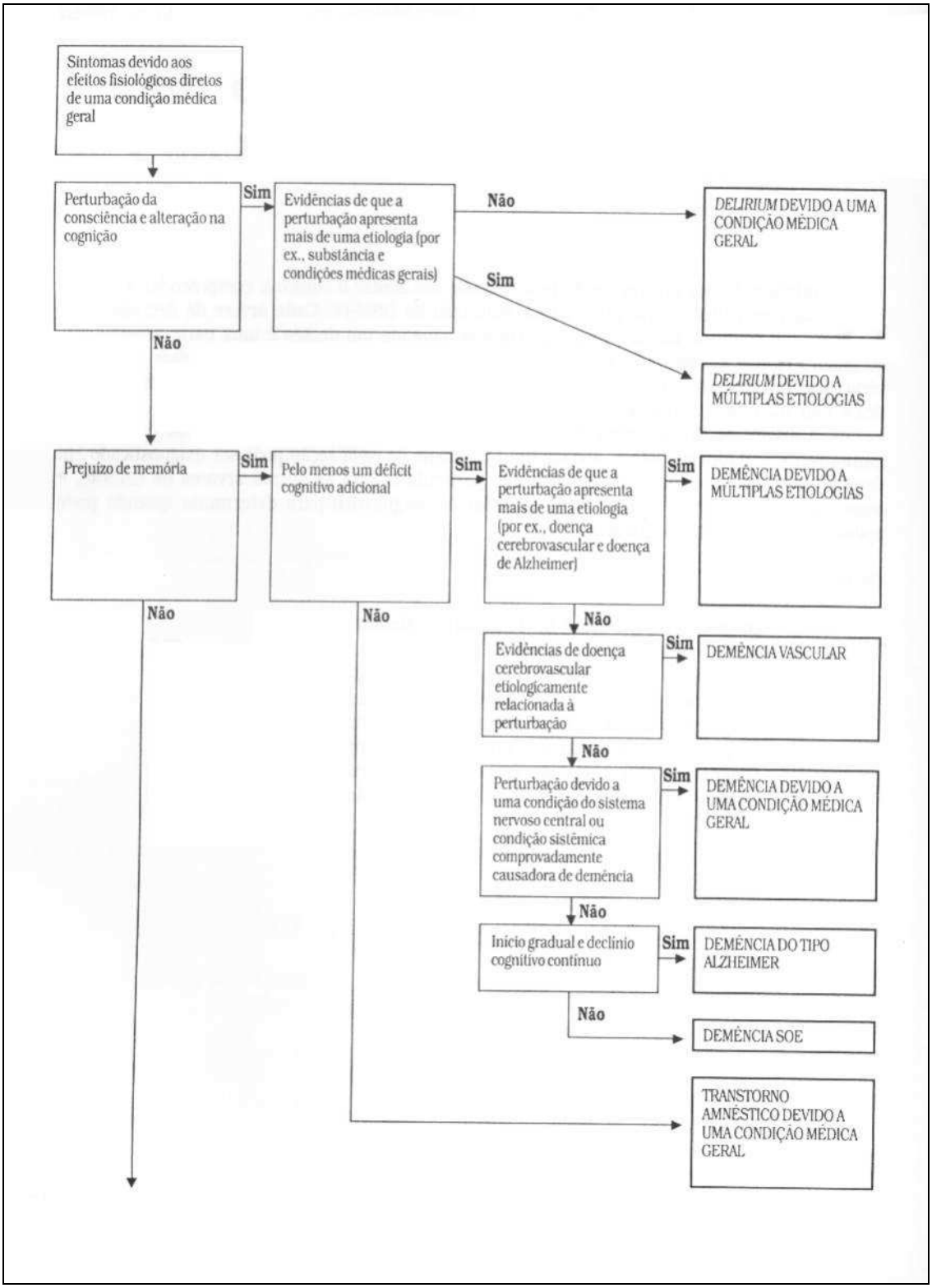
Todas as doenças mentais catalogadas, cerca de 80, encontram-se descritas no Manual Diagnóstico de Transtornos Mentais. Deste manual, juntamente com o auxílio do especialista de conhecimento, foram tiradas todas informações necessárias para montar a base de conhecimento.

As doenças estão divididas em seis grupos gerais:

- Transtornos mentais devido a uma condição médica geral;
- Transtornos mentais induzidos por substâncias;
- Transtornos psicóticos;
- Transtornos do Humor;
- Transtornos de Ansiedade;
- Transtornos Somatoformes.

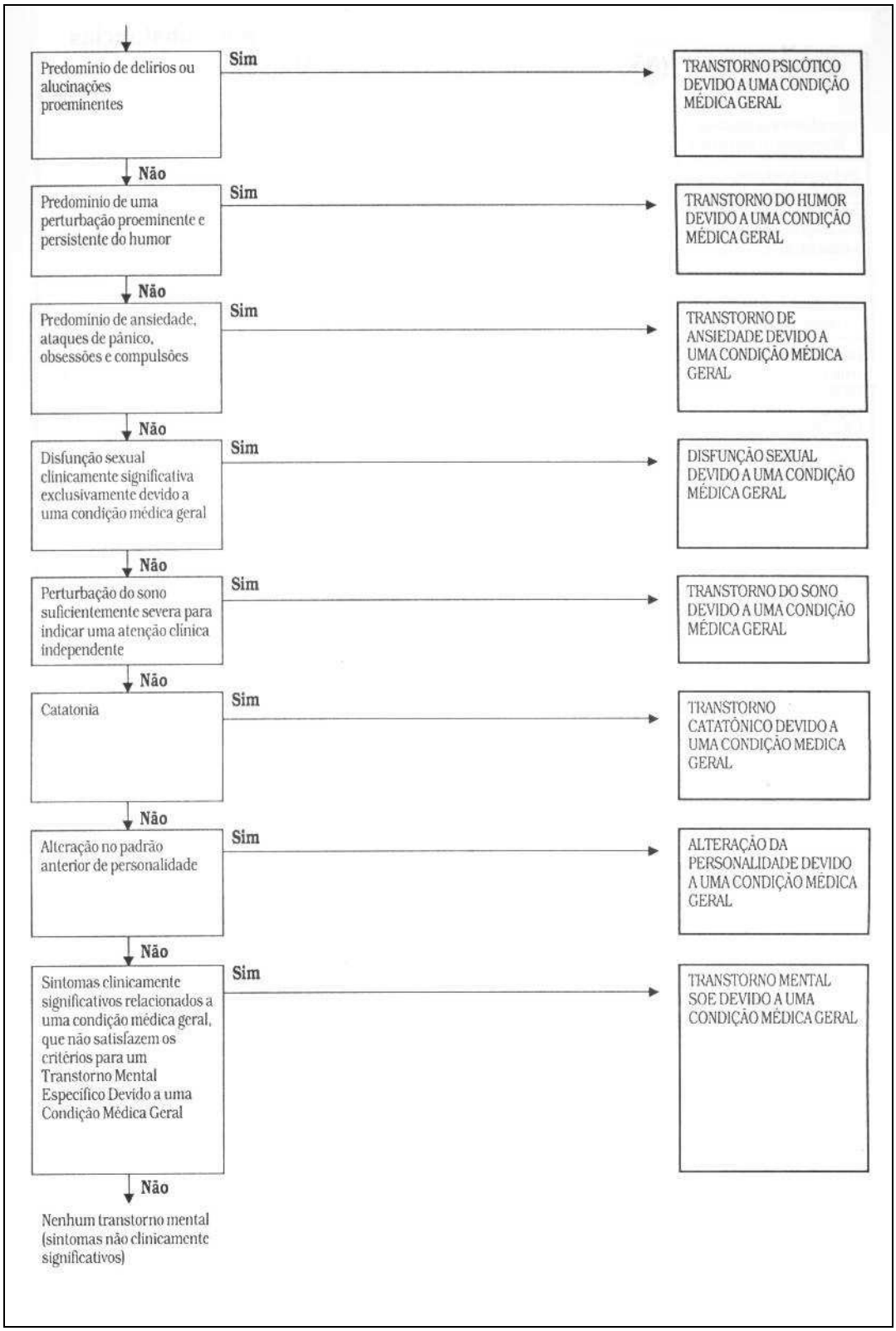
Dentro de cada grupo, existem os vários sintomas que são utilizados para se chegar às doenças. A melhor maneira de visualizar e facilitar o entendimento de como se dá esse processo, é através de árvores de decisão, apresentadas na **Figura 8** à **Figura 19.**

Figura 8 – Árvore de Decisão dos Transtornos devido a uma condição médica geral



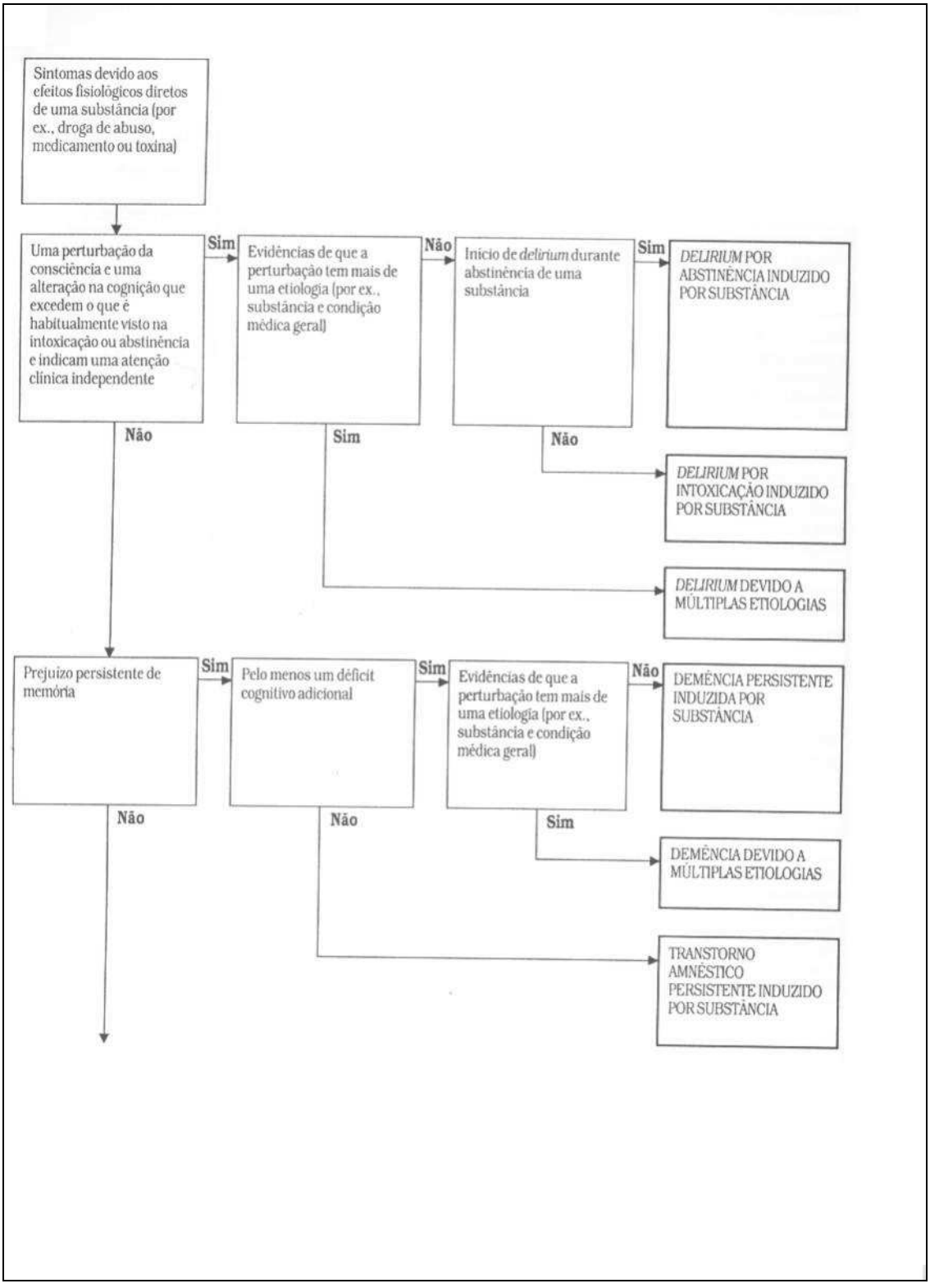
Fonte: (Jorge, 1995)

Figura 9 – Árvore de Decisão dos Transtornos devido a uma condição médica geral (cont.)



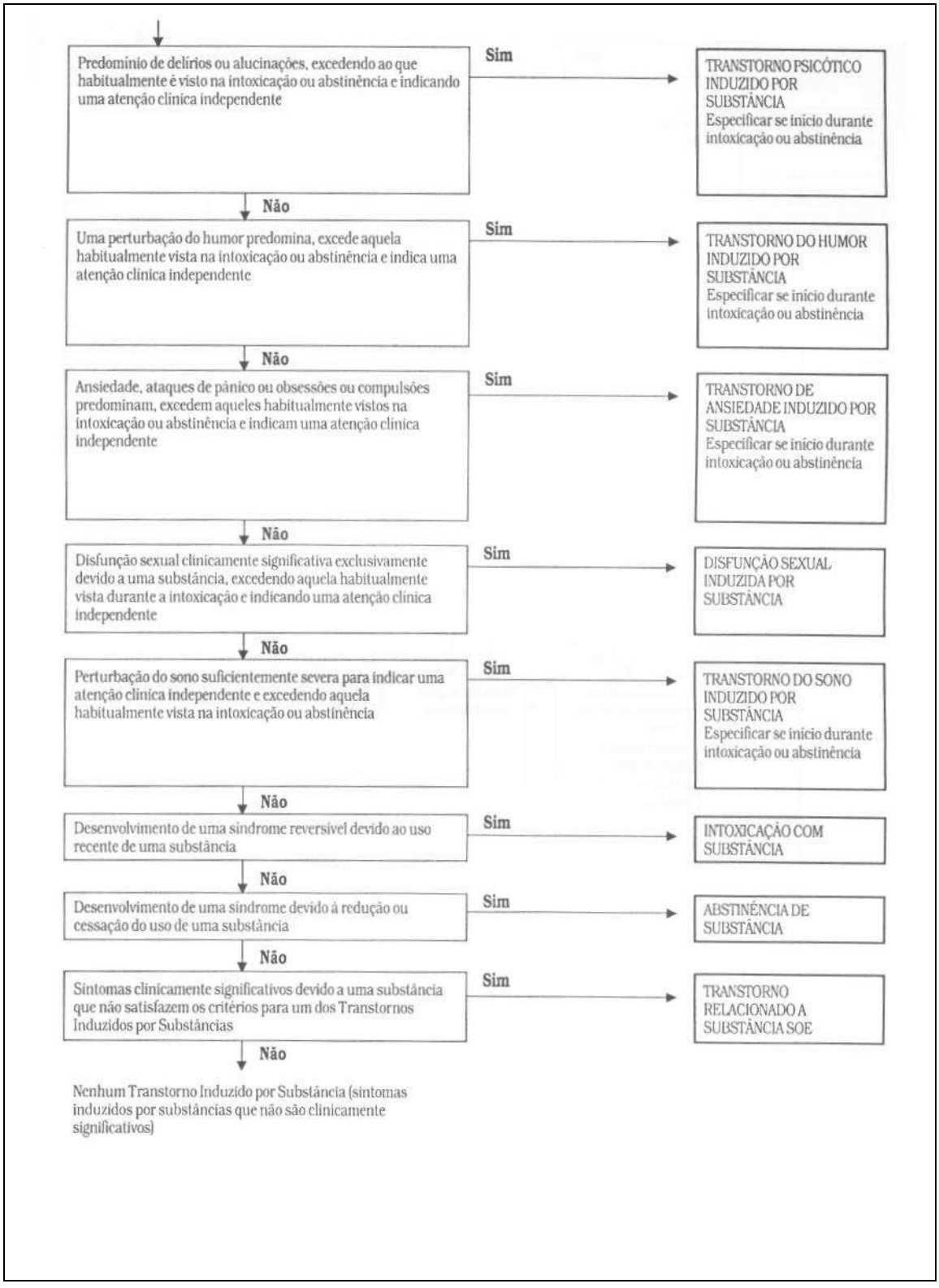
Fonte: (Jorge, 1995)

Figura 10 – Árvore de Decisão dos Transtornos induzidos por substâncias



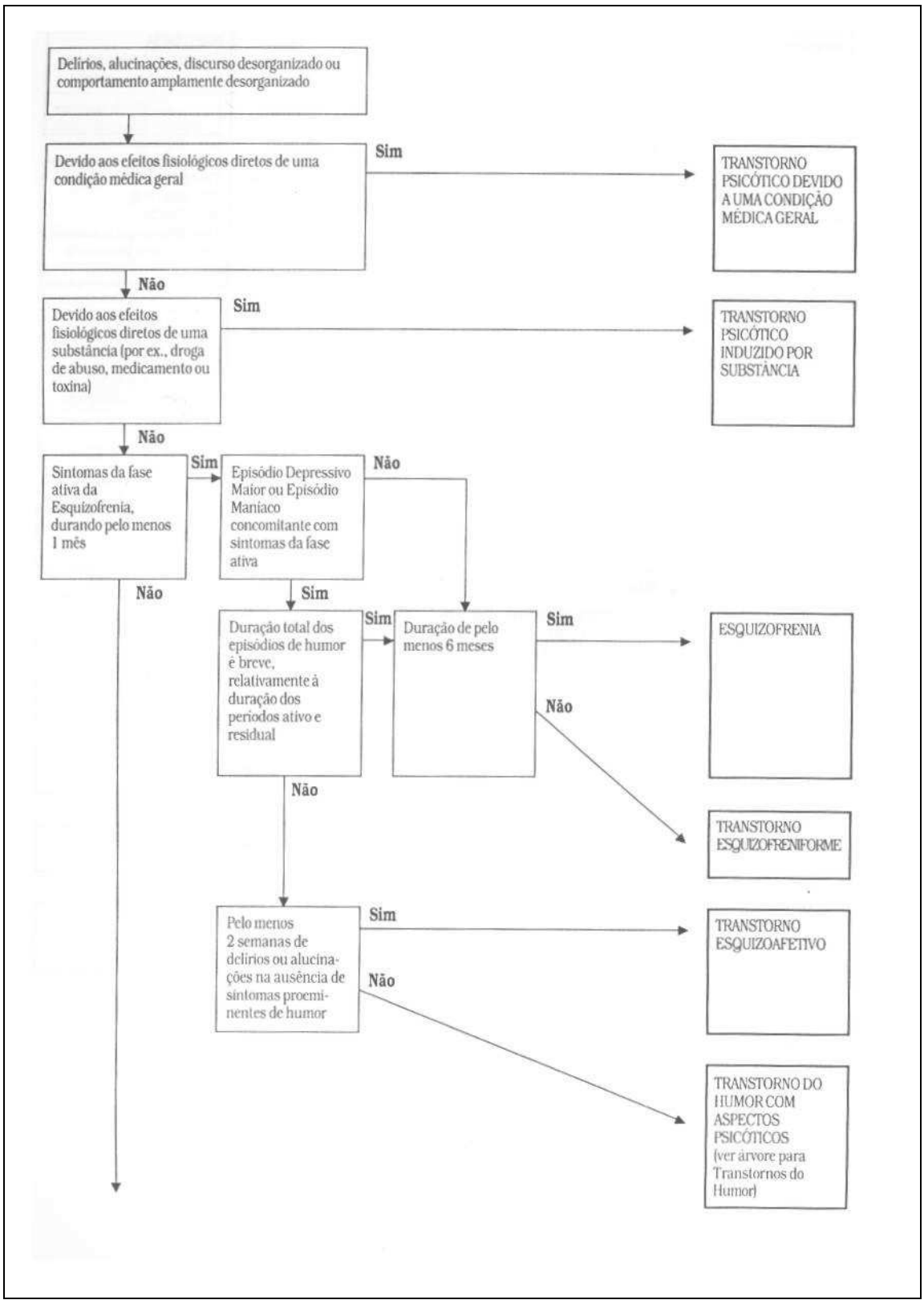
Fonte: (Jorge, 1995)

Figura 11 - Transtornos Induzidos por Substâncias (Cont.)



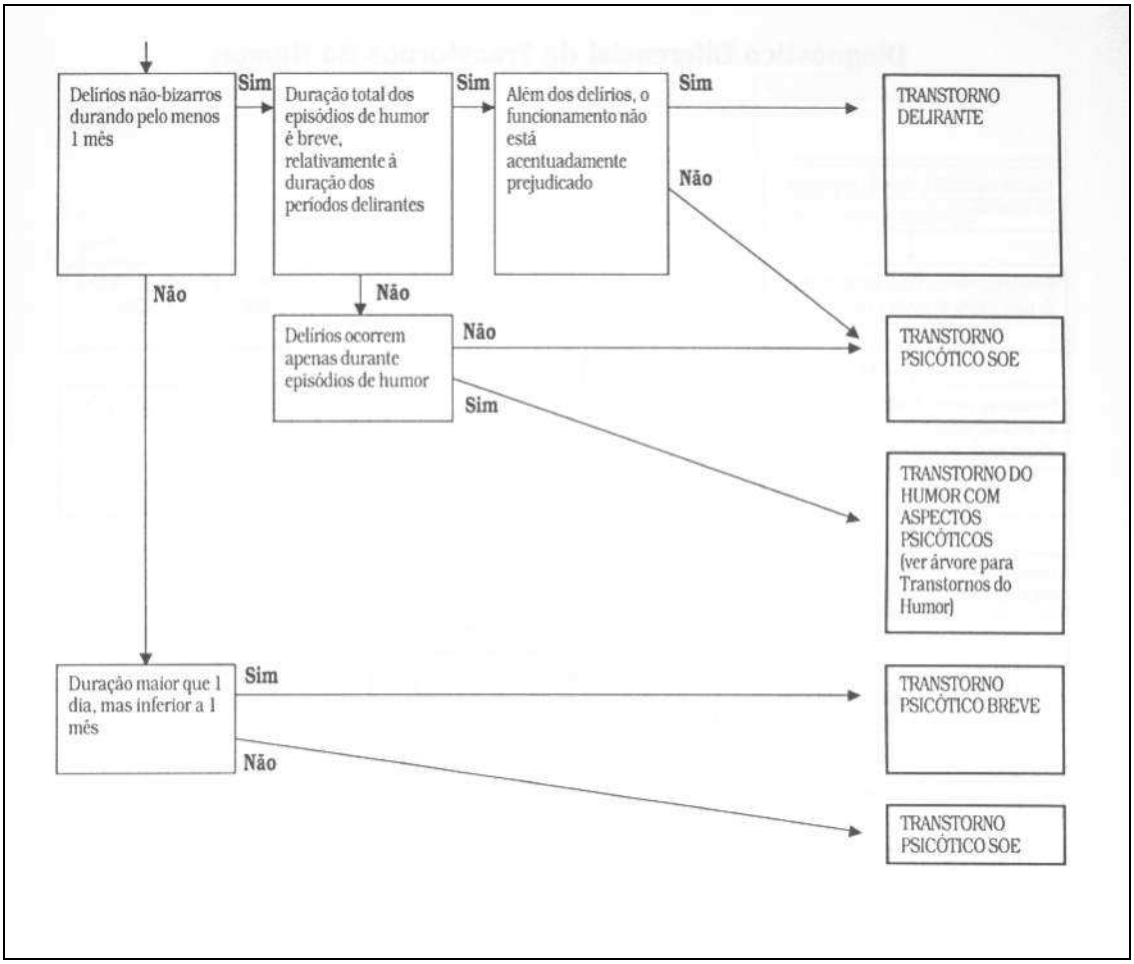
Fonte: (Jorge, 1995)

Figura 12 – Árvore de Decisão dos Transtornos Psicóticos



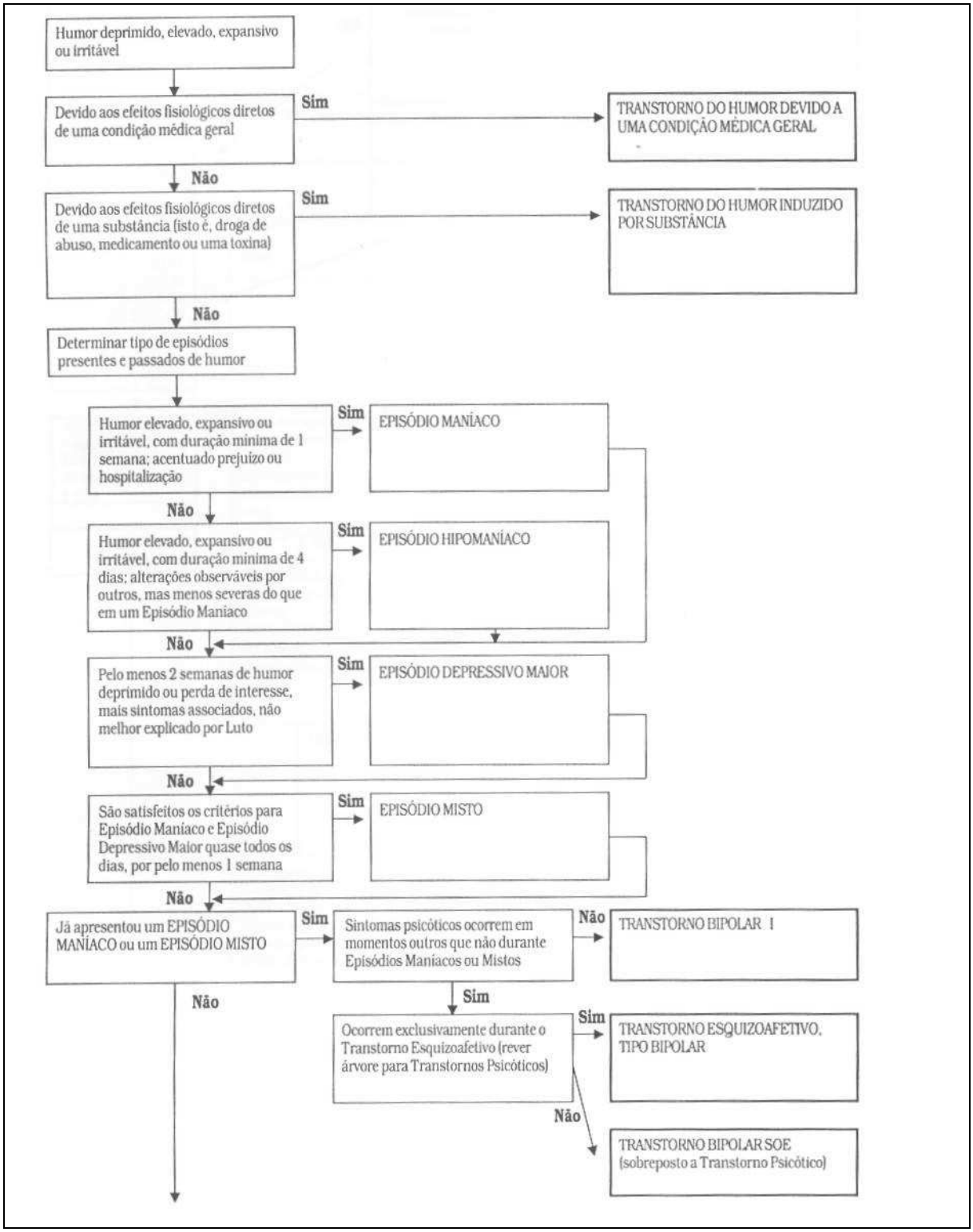
Fonte: (Jorge, 1995)

Figura 13 – Árvore de Decisão dos Transtornos Psicóticos (cont.)



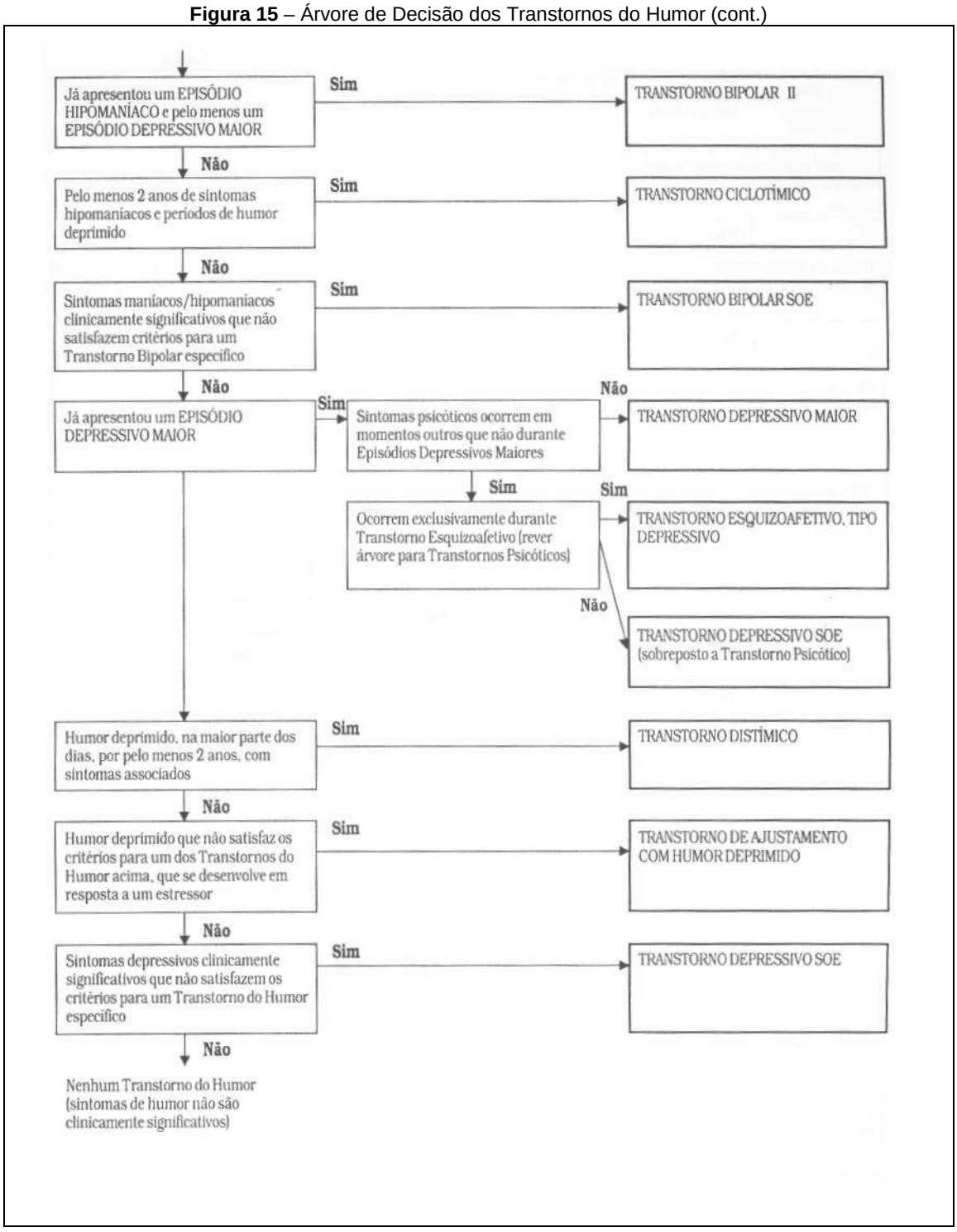
Fonte: (Jorge, 1995)

Figura 14 – Árvore de Decisão dos Transtornos do Humor



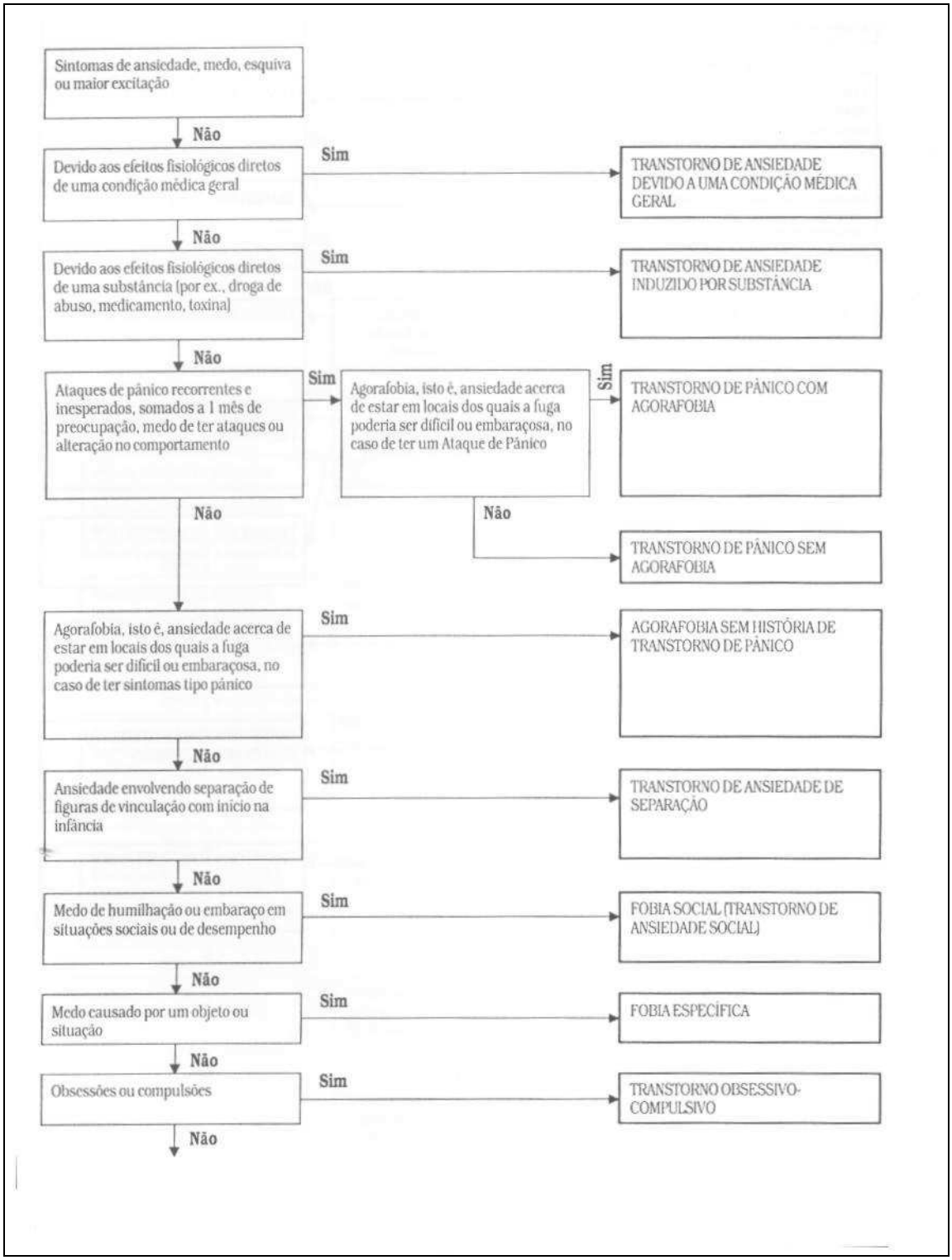
Fonte: (Jorge, 1995)

Figura 15 – Árvore de Decisão dos Transtornos do Humor (cont.)



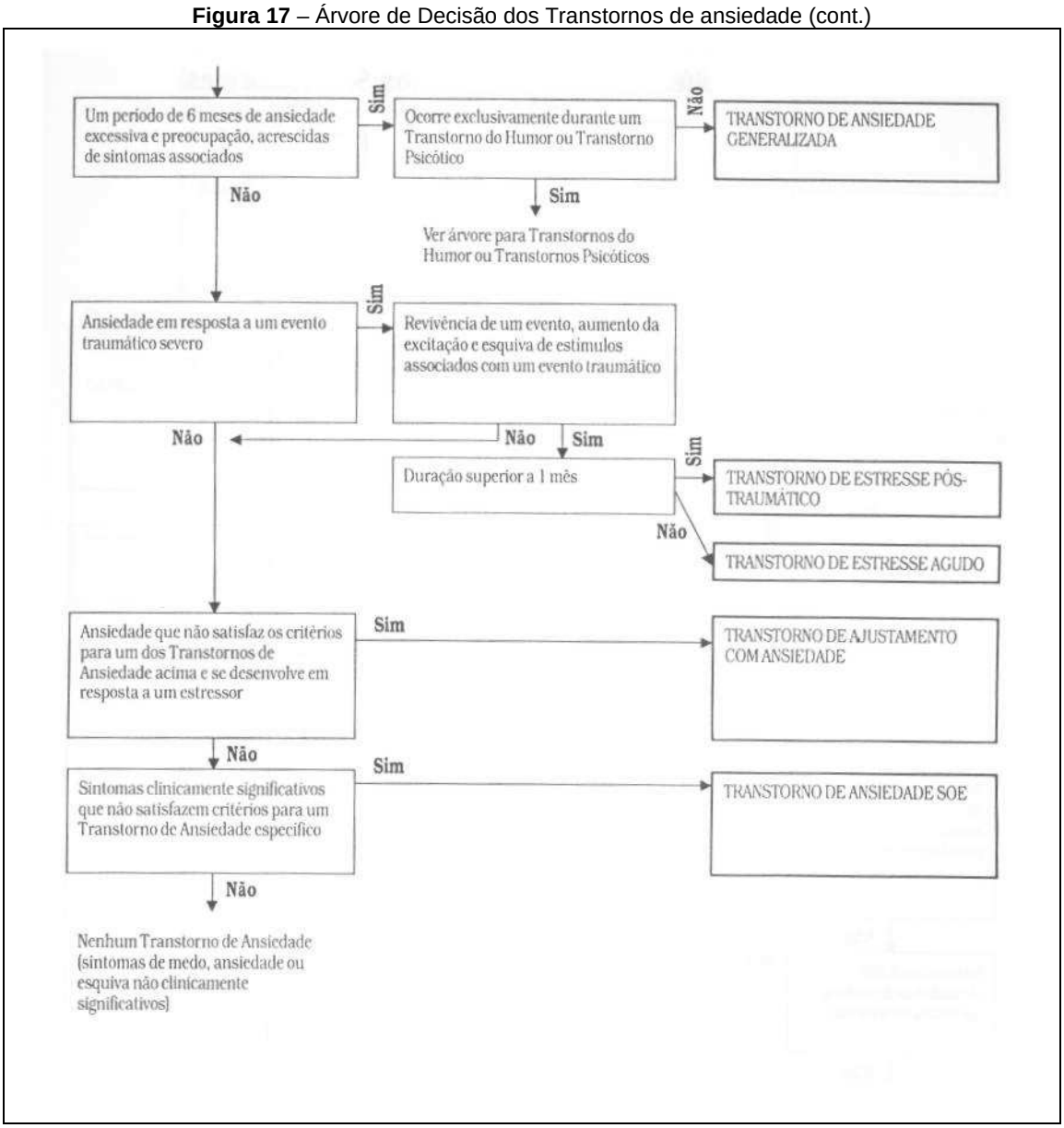
Fonte: (Jorge, 1995)

Figura 16 – Árvore de Decisão dos Transtornos de Ansiedade



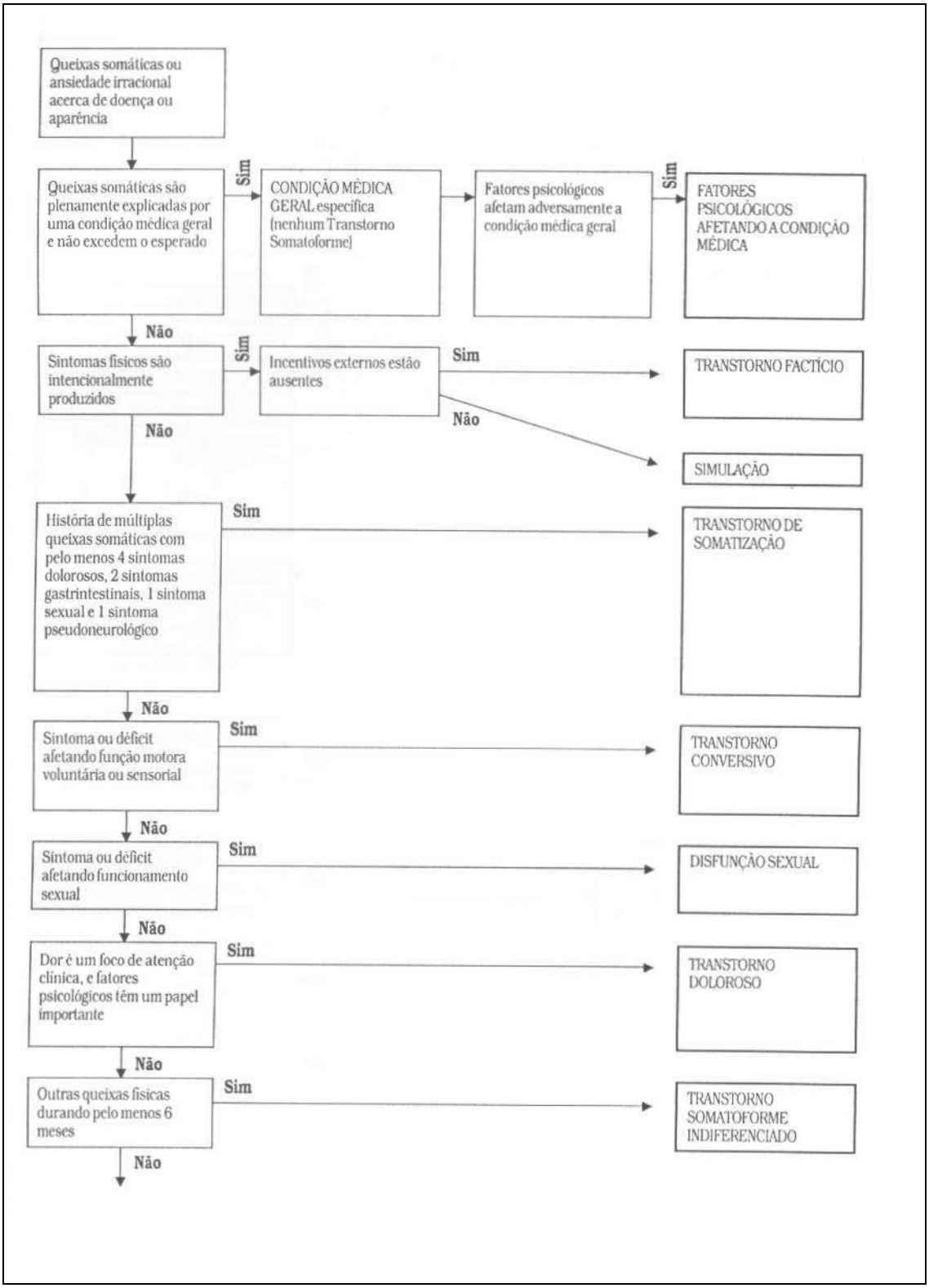
Fonte: (Jorge, 1995)

Figura 17 – Árvore de Decisão dos Transtornos de ansiedade (cont.)



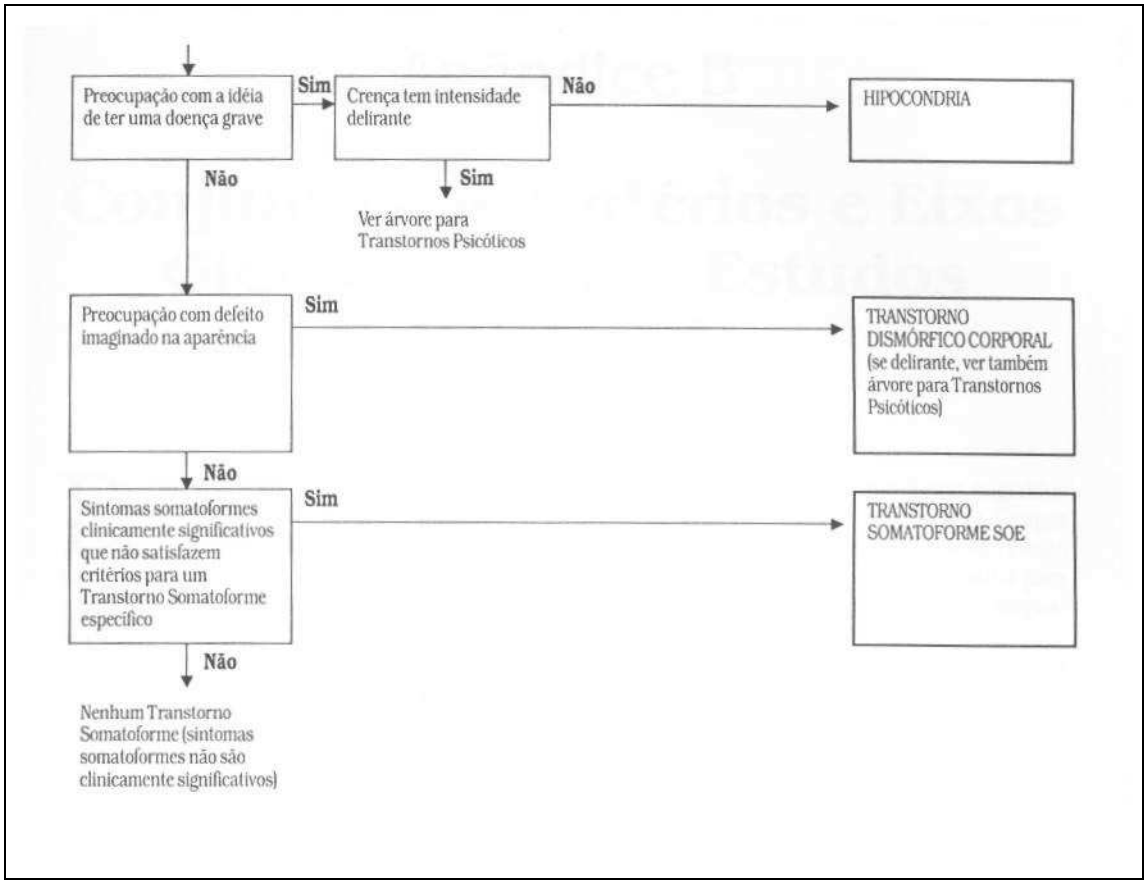
Fonte: (Jorge, 1995)

Figura 18 – Árvore de Decisão dos Transtornos Somatoformes



Fonte: (Jorge, 1995)

Figura 19 – Árvore de Decisão dos Transtornos Somatoformes (cont.)



Fonte: (Jorge, 1995)

7.2 ESPECIFICAÇÃO

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. Desta forma, escolheu-se também as regras de produção como forma de especificação do problema, conforme implementação especificada a seguir:

Primeiramente, são descritas duas funções com a finalidade de ajudar o projetista a fazer a pergunta ao usuário. Estas duas funções são comuns a vários sistemas desenvolvidos em CLIPS, conforme Quadro 7.

Quadro 7 – Funções para auxiliar a fazer perguntas ao usuário

```
(deffunction ask-question (?question $?allowed-values)
   (printout t ?question)
   (bind ?answer (read))
   (if (lexemep ?answer)
       then (bind ?answer (lowcase ?answer)))
```

```
    (while (not (member ?answer ?allowed-values)) do
       (printout t ?question)
       (bind ?answer (read))
       (if (lexemep ?answer)
           then (bind ?answer (lowcase ?answer))))
    ?answer)

(deffunction SN (?question)
   (bind ?response (ask-question ?question s n))
   (if (eq ?response s)
       then TRUE
    else FALSE))
```

Em segundo lugar, são definidas as regras para separar os grupos gerais de sintomas, especificadas no Quadro 8.

Quadro 8 – Funções para separar os grupos gerais de sintomas

```
(defrule grupo-condicao_medica_Geral ""
  (not(doenca $?))
=>
  (if(SN "
O paciente apresenta sintomas devido aos efeitos fisiológicos
diretos de uma condição médica geral (s/n)? ")
    then (assert (grupo CMG))))

(defrule grupo-Substancia ""
  (not(doenca $?))
=>
  (if(SN "
O paciente apresenta sintomas devido aos efeitos fisiológicos diretos
de uma substância (ex.: droga de abuso, medicamento ou toxina (s/n)?
")
    then (assert (grupo Substancia))))

(defrule grupo-Transtornos_Psicoticos ""
  (not(doenca $?))
=>
  (if(SN "
O paciente delírios, alucinações, discurso desorganizado
ou comportamento amplamente desorganizado (s/n)? ")
    then (assert (grupo Psicoticos))))

(defrule grupo-Transtornos_Humor ""
  (not(doenca $?))
=>
  (if(SN "O paciente apresenta humor deprimido, elevado, expansivo ou
irritável (s/n)? ")
    then (assert (grupo Humor))))

(defrule grupo-Transtornos_Ansiedade ""
  (not(doenca $?))
=>
  (if(SN "O paciente apresenta sintomas de ansidade, medo, esquiva ou
maior excitação (s/n)? ")
    then (assert (grupo Ansiedade))))

(defrule grupo-Transtornos_Somatoforme ""
  (not(doenca $?))
=>
  (if(SN "
O paciente apresenta queixas somáticas ou ansiedade irracional
acerca de doença ou aparência (s/n)? ")
    then (assert (grupo Somatoforme))))
```

Uma vez separado o grupo geral de sintomas, especificam-se as regras para checar os sintomas daquele grupo.

No Quadro 9, encontram-se exemplos de regras para o grupo de transtornos devido a uma condição médica geral.

Quadro 9 – Exemplos de regras para transtornos devido a uma condição médica geral

```
(defrule MULTIPLAS_ETIOLOGIAS-ou-COND_medica_GERAL ""
  (not(doenca $?))
  (grupo CMG)
=>
  (if(SN "O paciente possui perturbação da consciência e alteração na
Cognição (s/n)? ")
  then
      (if(SN "Há evidencias de que a perturbacao apresenta mais de uma
etiologia (s/n)? " )
      then (assert(doenca "Delirium devido a múltiplas etiologias"))
      else
        (assert(doenca "Delirium devido a uma condição médica geral"))
      )
  )
)


(defrule TRANSTORNO-PSICOTICO-CMG ""
  (not(doenca $?))
  (grupo CMG)
=>
  (if(SN "Há predominio de delirios ou alucinações proeminentes (s/n)?
")
  then (assert(doenca "Transtorno psicotico devido a uma condição
médica geral"))
  )
)
```

Já no Quadro 10 estão exemplos de regras para transtornos induzidos por substâncias.

Quadro 10 – Exemplos de regras para transtornos induzidos por substância

```
(defrule GRUPO-PERTURBACAO-CONSCIENCIA-SUB ""
  (not(doenca $?))
  (grupo Substancia)
=>
  (if(SN "Há uma perturbacao da consciencia e uma alteracao que
excedem o que eh Hábitualmente
 visto na intoxicacao ou abstinencia e indicam uma atencao clinica
independente (s/n)? ")
  then
     (if(SN "Há evidencias de que a perturbacao tem mais de uma
etiologia (s/n)? ")
     then
       (assert (doenca "Delirium devido a multiplas etiologias"))
     else
       (if(SN "Há inicio de delirium durante abstinencia de uma
substancia (s/n)? ")
       then (assert(doenca "Delirium por abstinencia induzido por
substancia"))
       else (assert(doenca "Delirium por intoxicacao induzido por
substancia"))
       )
```

```
        )
    )
)


(defrule GRUPO-PREJUIZO-PERSISTENTE-SUB ""
  (not(doenca $?))
  (grupo Substancia)
=>
  (if(SN "Há prejuízo persistente de memoria (s/n)? ")
  then
    (if(SN "Há pelo menos um deficit cognitivo adicional (s/n)? ")
    then
      (if(SN "Há evidencias de que a perturbacao tem mais de uma
etiologia (s/n)? ")
      then (assert(doenca "Demencia devido a multiplas etiologias"))
      else   (assert(doenca   "Demencia   persistente   induzida   por
substância"))
      )
    else (assert(doenca "Transtorno Amnéstico persistente induzido por
substância"))
    )
  )
)
```

Assim segue-se a escrita do protótipo, onde estão implementadas todas as regras necessárias para se chegar ao diagnóstico diferencial de todas as doenças mentais contidas no Manual Diagnóstico de Transtornos Mentais.

Por fim, no Quadro 11, há uma função que mostrará ao usuário qual é a doença apresentada pelo paciente:

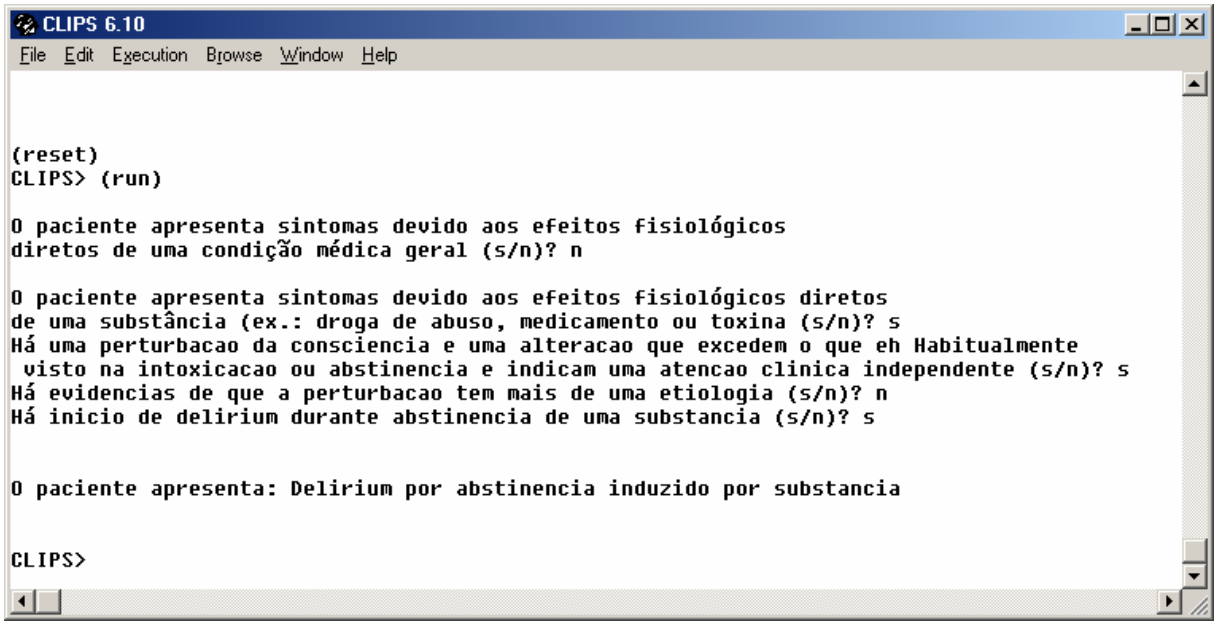
Quadro 11 – Função para imprimir a doença

```
(defrule IMPRIME-DOENCA ""
   (doenca ?item)
=>
(printout t crlf crlf)
(printout t "O paciente apresenta: ")
;(printout t crlf crlf)
(format t "%s%n%n%n" ?item))
```

Quando o programa é inicializado, entram em ação as regras necessárias para separar a qual grupo geral de sintomas que o paciente pertence, as quais utilizam as regras que auxiliam a fazer as perguntas ao usuário.

Os tipos de regras apresentadas no **Quadro 9** e **Quadro 10**, verificam passo a passo quais os sintomas específicos de cada paciente, através das respostas destes. Após respondidas as questões pelo usuário e ter chego a uma conclusão, o sistema coloca em ação a regra que irá imprimir a doença que o paciente possui. Todo esse procedimento pode ser conferido na **Figura 20.**

Figura 20 – Procedimentos após rodar o programa



8 ANÁLISE DA FERRAMENTA CLIPS

Para analisar uma ferramenta, são necessários critérios devidamente justificados. Buscou-se avaliar a ferramenta através dos aspectos já apresentados, relacionados à qualidade de *software* (ISO/IEC 9126), além de critérios considerados relevantes para sistemas especialistas definidos por Souza (2001).

8.1 CARACTERÍSTICAS DA QUALIDADE E MÉTRICA ISO/IEC9126

O critério proposto pela ISO/IEC 9126 consta de seis grandes características principais as quais são divididas em um conjunto de subcaracterísticas, abordado no capítulo seis.

A ferramenta em estudo será avaliada de acordo com essas características. As respostas, a cada uma das perguntas, poderá ser: Sim, Não ou Parcialmente, de acordo com o grau de conformidade da ferramenta em relação à característica a ser avaliada.

Quadro 12 – Características da qualidade software

CARACTERÍSTICA	**SUB CARACTERÍSTICA**	**PERGUNTA**	**CLIPS**
1.Funcionalidade	1.1 Adequação	1.1.1 O software possui todas as funções mencionadas no produto?	S
	1.2 Acurácia	1.2.1 O software é preciso na execução das funções?	S
		1.2.2 O software é preciso nos resultados?	S
	1.3 Interoperabilidade	1.3.1 O software tem restrições quanto ao número de estações trabalhando ao mesmo tempo?	N
	1.4 Conformidade	1.4.1 O software é conciso às leis vigentes?	S
	1.5 Segurança	1.5.1 O software dispõe de segurança de acesso através de senhas?	N
2. Confiabilidade	2.1 Maturidade	2.1.1 O software tem capacidade de continuar executando na ocorrência de erros de execução?	N
	2.2 Tolerância à falhas	2.2.1 O software possui advertência de erros cometidos pelo usuário?	S
		2.2.2 O software controla preenchimento de campos?	N

		2.2.3 O software tem capacidade de verificar se as informações cadastradas estão corretas?	S
		2.2.4 O software tem capacidade de evitar a inclusão de dados existentes?	N
		2.2.5 O software tem boa performance para o processamento com grande volume de dados?	S
	2.3 Recuperabilidade	2.3.1 O software tem capacidade de recuperar dados excluídos?	N
3. Usabilidade	3.1 Inteligibilidade	3.1.1 O software possui autodemonstração?	N
		3.1.2 As telas do software são autoinstrutivas, permitindo ao usuário visualizar com facilidade qual sua função?	N
		3.1.3 As telas do mesmo nível possuem o mesmo padrão?	S
		3.1.4 A ordem de apresentação dos menus segue uma lógica?	S
		3.1.5 O usuário precisa ter conhecimento na área para utilizar o software?	S
		3.1.6 Caso o usuário seja leigo, o software fornece as informações adequadas para sua perfeita utilização?	N
		3.1.7 A teoria que embasa o software é explicada com documentação impressa?	S
		3.1.8 Os jargões técnicos utilizados, são explicados para o usuário?	N
	3.2 Apreensibilidade	3.2.1 O software possui manual de operação?	S
		3.2.2 Todas as funções do software estão explicadas no manual?	S
		3.2.3 O manual apresenta todos os erros que podem ocorrer no software?	N
		3.2.4 A linguagem utilizada na mensagem de ajuda é	N

		facilmente entendida pelo usuário?	
		3.2.5 O software mostra como o usuário deve navegar pelo arquivo de ajuda?	N
	3.3 Operacionalidade	3.3.1 Os comandos do software estão de acordo com os padrões existentes (F1, DEL, ESC, ENTER)?	N
		3.3.2 É possível prever o que existe dentro de cada opção do menu?	P
		3.3.3 Existe padronização de teclas de função para todo o software?	S
4. Eficiência	4.1 Comportamento em Relação ao Tempo	4.1.1 O tempo necessário para inicializar o software é satisfatório?	S
		4.1.2 O tempo de resposta é adequado em relação ao volume de dados envolvidos?	S
5. Manutenibilidade	5.1 Analisabilidade	5.1.1 O software contém funções com objetivos específicos?	S
		5.1.2 O software possui decisões comentadas (sistema de justificação)?	N
		5.1.3 O nome de todas as variáveis do software são exclusivos?	S
		5.1.4 Todas as variáveis são inicializadas antes do uso?	N
6. Portabilidade	6.1 Adaptabilidade	6.1.1 O software pode ser facilmente modificado para atender às necessidades do usuário?	N
	6.2 Instalação	6.2.1 O software possui um programa de instalação?	N
		6.2.2 O software possui *help* de instalação?	N
	6.3 Substituição	6.3.1 O software tem capacidade de ser substituído por novas versões e continuar utilizando a mesma base de dados?	S

8.2 ASPECTOS RELEVANTES PARA SISTEMAS ESPECIALISTAS

Já não é de hoje que existem disponíveis várias ferramentas para construção de sistemas especialistas. Isto faz com que seja dificultosa a decisão do projetista do sistema. Outro fator que dificulta a escolha, é a falta de informações que sejam realmente objetivas sobre as ferramentas existentes (Souza, 2001).

Neste capítulo, serão tratadas algumas características relevantes encontradas durante o estudo da linguagem CLIPS. Essas características são: Interface com o usuário, interface de desenvolvimento e interface com o sistema operacional, motor de inferência e representação do conhecimento.

8.2.1 INTERFACE COM O USUÁRIO

Em qualquer tipo de *software*, a interface com o usuário final é fundamental para o seu sucesso. Em particular, um sistema especialista, além de apresentar uma interface ergonomicamente bem projetada, deve ainda levar em conta o grau de familiarização do usuário com o domínio de trabalho do sistema e com os computadores em geral (Souza, 2001).

A atividade principal, e talvez até única das ferramentas para sistemas especialistas é receber informações e a partir delas gerar e/ou adicionar mais informações aos dados inicialmente fornecidos, para que estes sejam utilizados resolução de problemas.

A equipe de projeto deve observar se as informações recebidas, após o uso da ferramenta, são suficientes para a necessidade que apresentam. Caso o resultado não seja satisfatório, será necessário complementar esta informação com um estudo ou, até mesmo, com a utilização de uma outra ferramenta. Neste caso, a ferramenta em análise, deve ser considerada deficiente e deve ser analisada para verificar se existe algum problema em relação a interface com o usuário. Caso não seja adequado a utilização de apenas uma ferramenta, deve-se procurar o uso do menor número de ferramentas.

Algumas técnicas tornam a interface com o usuário mais amigável, por exemplo, o uso de janelas, menus, gráficos de alta resolução, animação, cores, etc. De todo modo, as telas a serem apresentadas ao usuário devem ser de fácil compreensão, e as explicações necessárias devem ser claras e diretas. Mais algumas características interessantes para interfaces com o

usuário são: (i) disponibilidade de diversos tipos de interfaces, adaptadas ao tipo de usuário (iniciante, especialista); (ii) possibilidade de interromper a execução do sistema em um determinado ponto e poder retomá-la sem necessidade de reprocessamento; (iii) mensagens de erro claras e informativas; (iv) possibilidade de alterar certas entradas ao sistema e comparar os resultados obtidos; (v) capacidade de capturar e armazenar telas de execução.

A ferramenta CLIPS não apresenta recursos interessantes para a construção de uma interface muito amigável. Os sistemas gerados em CLIPS puro possuem uma interface fraca e o acesso é difícil para usuários com pouco conhecimento. Porém existem fortes recursos para integrar o CLIPS com outras linguagens, onde se pode, então, criar interfaces melhoradas para as bases feitas em CLIPS (Giarratano[1], 1998).

Outro recurso é o wxCLIPS que, como já foi exposto no item 5.6, é capaz de criar interfaces visuais para os sistemas feitos em CLIPS.

8.2.2 INTERFACE DE DESENVOLVIMENTO

Ao buscar uma ferramenta adequada, um projetista procura algo que realiza sua atividade com a menor dificuldade possível. À medida que uma ferramenta deixa de facilitar a sua aprendizagem e uso, o emprego da mesma poderá não trazer resultados desejados. Fatores como este, poderão fazer com que o projetista evite a aplicação da ferramenta.

O critério usabilidade, já exposto anteriormente, pode verificar qual o nível de dedicação que o projetista precisará dispensar nos processos de aprendizagem e de uso. Este critério está diretamente relacionado à maneira pela qual a ferramenta permite um uso mais fácil ou não, avaliando tanto o processo de aprendizagem como de utilização.

Por ser muito subjetivo, antes da avaliação por este critério, deve-se investigar o nível de dificuldade para o aprendizado, quais atividades de preparação ao uso são necessárias e a maneira como é efetuado o processo de utilização. Para realizar esta investigação, deve-se buscar informações sobre a necessidade de um treinamento prévio para a utilização, a leitura de manuais, preparativos ao uso, entre outros. Esta investigação deve ser repetida para cada uma das ferramentas. Porém, deve-se observar que, para determinadas aplicações, algumas ferramentas vão necessitar de uma mão-de-obra maior que outras, enquanto que, para outras aplicações, as mesmas ferramentas não terão o mesmo problema (Souza, 2001).

Correção de erros, manutenção de uma lista de comandos da seção, índice cruzado de símbolos e regras, visualização gráfica dos encadeamentos de regras utilizadas na solução, editor de regras, facilidades de explicação e uma boa documentação são outras características positivas para a interface de desenvolvimento.

Aspectos considerados críticos: facilidade para explicação, documentação, facilidade para customizar explicações e prototipagem rápida.

A ferramenta CLIPS necessita de um bom conhecimento do projetista. O editor é um ambiente separado ao compilador, o que torna a utilização um pouco incômoda. Além disso, se já estiver aberto um sistema no CLIPS e for solicitada a abertura de outro sistema, o CLIPS, além de não retirar o primeiro sistema, mistura as regras dos dois sistemas, permanecendo ambos abertos. Para que isso não aconteça, o projetista deve, antes de abrir o segundo sistema, utilizar o comando *(clear)*.

Apesar da ferramenta apresentar algumas características um pouco frustrantes à primeira vista (por exemplo, cada vez que se deseja rodar um programa, não basta digitar o comando *(run)*; deve-se digitar *(reset)* e depois *(run)*), encontra-se facilmente na Internet toda a documentação necessária para a operação.

8.2.3 INTERFACE COM O SISTEMA OPERACIONAL

A facilidade de comunicação com aplicações convencionais (como bancos de dados, planilhas e sistemas de rede) é considerada uma importante característica em um sistema especialista. O fato de essa facilidade ter sido relegada a um segundo plano nos primeiros sistemas especialistas representou uma grande dificuldade para sua disseminação em ambiente comercial. As ferramentas precisam fazer alguma coisa a mais para justificarem seu uso. Elas precisam facilitar a integração dos sistemas especialistas com outros tipos de programas. Os sistemas especialistas não podem operar no vácuo, como também não podem os humanos. Eles precisam acessar bancos de dados das corporações, e esse acesso precisa ser controlado como em outros sistemas. Eles, em geral, estão embutidos em programas aplicativos maiores, que usam basicamente técnicas de programação convencional. Então, uma das características importantes que uma ferramenta precisa ter é uma interface entre o sistema especialista, e um ambiente de programação maior e provavelmente mais convencional (Souza, 2001).

Outra característica ligada ao sistema operacional é o grau de segurança oferecido por um sistema especialista. É possível que um sistema especialista envolva informações confidenciais ou únicas que não devem ser acessadas e modificadas por qualquer usuário.

Aspectos considerados críticos: facilidade para integração com sistemas já existentes.

Como já visto, a ferramenta CLIPS foi desenhada para uma total integração com linguagens como C e Ada.

O CLIPS não apresenta, diretamente, características relacionadas à segurança.

8.2.4 MOTOR DE INFERÊNCIA

As principais características do motor de inferência disponível nos sistemas especialistas dizem respeito às seguintes funcionalidades: método de raciocínio, estratégia de busca, resolução de conflito e representação de incerteza. Dentre todos estes aspectos, foi considerada crítica a existência dos métodos de raciocínio.

8.2.4.1 MÉTODO DE RACIOCÍNIO

Em geral, os sistemas especialistas adotam apenas um modo de raciocínio; no entanto, existem alguns que permitem ambos os modos, mas de maneira independente, e ainda outros que permitem um encadeamento misto, onde os encadeamentos progressivo e regressivo se alternam de acordo com o desenvolvimento da solução do problema e com a disponibilidade de dados.

Uma característica importante do modo de raciocínio se refere à monotonicidade ou não do método de inferência. Sistemas monotônicos não permitem a revisão de fatos, isto é, uma vez um fato declarado verdadeiro, ele não pode mais tornar-se falso. Sistemas não monotônicos, por outro lado, permitem a alteração dinâmica dos fatos. O preço desta capacidade é a necessidade de um mecanismo de revisão de crenças, pois uma vez que um fato, antes verdadeiro, torna-se falso, todas as conclusões baseadas neste fato também devem tornar-se falsas (Bittencourt, 1998) *apud* (Souza, 2001).

Uma vez definido o tipo de encadeamento, o motor de inferência necessita ainda de uma estratégia de busca para guiar a pesquisa na memória de trabalho e na base de regras e ao terminar o processo de busca, o motor de inferência dispõe de um conjunto de regras que

satisfazem à situação atual do problema, o chamado conjunto de conflito, porém esses aspectos não estão referenciados nas *ferramentas*.

O CLIPS utiliza-se somente do encadeamento para frente, onde a busca inicia-se pela primeira regra verificando suas premissas. Se estas forem satisfeitas, a conclusão da regra é executada. O mesmo procedimento é feito com a segunda regra, com a diferença que, entre as premissas, pode estar a conclusão da primeira regra e assim por diante.

8.2.4.2 REPRESENTAÇÃO DE INCERTEZA

Diversos métodos foram propostos para tratar este problema, por exemplo, método Bayesiano, fatores de certeza, teoria de Dempster-Shafer, teoria dos conjuntos nebulosos, teoria de probabilidades subjetivas e teoria de possibilidades (Bittencourt, 1998) *apud* (Souza, 2001).

De maneira geral, estes métodos atribuem aos fatos e regras uma medida numérica que represente de alguma forma a "confiança" do especialista. Os métodos utilizados não são necessariamente coerentes uns com os outros e cada método adapta-se melhor a determinados tipos de problemas. Diversos sistemas especialistas dispõem de mais de um método de tratamento de incerteza, deixando ao usuário a escolha do mais adequado ao seu problema. Uma característica freqüente desses métodos é a existência de um limite mínimo para a medida de incerteza, abaixo do qual o fato ou regra é desconsiderado. Este limite pode, em geral, ser fixado pelo usuário.

O CLIPS possui as palavras chaves *declare salience* para declarar a prioridade (grau de confiança) das regras. O grau de confiança é declarado utilizando um valor numérico que pode estar entre -10000 e 10000. Se uma regra não possui um grau de confiança explícito pelo desenvolvedor, o CLIPS assume grau de confiança zero. O grau zero é o meio termo entre o mais baixo e mais alto valor de confiança. O valor zero para o grau de confiança não significa que a regra não possui grau, mas sim, que ela possui um nível de prioridade intermediária.

No Quadro 13 pode-se visualizar como é feita a declaração de grau de confiança.

Quadro 13 – Exemplo de declaração de grau de confiança

```
(defrule teste-1
(declare (salience 99))
(dispara teste-1)
```

```
        =>
        (printout t "Regra teste-1 disparando." crlf))

        (defrule teste-2
        (declare (salience (10))
        (dispara teste-2)
        =>
        (printout t "Regra teste-2 disparando." crlf))
```

8.2.5 REPRESENTAÇÃO DO CONHECIMENTO

Em geral, os sistemas especialistas se limitam a oferecer um único tipo de representação de conhecimento. Alguns sistemas dispõem de mais de um formalismo, os quais, no entanto, devem ser utilizados de maneira isolada. Alguns poucos sistemas especialistas possuem os chamados sistemas híbridos de representação de conhecimento que, além de possuir diversos formalismos de representação, dispõem também de algoritmos de acesso que integram os conhecimentos representados nos diversos formalismos para permitir sua utilização de maneira integrada (Bittencourt, 1998) *apud* (Souza, 2001).

O CLIPS permite a representação do conhecimento através de regras, o chamado conhecimento heurístico, e a representação através do conhecimento procedural, onde se utiliza a programação orientada a objetos.

8.2.6 TESTES EFETUADOS E RESULTADOS

Para que pudesse proceder a análise da ferramenta, foi desenvolvido, conforme especificado no capítulo 7, o protótipo de um sistema especialista diagnóstico de transtornos mentais.

A implementação do protótipo através do CLIPS chegou a um bom resultado. A ferramenta demonstrou-se bastante apta para a construção de sistemas especialistas baseados em regras de produção.

Para efetuar os testes com o protótipo desenvolvido, seguiu-se, passo a passo, cada uma das condições de todas as árvores de decisão. O protótipo foi capaz de chegar ao diagnóstico correto para todas as doenças.

9 CONCLUSÃO

O desenvolvimento deste projeto permitiu uma análise das características e particularidades da ferramenta para sistemas especialistas CLIPS e o desenvolvimento de um protótipo diagnóstico de transtornos mentais para avaliar a ferramenta. A ferramenta demonstrou-se útil e prática na construção de sistemas especialistas baseados em regras. Contudo, o projetista precisa (i) ter um bom conhecimento relativo à linguagem e (ii) adaptar-se à sua lógica. Também apresentou eficiência em resolver problemas reais em um curto espaço de tempo e utilizando poucos recursos.

A deficiência da ferramenta encontrou-se na interface com o usuário. O CLIPS não possui recursos para uma boa interface, mas pôde-se perceber, conforme literaturas, que o CLIPS pode comunicar-se com outras linguagens onde, nestas, pode-se criar uma interface mais amigável. Além disso, existe o wxCLIPS, uma ferramenta com a finalidade de criar interfaces para o CLIPS.

Os obstáculos encontrados no desenvolver deste trabalho surgiram, primeiramente, na relativa escassez de literaturas sobre a ferramenta CLIPS. Houve também, dificuldades quanto à lógica do CLIPS no que diz respeito à construção de certas estruturas, como por exemplo algumas funções específicas.

O objetivo do protótipo proposto foi alcançado. O mesmo consegue diagnosticar todas as doenças – cerca de 80 – apresentadas no Manual Diagnóstico e Estatístico de Transtornos Mentais.

Segundo o especialista de conhecimento da área psicológica, o protótipo consegue auxiliar os profissionais desta área que têm dificuldades ao interpretar o livro Manual Diagnóstico e Estatístico de Transtornos Mentais (DSM), além de agilizar a tomada de decisão quanto ao diagnóstico do paciente. O ponto negativo do sistema foi a interface que, para o especialista, é confusa e pouco estimulante. O sistema, de modo geral, pode tornar mais eficaz e rápido o atendimento a pacientes portadores de transtornos mentais.

9.1 EXTENSÕES

Como extensão para futuros projetos, sugere-se implementar o mesmo sistema diagnóstico utilizando a ferramenta wxCLIPS para criar uma interface mais amigável.

Também sugere-se implementar este sistema em outras ferramentas, tais como M.4VB e Jess, as quais foram citadas no capítulo 4, ou ainda integrar a ferramenta CLIPS com outras linguagens como C.

REFERÊNCIAS BIBLIOGRÁFICAS

BITTENCOURT, Guilherme. **Inteligência Artificial:** ferramentas e teorias. Florianópolis: Editora da UFSC, 1998. 362 p.

CARVALHO, André Ponce de Leon F. de. **Algoritmos Genéticos**. São Carlos, jul. 1997. Disponível em: <http://www.icmsc.sc.usp.br/~andre/gene1.html>. Acesso em: 23 out. 2001.

COSTA, Marcello Thiry Comicholi da. **Uma Arquitetura Baseada em Agentes para Suporte ao Ensino à Distância**. Florianópolis, abr. 1999. Disponível em: <http://www.eps.ufsc.br/teses99/thiry/cap3.htm>. Acesso em: 20 out. 2001.

COSTA, Raimundo Machado. **Qualidade de produtos de software**. Rio de Janeiro 2001. Disponível em: <http://raimac.netdados.com.br/qualidade.htm>. Acesso em : 25 out. 2001.

CRIPPA, Maurício. **Sistemas especialistas**: a engenharia do conhecimento aplicada às organizações. Florianópolis, 1999. Disponível em: <http://n27.xudesc.br/demo/trabalhos/alunos/mc/index.html>. Acesso em: 02 set. 2001.

FÁVERO, Alexandre José. **Sistemas Especialistas**: Tutorial. Maringá, [2000?]. Disponível em: <http://din.uem.br/ia/especialistas>. Acesso em: 08 set. 2001.

FERNANDES, A.M.R. **Sistema especialista difuso aplicado ao processo de análise química qualitativa de amostras de minerais**. Florianópolis, 1996. 121 f. Dissertação (Mestrado no Curso de Pós-Graduação em Ciências da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

FERREIRA, Aurélio Buarque de Holanda. **Dicionário Aurélio básico da língua portuguesa.** Rio de Janeiro: Nova Fronteira, 1988.

FRIEDMAN-HILL, Ernest J. **Jess**: the rule engine for the java plataform. Livemore, nov. 2001. Disponível em: <http://herzberg.ca.sandia.gov/jess/>. Acesso em: 13 nov. 2001.

GIARRATANO[1], Joseph C. **Clips user's guide**. 1998.

GIARRATANO[2], Joseph C. **Clips:** Reference Manual Volume I – Basic Programing Guide. 1998.

I.A., Grupo de. **Inteligência artificial.** Belo Horizonte, 2000. Disponível em: <http://www.grupoia.cjb.net>. Acesso em: 15 set. 2001.

HEINZLE, Roberto. **Protótipo de uma ferramenta para criação de sistemas especialistas baseados em regras de produção.** 1995. 145 f. Tese (Mestrado em Engenharia de Produção), Universidade Federal de Santa Catarina, Florianópolis.

JORGE, Miguel R. **DSM-IV:** manual diagnóstico e estatístico de transtornos mentais. 4 ed. Porto Alegre: Artes Médicas, 1995.

KELLER, Robert. **Tecnologia de sistemas especialistas**: desenvolvimento e aplicação. São Paulo: Makron Books, 1991.

MANCHINI, Daniela Patrícia; PAPPA, Gisele Lobo. **Sistemas Especialistas.** Maringá, [2000?]. Disponível em: <http://www.din.uem.br/~ia/intelige/especialistas/especialistas/>. Acesso em: 10 set. 2001.

MINSKY, Marvin. **Semantic information processing.** Cambridge: Mit Press, 1968.

NOGUEIRA, José Helano Matos. **Expert SINTA**: help. Fortaleza: Grupo SINTA, 1995.

OLIVEIRA , Fabio Abreu Dias de. **Arquiteturas Especiais de Computadores**. Porto Alegre, [1999?]. Disponível em: <http://www.inf.ufrgs.br/procpar/disc/cmp135/trabs/992/Parser>. Acesso em: 15 out. 2001.

ORCHARD, Bob. **FuzzyCLIPS**. Canadá, [2000?]. Disponível em: < http://ai.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPSIndex.html>. Acesso em: 10 set. 2001.

PARPINELLI, Rafael Stubs; WATANABE, Wagner Tatsuya. **CLIPS**. Maringá, 2000. Disponível em: <http://www.din.uem.br>. Acesso em: 25 ago. 2001.

PEREIRA, C. G. **Análise do crédito bancário:** um sistema especialista com técnicas difusas para os limites da agência. Florianópolis, 1995. 135 f. Dissertação (Mestrado em Engenharia de Produção e Sistemas), Universidade Federal de Santa Catarina, Florianópolis.

RABUSKE, Renato Antonio. **Inteligência artificial**.. Florianópolis: Editora da UFSC, 1995.

RAMOS, Ronaldo Fernandes. **Sistemas Especialistas:** uma abordagem baseada em objetos com prototipagem de um selecionador de processo de soldagem. Florianópolis, fev. 1995. Disponível em: <http://www.eps.ufsc.br/disserta/ramos>. Acesso em: 18 out. 2001.

REGINALDO. **Raciocínio Baseado em Casos**. Itajaí, [2000?]. Disponível em: <http://cejurps.univali.br/reginaldo/>. Acesso em: 23 set. 2001.

RICH, Elaine; KNIGHT, K. **Inteligência artificial.** São Paulo: Makron Books, 1993.

SOUZA, Aline R. **Comparativo de ferramentas para sistemas especialistas**. 2001. 99 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOUZA, Marilda M. **Sistema de avaliação de aptidão física relacionada à saúde**. 1998. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Educação Superior de Ciências Tecnológicas, da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí.

STORCH, Mirian Mirdes. **Proposta de avaliação da qualidade de produtos de software utilizando a norma ISO/IEC 9126**. 2000. 82 f. Trabalho de conclusão de curso (Bacharelado em Ciências da Computação), Universidade Regional de Blumenau, Blumenau.

TATIBANA, Cassia Yuri; KAETSU, Deisi Yuki. **Redes Neurais**. Maringá, [2000?]. Disponível em: <http://www.din.uem.br/ia/neurais/>. Acesso em: 23 out. 2001.

UFSC. Universidade Federal de Santa Catarina. **Expert Systems**: in production engineering – EPS3416. Florianópolis, 2000. Disponível em: <http://www.eps.ufsc.br/~oscar/exp_sys/ing/>. Acesso em: 30 out. 2001.

ULBRICHT, Vania Ribas; WAZLAWICK, Raul. **Representação do conhecimento de geometria descritiva através de redes semânticas**. Florianópolis, [2000?]. Disponível em: <http://www.inf.ufsc.br/~raul/public1/graph2.htm>. Acesso em: 18 out. 2001.

WINTERS. 1997. **Sistemas Especialistas**. 1997. Disponível em: <http://www.istoline.com/m4.html>. Acesso em: 20 out. 2001.