UNIVERSIDADE FEDERAL DA PARAIBA

CENTRO DE CIENCIAS E TECNOLOGIA

CURSO DE MESTRADO EM ENGENHARIA ELÉTRICA

MODELAGEM E IMPLEMENTAÇAO DE UMA BASE DE CONHECIMENTO PARA PROJETOS DE FILTROS DIGITAIS

ANTONIO MENDES DA SILVA FILHO

CAMPINA GRANDE - PB

AGOSTO - 1991

ANTONIO MENDES DA SILVA FILHO

MODELAGEM E IMPLEMENTAÇÃO DE UMA BASE DE CONHECIMENTO PARA PROJETOS DE FILTROS DIGITAIS

Dissertação apresentada ao Curso de Mestrado em Engenharia Elétrica da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de mestre.

AREA DE CONCENTRAÇÃO: PROCESSAMENTO DE SINAIS

Prof. JOÃO MARQUES DE CARVALHO, Ph.D.
Orientador

CAMPINA GRANDE - PB
AGOSTO - 1991

S586m Silva Filho, Antonio Mendes da
Modelagem e implementacao de uma base de conhecimento para projetos de filtros digitais / Antonio Mendes da Silva Filho. - Campina Grande, 1991.
126 f. : il.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Processamento Digital de Sinais 2. Computacao - 3. Engenharia Eletrica 4. Dissertacao I. Carvalho, Joao Marques de, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.383.3(043)

MODELAGEM E IMPLEMENTAÇÃO DE UMA BASE DE CONHECIMENTO
PARA PROJETO DE FILTROS DIGITAIS

ANTONIO MENDES DA SILVA FILHO

DISSERTAÇÃO APROVADA EM 30.08.91

JOAO MARQUES DE CARVALHO, Ph.D., UFPB
Orientador

MISAEL ELIAS DE MORAIS, Dr.Ing., UFPB
Componente da Banca

BENEDITO GUIMARÃES AGUIAR NETO, Dr.-Ing., UFPB
Componente da Banca

CAMPINA GRANDE - PB
AGOSTO - 1991

AGRADECIMENTOS

Agradeço àquelas pessoas que direta ou indiretamente contribuíram para a realização deste trabalho.

Agradeço em especial:

Ao Prof. JOÃO MARQUES DE CARVALHO, pela orientação e ensinamentos;

Aos Professores MAURO CAVALCANTI PEQUENO e BENEDITO GUIMARAES AGUIAR NETO pelos ensinamentos e apoio;

Ao Prof. ANGELO PERKUSICH pela contribuição e ensinamentos;

A amiga MARIA LIGIA BARBOSA PERKUSICH pelo apoio e contribuição.

Aos meus pais Antonio e Eunide, aos meus filhos Bruno e Sthéfano, a minha irmã Conceição e a minha companheira Viviane.

RESUMO

Este trabalho objetiva fazer a modelagem e implementação de uma base de conhecimento. Esta base de conhecimento aliada a um mecanismo de inferência servirá de suporte a um sistema especialista para projetos de filtros digitais.

A base de conhecimento reunirá informações sobre as diversas áreas de aplicações de processamento digital de sinais e sobre os algoritmos empregados para a síntese de filtros digitais.

O mecanismo de inferência utiliza estas informações da base de conhecimento juntamente com as especificações fornecidas pelo usuário para selecionar o algoritmo adequado para uma dada situação.

CONTEUDO

CAPITULO 1

INTRODUÇAO 01
1.1 - Introdução 01
1.2 - Objetivo do Trabalho 03
1.3 - Apresentação 04

CAPITULO 2

FILTROS DIGITAIS E PROCESSAMENTO DIGITAL DE SINAIS 06
2.1 - Introdução 06
2.2 - Aspectos Gerais 06
2.2.1 - Representação do Sinal Digital 07
2.2.2 - Caracterização de Filtros Digitais 08
2.2.3 - Análise Comparativa Quanto ao Uso de Filtros Digitais e Analógicos 13
2.2.4 - Etapas de Projeto de Filtros Digitais 15
2.3 - Técnicas para Projeto de Filtros Digitais 15
2.3.1 - Técnicas para Projeto de Filtros IIR 16
2.3.2 - Técnicas para Projeto de Filtros FIR 17
2.4 - Areas de Aplicações de PDS 18
2.4.1 - Processamento de Voz 18
2.4.2 - Audio 20
2.4.3 - Comunicações 21
2.4.4 - Biomédica 22
2.4.5 - Outras 23
2.5 - Resumo 24

CAPITULO 3

AQUISIÇAO E REPRESENTAÇAO DE CONHECIMENTO 25

3.1 - Conceitos Básicos 26
3.2 - Aquisição de Conhecimento 26
3.2.1 - Aspectos Relacionados à Aquisição de Conhecimento 26
3.2.2 - Métodos de Aquisição de Conhecimento 27
3.3 - Representação de Conhecimento 29
3.3.1 - Aspectos Relacionados à Representação de Conhecimento 29
3.3.2 - Tipos de Representação de Conhecimento 30
3.4 - Resumo 32

CAPITULO 4

AQUISIÇAO E REPRESENTAÇAO DE CONHECIMENTO: MODELAGEM E IMPLEMENTAÇAO 33

4.1 - Metodologia para Modelagem da Base de Conhecimento 33
4.2 - Dificuldades Encontradas na Aquisição de Conhecimento 40
4.3 - Considerações Sobre o Uso de Regras de Produção 42
4.4 - Apresentação da Arvore de Decisão 43
4.5 - Resumo 53

CAPITULO 5

UNIDADE CENTRAL - ESTRUTURA PARA SELEÇAO DE ALGORITMOS PARA PROJETOS DE FILTROS DIGITAIS 54

5.1 - Introdução 54

5.2 - Arquitetura da Unidade Central 54

5.3 - Interface para Voluntariamento 55

5.4 - Unidade Central 58

5.4.1 - Base de Conhecimento 58

5.4.2 - Memória de Trabalho 60

5.4.3 - Módulo de Inferência 60

5.5 - Acoplamento da Unidade Central a um Sistema Especialista 63

5.6 - Resumo 64

CAPITULO 6

CONCLUSOES 65

REFERENCIAS BIBLIOGRAFICAS 67

APENDICE 73

APENDICE A1 73

APENDICE A2 78

APENDICE A3 80

APENDICE A4 87

APENDICE A5 89

APENDICE A6 93

APENDICE A7 95

APENDICE A8 101

APENDICE B1 103
APENDICE B2 120

LISTA DE FIGURAS

Fig. 2.1.a. Filtro passa-baixa 09

Fig. 2.1.b. Filtro passa-alta 09

Fig. 2.2.a. Filtro passa-faixa 09

Fig. 2.2.b. Filtro rejeita-faixa 09

Fig. 3.1. Consulta do Engº de Conhecimento com o Especialista 27

Fig. 3.2. Resolução de conflito 31

Fig. 3.3. Exemplo de Rede Semântica 31

Fig. 3.4. Exemplo de conceito de frame 32

Fig. 4.1. Processo de modelagem da base de conhecimento ... 34

Fig. 4.2. Arvore de decisão 47-53

Fig. 5.1. Arquitetura da unidade central 54

Fig. 5.2. Diagrama Nassi Schneiderman do processo de voluntariamento e diagnóstico 55

Fig. 5.3. Questões do processo de voluntariamento 56-57

Fig. 5.4. Base de conhecimento celular 58

Fig. 5.5. Conteúdo da memória de trabalho 60

Fig. 5.6. Partes componentes do módulo de inferência 61

Fig. 5.7. Diagrama Nassi Schneiderman do mecanismo de inferência 63

Fig. 5.8. Controle da unidade central 64

Capítulo 1

Introdução

1.1. Introdução

O campo de processamento digital de sinais tem realizado um enorme progresso nos últimos vinte anos, tanto a nível teórico quanto prático. Todo o desenvolvimento deste campo se fez baseado na teoria de sistemas discretos no tempo, lineares e invariantes no tempo. Aliado a esse avanço teórico, houve também o desenvolvimento da tecnologia de circuitos digitais. A disponibilidade comercial de processadores de alta velocidade com o uso direcionado para processamento digital de sinais (processadores dedicados) associada ao seu alto grau de confiabilidade, reprodutibilidade, compactação e eficiência tem tornado os sistemas digitais mais atrativos em detrimento dos analógicos. Isto também se deve face ao objetivo de melhorar o desempenho dos sistemas.

O avanço teórico e prático no campo de processamento digital de sinais tem motivado o uso desta técnica numa variedade de aplicaçës. Dentre muitas aplicações, tem-se: áudio, processamento de voz, comunicações e processamento de imagens [CAST79].

De uma forma geral, processamento digital de sinais nada mais é do que filtragem digital. Desta forma, na filtragem digital, tem-se duas categorias de filtros digitais, os quais são classificados quanto à sua resposta ao impulso, onde se tem filtros com resposta ao impulso finita (FIR - Finite Impulse Response) e os filtros com resposta ao impulso infinita (IIR -

Infinite Impulse Response) [RABI75].

Embora uma grande quantidade das propriedades dos diferentes tipos de filtro digital sejam conhecidas, pouco tem sido feito para relacionar as várias técnicas existentes para projetos de filtros ao desempenho dos mesmos. Portanto, o projetista deve aprender detalhes dos diversos procedimentos de projeto antes de ser capaz de tomar uma decisão na escolha de uma estrutura de filtro adequada à sua aplicação específica.

A interação das áreas de inteligência artificial e processamento digital de sinais tem se evidenciado por vários fatores, destacando-se o interesse de melhorar a eficiência e consequentemente a desempenho dos sistemas digitais [BELL86]. Também, o campo de inteligência artificial tem explorado a utilização de sistemas especialistas ou dos sistemas inteligentes baseados em conhecimento em aplicações nas diversas áreas (domínios) [RYCH88].

Os sistemas especialistas são baseados numa extensa quantidade de conhecimentos sobre uma área específica, onde em geral, tem-se o conhecimento representado na forma de regras, às quais permitem a orientação ou tomada de decisões a partir de um conjunto de dados ou premissas.

Dentro dessa visão, em uma linha de pesquisa conjunta envolvendo o Laboratório de Automação e Processamento de Sinais - LAPS/DEE e o Grupo de Matemática Computacional - GMC/DSC da UFPb, têm sido desenvolvidos vários sistemas para atuar como ferramentas de análise/síntese de sinais e sistemas digitais, como o EXBITAN [PEQU88] e o FFTEX [PEQU89]. Outro sistema que se encontra em fase de desenvolvimento é o SIPREX - um sistema

especialista para projetos de filtros digitais, cujo objetivo maior é minimizar o conjunto de requisitos para um usuário-engenheiro, atuando como uma ferramenta capaz de efetuar a escolha de um algoritmo que forneça o melhor projeto de um filtro digital atendendo às especificações do usuário [CARV91]. Além de identificar o melhor algoritmo, o sistema SIPREX realiza o projeto do filtro digital, fornecendo um conjunto de coeficientes para sua implementação e também um conjunto de curvas das respostas em freqüência da magnitude, fase e atraso de grupo (group delay).

1.2. Objetivo do Trabalho

O objetivo deste trabalho é a modelagem e a implementação de uma base de conhecimento que aliada a um mecanismo de inferência sirvam de suporte a um sistema especialista a ser utilizado no projeto de filtros digitais. A base de conhecimento deverá reunir informações sobre diversas áreas de aplicações de processamento digital de sinais e portanto de filtro digitais, bem como sobre os algoritmos a serem empregados na síntese destes filtros. O mecanismo de inferência utiliza estas informações juntamente com as especificações fornecidas pelo usuário para selecionar o algoritmo mais adequado para uma dada situação.

A informação contida na base de conhecimento é obtida através de estudos da literatura existente sobre os assuntos mencionados acima, entrevistas com especialistas e testes com os programas que implementam os algoritmos utilizados para síntese de filtros digitais. Esta informação poderá ser modificada durante a fase inicial de aprendizagem do sistema especialista.

Para efeito de testes, foi implementada também uma interface homem-máquina que permite a iteração com possíveis usuários e com especialistas, propiciando assim condições para refinamento da informção contida na base de conhecimento. Esta interface, por sua vez, será abandonada quando da incorporação da base de conhecimento e do mecanismo de inferência ao sistema especialista, o qual deverá possuir sua própria interface.

Assim, o objetivo geral deste trabalho se constitui no desenvolvimento de uma ferramenta que auxilie na escolha de um algoritmo para projeto de filtros digitais, a qual reúne uma base de conhecimento (dividida em bancos modulares) contendo informações pertinentes às áreas de aplicação de processamento digital de sinais e algoritmos de projetos de filtros digitais , uma interface para voluntariamento com o usuário e a utilização de um mecanismo de inferência [PERK90], desenvolvido nesta universidade, adaptando-o para a proposta deste trabalho.

1.3. Apresentação

Na seqüência, o capítulo 2 apresenta um estudo geral sobre os filtros digitais e de algumas áreas de aplicação de processamento digital de sinais.

O capítulo 3 apresenta os processos de aquisição e representação de conhecimento, seus conceitos básicos e as técnicas existentes.

O capítulo 4 aborda os procedimentos de aquisição e representação de conhecimentos utilizados quando da modelagem da base de conhecimento. Também, é apresentado o conjunto de

algoritmos usados como soluções no sistema SIPREX.

O capítulo 5, relacionado com o capítulo 4, apresenta uma Unidade de Voluntariamento e Diagnóstico, aqui chamada de Unidade Central. Também, no capítulo 5 é feita uma abordagem sobre o acoplamento da Unidade Central ao SIPREX (Sistema Especialista para Projetos de Filtros Digitais).

O cap. 6 apresenta as conclusões deste trabalho e as possibilidades de extensões do mesmo.

Capítulo 2

Filtros Digitais e Processamento Digital de Sinais

2.1. Introdução

Quase todos os campos da ciência e engenharia, tais como acústica, física, telecomunicações, comunicações de dados e sistemas de controle manipulam com sinais. Em muitas aplicações, é desejável que o espectro de freqüências de um sinal seja modificado ou manipulado de acordo com uma especificação desejada. O processo pode envolver uma atenuação de um intervalo de componentes de freqüência e rejeitar ou isolar uma faixa de freqüências. Qualquer estrutura que exiba essa propriedade de seletividade em freqüência é chamada de filtro.

Como abordado no capítulo 1, um dos objetivos deste trabalho é o de estruturar uma base de conhecimento, composta de conhecimentos sobre filtros digitais e áreas de aplicação de processamento digital de sinais. Como introdução, inicialmente, os sinais e filtros digitais são apresentados, bem como é feita uma comparação deles com os filtros analógicos. Isto é feito com o objetivo de apresentar os conceitos básicos pertinentes ao estudo sobre filtros digitais.

2.2. Aspectos Gerais

Nesta seção, é feita uma caracterização do filtros digitais e uma comparação com os filtros analógicos.

2.2.1. Representação do Sinal Digital

Um sinal pode ser definido como uma função que transporta informação, normalmente, sobre o estado ou comportamento de um sistema ou fenômeno físico.

Os sinais podem ser representados de várias formas. Por exemplo, ele pode se apresentar variando no tempo ou no espaço. Os sinais são representados matematicamente como funções de uma ou mais variáveis independentes. Como exemplo de variáveis independentes mais comuns, tem-se o tempo (sinais unidimensionais) e as coordenadas espaciais (sinais multidimensionais).

Os sinais contínuo no tempo são sinais definidos de forma contínua no tempo e são representados por funcões de variáveis contínuas. Por sua vez, os sinais discretos no tempo são aqueles definidos de forma discreta no tempo e representados por seqüências de números.

Além de se ter a variável independente contínua ou discreta, pode-se ter a variável dependente (amplitude) contínua ou discreta. Assim, os sinais digitais são aqueles representados de modo discreto no tempo e na amplitude, enquanto os sinais analógicos são representados de modo contínuo no tempo e na amplitude. Dentro deste contexto, sistemas digitais são aqueles cujos sinais de entrada e saída se encontram na forma digital.

Os sinais discretos no tempo podem ser resultado de uma amostragem de um sinal contínuo ou terem sua origem através de um processo intrinsicamente discreto, ou seja, estes são gerados diretamente discretos no tempo. Quando estes sinais resultam de

uma amostragem, tem-se que a conversão é feita por um conversor analógico-digital. Num processo de amostragem, tem-se que a maior componente de freqüência do sinal amostrado seja no máximo metade do valor da freqüência de amostragem, de modo a permitir posterior recuperação do sinal [OPPE75].

2.2.2 - Caracterização do filtro digital

O termo "filtro digital" se refere ao algoritmo de cálculo através do qual um sinal digital ou seqüência de números (agindo como entrada) é transformada numa segunda seqüência de números chamada de sinal digital de saída [LOVR86].

Para que esse sistema seja linear, considera-se 2 (duas) seqüências x1(n) e x2(n) como entrada desse sistema e suas respectivas seqüências de saída y1(n) e y2(n). Dessa forma, se for aplicada à entrada a seqüência ax1(n) + bx2(n), a seqüência obtida na saída será ay1(n) + by2(n), onde a e b são constantes quaisquer.

No caso de um sistema ser invariante no tempo, tem-se que se uma seqüência x(n) for aplicada à entrada, produzirá na saída uma seqüência y(n) e se for aplicada à entrada uma seqüência x(n-n0), esta resultará na saída uma seqüência y(n-n0) para qualquer n0 [RABI75].

Para os filtros com as características acima, isto é, de ser linear e invariante no tempo, tem-se duas classes de filtros: a dos filtros seletivos em freqüência e a dos filtros não-seletivos em freqüência. Para os filtros seletivos em freqüência, tem-se como exemplos: filtro passa-baixa, passa-alta, passa-faixa, rejeita-faixa [OPPE83].

Na fig. 2.1.a e 2.1.b, tem-se as representações das magnitudes das respostas em freqüência dos filtros passa-baixa e passa-alta, respectivamente. Os valores wp e ws são as freqüências de corte nas faixas de passagem e rejeição, respectivamente. Na fig. 2.2.a e 2.2.b, tem-se as magnitudes das respostas em freqüência dos filtros passa-faixa e rejeita-faixa, respectivamente. Os valores wp1 e ws1 definem as primeiras freqüências de corte nas faixas de passagem e rejeição. Os valores wp2 e ws2 correspondem às freqüências de corte para a segunda faixa de passagem e rejeição. Haverão tantas wp´s e ws´s quantas forem as faixas de passagem e rejeição.

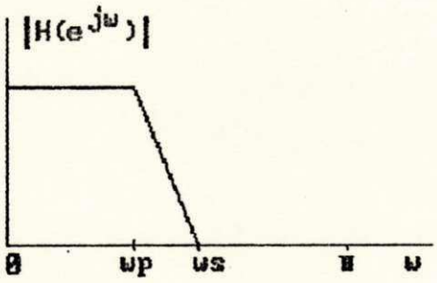

Fig. 2.1.a
Filtro passa-baixa

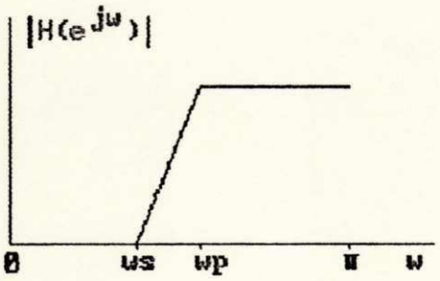

Fig. 2.1.b
Filtro passa-alta

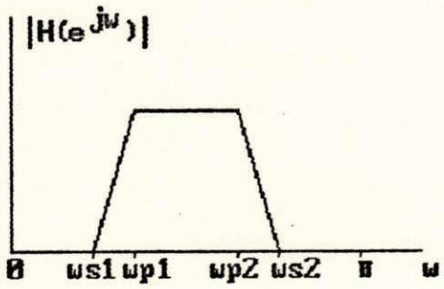

Fig. 2.2.a
Filtro passa-faixa

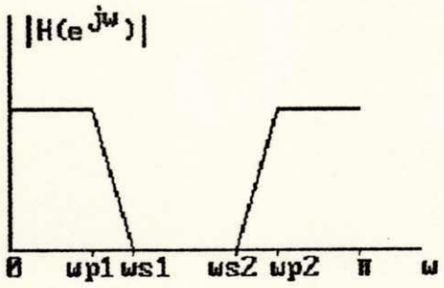

Fig. 2.2.b
Filtro rejeita-faixa

Os filtros seletivos em freqüência permitem a passagem de uma ou mais faixas de freqüências e atenuam ou eliminam as demais faixas. Há uma grande quantidade de aplicações para essa classe de filtros. Por exemplo, os filtros passa-baixa podem ser usados para a remoção de ruído de altas freqüências em sistemas de áudio.

Exemplificando os filtros não-seletivos em freqüência, tem-se o diferenciador. O diferenciador é caracterizado por ter a magnitude da resposta em freqüência variando linearmente com a freqüência. Os diferenciadores são usuais na identificação de transições rápidas de um sinal e uma aplicação para eles é na identificação de bordas no processamento de imagens [OPPE83]. Uma vez que o diferenciador, apresenta-se com sua amplitude variando linearmente com a freqüência, pode também ser utilizado como um discriminador de freqüência (elemento sensível à variação de freqüência), na detecção do sinal modulador de uma portadora FM [LATH79].

Dentro do conjunto de especificações de um filtro, tem-se o conjunto de freqüências que definem o tipo de filtro quanto a sua seletividade em freqüência. Quando da especificação de filtros digitais, são definidas as freqüências de corte do filtro e, normalmente, estas são normalizada em relação a freqüência de amostragem.

Também, quando da especificação do filtro, informa-se as tolerâncias permitidas nas faixas de passagem ($\delta 1$) e de rejeição ($\delta 2$).

Neste trabalho, a tolerância $\delta 1$ é referida como desvio (ripple) máximo na faixa de passagem e a tolerância $\delta 2$ é referida

como atenuação mínima na faixa de rejeição.

A ordem do filtro (N) está relacionada à quantidade de coeficientes do filtro, consequentemente a sua complexidade. Esta relação é feita para os filtros com resposta ao impulso finita ou FIR (Finite Impulse Response) e para os filtros com resposta ao impulso infinita IIR (Infinite Impulse Response). A classificação dos filtros quanto ao tipo de resposta ao impulso é mostrada a seguir.

O filtro com resposta ao impulso finita, comumente, chamado de FIR é aquele cuja resposta ao impulso é uma seqüência de duração finita. Como função de transferência, tem-se:

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

Considerando h(n) a resposta ao impulso de um sistema linear e invariante no tempo (LIT) e {h(n)} uma seqüência FIR, verifica-se que esse sistema LIT é estável pois a condição de estabilidade envolve uma soma finita de valores finitos [RABI75], isto é:

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty$$

A função de transferência do filtro FIR consiste portanto apenas de zeros, sendo algumas vezes este filtro referido de "all-zero" [WILL88]. Este filtro apresenta a propriedade de ter a fase exatamente linear, ou seja, tem a fase variando linearmente com a freqüência.

Para se obter uma característica da magnitude da resposta em freqüência do filtro com corte abrupto (sharp cutoff) na região

de transição, é necessário que se tenha um grande valor de N (tamanho do filtro). Uma observação interessante feita por Rabiner é que em projetos de filtros onde se tenha grandes valores para a tolerância δ1, pequenos valores para a tolerância δ2 e largura extensa na faixa de transição, os filtros FIR são mais recomendados [RABI74].

Já o filtro com resposta ao impulso infinita, conhecido como filtro IIR é aquele cuja resposta ao impulso é uma seqüência de duração infinita, isto é, sua resposta ao impulso se estende de $-\infty$ ou para $+\infty$ ou ambos. A função de transferência de um filtro IIR é:

$$H(z) = \sum_{}^{\infty} h(n)z^{-n} = \frac{\sum_{i=0}^{M} b_i z^{-n}}{\sum_{i=0}^{N} a_i z^{-n}}$$

O principal objetivo no projeto de um filtro digital é satisfazer à magnitude pré-especificada da resposta em freqüência. Em comparação aos filtros FIR, os filtros IIR se apresentam com um valor de ordem significativamente menor, considerando-se a necessidade de satisfazer às características da magnitude.

Geralmente, utiliza-se o número de multiplicações por amostra para avaliar a capacidade (throughput rate) do sistema, tanto em hardware quanto em software [RABI74]; assume-se ser esse número de multiplicações proporcional à velocidade de processamento; assim, tem-se que os filtros IIR são mais rápidos, isto é, têm o processamento mais eficiente [WILL88].

Grande parte da teoria de projetos de filtros digitais IIR envolve as técnicas de projetos de filtros analógicos que apresentam como aproximações as de Butterworth, Chebyshev e Elíptica.

A aproximação Butterworth apresenta borda pouco íngreme (transição suave) na faixa de transição da resposta em freqüência da magnitude. Apresenta a faixa de passagem maximamente plana [WILL88].

A aproximação Chebyshev apresenta a resposta em freqüência da magnitude mais retangular (borda mais íngreme) do que a de Butterworth na faixa de transição (faixa de transição mais estreita). Como na faixa de passagem, tem-se o ripple maior, a transição é mais rápida [WILL88].

A aproximação Elíptica apresenta um comportamento equiripple na faixa de passagem similar ao Chebyshev. Esse tipo de aproximação elíptica tem uma taxa de decaimento mais rápida (mais aguda) do que os demais; Para uma ordem N do filtro, a atenuação mínima da faixa de rejeição diminui quando a transição da faixa de passagem para a faixa de corte (sharpness) aumenta, ou vice-versa [OPPE75].

2.2.3 - Análise Comparativa Quanto ao Uso de Filtro Digitais e Analógicos

A função de seletividade em frequëncia é fornecida pelos filtros analógicos e digitais, pois ambas categorias de filtros têm como objetivo principal o de satisfazer a uma resposta em freqüência da magnitude.

Os filtros analógicos consistem de elementos passivos como indutores, capacitores e resistores, e possivelmente de elementos ativos como amplificadores operacionais ou transistores.

Os filtros digitais são implementados num hardware digital de alto desempenho para proposta geral ou através de circuitos integrados de aplicação específica (ASIC - Application Specific Integrated Circuits) [WILL88].

Dessa forma, observa-se que os filtros digitais não apresentam problemas quanto ao casamento de impedância de entrada ou saída. Também, os filtros digitais não necessitam de ajuste periódico, nem têm seu desempenho degradado devido ao envelhecimento, fatos estes observados nos filtros analógicos.

Uma importante vantagem observada é que os filtros digitais quando implementados com um processador programável tal como o TMS320, permite que sejam modificados os parâmetros do filtro modificando portanto suas características, sem que alterações de hardware sejam necessárias [LOVR86].

Os filtros digitais com resposta ao impulso finita - FIR fornecem resultados com excelente linearidade de fase quando comparados com os filtros IIR.

Os filtros digitais, quando processando sinais analógicos, necessitam de conversores analógico-digital (A/D) e digital-analógico (D/A) que têm a função de converter os sinais da forma analógica para a forma digital e vice-versa, respectivamente. Isto, pois os sinais quando adquiridos vêm originalmente na forma analógica, necessitando voltar também na forma analógica. No entanto, o processamento do sinal é feito na forma digital.

Quando do uso de filtros digitais, é necessário que seja

feito um estudo sobre os efeitos de erros aritméticos devido ao fato dos dispositivos práticos serem de precisão finita [LOVR86].

2.2.4 - Etapas de Projeto de um Filtro Digital:

As etapas a serem seguidas quando do projeto de um filtro digital é sumarizada abaixo:

1.a - determinar os coeficientes do filtro que satisfaçam às especificações de desempenho do mesmo (problema de aproximação);

2.a - escolher uma estrutura específica na qual o filtro será realizado e quantizar os coeficientes numa palavra de tamanho fixo;

3.a - quantizar as variáveis de entrada, saída e intermediárias do filtro digital;

4.a - simular o filtro a fim de verificar se o projeto resultante atende às especificações de projeto.

Os resultados obtidos na 4.a etapa pode levar a revisões nas 2.a e 3.a etapas.

2.3 - Técnicas para Projetos de Filtros Digitais

Para os filtros digitais com resposta ao impulso finita (FIR) e resposta ao impulso infinita (IIR), têm-se as técnicas apresentadas a seguir.

2.3.1 - Técnicas para Projetos de Filtros IIR

O problema de projetar um filtro digital é o de encontrar os coeficientes do filtro tal que a resposta do filtro satisfaça ao comportamento desejado pelo projetista. Este problema de projetar um filtro é basicamente um problema de aproximação matemática.

Como solução deste problema, a técnica utilizada para projetar um filtro digital IIR é realizar um projeto de um filtro analógico e fazer a transformação deste num filtro digital. Para se realizar um projeto de um filtro analógico, tem-se várias técnicas de aproximação, dentre elas: Butterworth, Chebyshev e Cauer ou elíptica. Esta técnica usa os procedimentos de projetos de filtros analógicos já desenvolvidos, tornando simples a transformação analógico-digital do filtro. Se o problema de aproximação é realizado no plano-s, o filtro resultante é um filtro analógico. No caso de um filtro digital, tem-se que o problema de aproximação é realizado no plano-z. Na transformação de um filtro analógico em um filtro digital, pode-se ter um dos 3 (três) procedimentos seguintes: a transformação invariante ao impulso, a transformação bilinear ou o mapeamento de diferenciais.

Tem-se ainda os métodos de projetos de filtros digitais IIR, como métodos de otimização, nos quais um procedimento de otimização matemática é usado para determinar os coeficientes do filtro que minimizam o erro segundo algum critério. O algoritmo IIR2 [DECZ79] apresentado na seção 4.1 do capítulo 4 é um exemplo.

Uma discussão mais detalhada sobre essas técnicas de projeto

é feita por Oppenheim [OPPE75].

2.3.2 - Técnicas para Projetos de Filtros FIR

Geralmente, os filtros digitais FIR apresentam a fase linear, sendo então as técnicas de projeto para filtros FIR de considerável interesse quando se deseja preservar a linearidade da fase.

Relativo as técnicas de projetos de filtros digitais FIR com fase linear, tem-se: a técnica de janelas e a técnica de amostragem no domínio da freqüência. Outra técnica de projeto de filtros digitais FIR é a técnica de projetos de filtros ótimos, a exemplo dos filtros IIR.

A técnica de janelas (window) é a forma de se obter um filtro, na qual é feito um truncamento, usando-se uma seqüência de ponderação finita w(n), chamada janela, para modificar os coeficientes de filtro $h_d(n)$. Assim, tem-se:

$$h(n) = h_d(n)w(n) \ , \ 0 \leq n \leq N-1$$

$$h(n) = 0 \ , \text{ fora deste intervalo}$$

Há vários tipos de janelas que procuram aproximar as características desejadas para o filtro digital. Dentre elas, tem-se: as janelas retangular, generalizada de Hamming e a de Kaiser.

A técnica de amostragem no domínio da freqüência é baseada na especificação de um período da resposta em freqüência através de um conjunto de amostras. A idéia principal desta técnica é que a resposta em freqüência desejada pode ser aproximada por sua amostragem em N amostras (pontos), espaçadas igualmente, e então

obter a resposta em freqüência interpolada que passa através do conjunto de amostras [OPPE75].

Outra técnica utilizada quando do projeto de filtros digitais FIR com fase linear pode ser formulado como um problema de aproximação de Chebyshev, no qual se tem uma função de ponderação do erro de aproximação. Rabiner apresentam uma abordagem completa das características de projetos de filtros digitais FIR [RABI75].

2.4. Areas de Aplicação de PDS

Nesta seção, ver-se-á caracterizado de forma sumária algumas das áreas de aplicação de processamento digital de sinais.

A área que apresenta maior ênfase é a de processamento de voz, área esta para a qual se fez a modelagem da base de conhecimento.

2.4.1. Processamento de Voz

Algumas das aplicações mais importantes das técnicas de processamento digital de sinais têm sido feitas na área de processamento de voz. Isto pelo fato de grande parte da base teórica de processamento digital de sinais ter se derivado de estudos feitos na área de processamento de voz.

Geralmente, a área de processamento de voz pode ser dividida em 3 (três) classes de aplicações.

A primeira classe de aplicação envolve apenas a análise do sinal de voz. Pode-se ter, como exemplo, o reconhecimento da fala, onde a partir do reconhecimento automático de parâmetros,

intrínsecos a determinadas palavras ou conjunto de palavras, uma ação é realizada. Outro exemplo dado é o de identificação de locutor.

A segunda classe de aplicação envolve a síntese do sinal de voz, onde se tem, como exemplo, um sistema de leitura automática (text to speech) no qual a entrada é um texto escrito e a saída é o sinal de voz.

Na terceira classe de aplicação, tem-se a codificação do sinal de voz. Como exemplo, pode-se ter o canal vocoder (voice coder) que é um sistema de análise e síntese do sinal de voz baseado no conhecimento que se tem do mecanismo de produção e percepção da voz humana.

A voz humana é o produto de uma excitação original, que ocorre no pulmão, para em seguida ser modificada (modulada) pela região conhecida como trato vocal.

O primeiro tipo de excitação, responsável pela produção de sons sonoros, é a modulação que ocorre com a vibração das cordas vocais quando do fluxo de ar produzido pelo pulmão. O sinal desta excitação é periódico e chamado de freqüência fundamental (freqüência "pitch"). A freqüência fundamental corresponde à freqüência dos pulsos (sinal) de excitação da região de articulação. Os valores tipicos para os homens é de 120 Hz e para as mulheres é de 240 Hz [OPPE78].

O segundo tipo de excitação, responsável pela produção de sons surdos, é a turbulência do ar provocada pela constricção da região vocal. Este sinal tem características de ruído. A região vocal atua como um elemento ressonante que modifica o sinal de excitação. Para os sons sonoros, a região vocal, normalmente,

apresenta 4 ressonâncias as quais são chamadas de formantes e são responsáveis pela caracterização do som.

Do ponto de vista da percepção, tem-se que apenas os três primeiros formantes são suficientes na determinação do som escutado pelas pessoas [RABI75]. Porém, para se ter a inteligibilidade preservada, ou seja, uma percepção a qual se permita identificar um locutor, conserva-se os 4 (quatro) primeiros formantes.

A faixa de freqüência na qual se tem o sinal de voz vai de 80 Hz a 12 kHz.

O sinal de voz apresenta maior concentração de energia nos 2 (dois) primeiros formantes. Também, tem-se que o sinal de voz apresenta maior concentração de energia nas baixas freqüências.

De um modo geral, é desejável que nas aplicações de processamento de sinais de voz a linearidade de fase seja preservada [RABI75].

Do ponto de vista comparativo, tem-se que os filtros com resposta ao impulso finita (FIR) exibe um comportamento de fase linear melhor do que os filtros com resposta ao impulso infinita, sendo sua aplicação usual para processamento de sinais de voz [WILL88].

2.4.2. Áudio

Os sistemas pertinentes à área de áudio apresentam como funções principais:

- aquisição de sinais fonte;
- armazenagem (gravação) dos sinais de áudio;

- processamento;
- transmissão;
- reprodução.

Ao contrário do sinal de voz processado digitalmente que está sujeito a um teste de inteligibilidade, o sinal de áudio digitalizado precisa atender a um critério de fidelidade. Este critério de fidelidade é necessariamente subjetivo, pois o teste final de qualidade é baseado na percepção dos ouvintes.

Foi observado que o ouvido humano é mais sensível às freqüências na faixa de 1000 Hz a 5000 Hz, caindo em aproximadamente 20 dB nas freqüências próximas de 15 kHz [BLES78].

Tem-se ainda que o ouvido humano, normalmente, pode perceber freqüências situadas entre 20 Hz e 15 kHz, sendo esta uma largura de faixa aceitável. No entanto, nos sistemas digitais, como Compact Disc e Sistemas Digitais de Gravação, tem-se especificado a resposta em freqüência na faixa de 20 Hz a 20 kHz [BLO085].

Como outras aplicações de sistemas de áudio, tem-se:
- equipamentos e sistemas digitais de áudio em geral;
- sistema de gravação e mixagem digital;
- sintetizadores musicais.

2.4.3. Comunicações

Os sinais são elementos fundamentais em todos os sistemas de comunicações. Também, tem se observado uma crescente utilização das técnicas de processamento digital de sinais nesta área. Isto se deve em grande parte a disponibilidade de circuitos integrados

com características desejáveis como: tamanho reduzido, baixo consumo de potência, baixo custo, imunidade ao ruído e confiabilidade [FREE78].

De um modo geral, nos sistemas de comunicações se procura fazer a codificação do sinal (fonte) com o objetivo de reduzir a largura de faixa do canal, a potência de transmissão, o tempo de transmissão e a capacidade de armazenagem (memória). Isto se reflete como uma melhoria no desempenho dos sistemas.

Como aplicações de técnicas de processamento digital de sinais dentro desta área, tem-se:

- telefonia digital;
- multiplexação de canais;
- modem´s;
- equalizadores;
- transmissão digital.

2.4.4. Biomédica

Outra área de aplicação de processamento digital de sinais é a área de biomédica. Como exemplo de aplicações, tem-se: ferramentas de diagnósticos (eletrocardiograma-ECG, eletroencefalograma-EEG, eletroneurograma-ENG, eletromiograma-EMG e eletroretinograma-ERG), equipamentos de ultrasonografia.

A área de biomédica está relacionada com os sinais bioelétricos, os quais permitem registrar os seguintes diagnósticos: EEG, ECG, ENG, EMG e ERG.

Os potenciais bioelétricos são produzidos como um resultado da atividade eletroquímica de uma certa classe de células,

conhecidas como células estáveis, as quais compõem o tecido nervoso, muscular ou grandular. Tais células possuem um potencial em repouso que quando estimulado apresenta um potencial de ação.

No estado de repouso, as células excitáveis apresentam uma ddp entre os meios interno e externo. O potencial de repouso no meio interno em relação ao meio externo se situa na faixa de -50 a -100mV [WEBS78].

2.3.5. Outras

São diversas as área de aplicação de processamento digital de sinais. É mostrado a seguir algumas delas [LOVR86].

- instrumentação:

i) análise de espectro;

ii) processamento sísmico;

iii) filtragem digital.

- imagem:

i) visão robótica;

ii) transmissão e compressão de imagens;

iii) reconhecimento de padrões;

iv) realce de imagens.

- controle:

i) robótica;

ii) servo controle;

iii) controle de máquinas.

- militar:

i) Comunicações sigilosas;

ii) processamento de radar;

iii) " de sonar;

iv) " de imagem;

v) navegação.

- automotiva:

i) controle de máquina(motor);

ii) freios anti-derrapantes;

iii) navegação;

iv) rádio digital;

v) comandos por voz.

- industrial:

i) robótica;

ii) controle numérico;

iii) monitores de linha de potência;

iv) Acesso de segurança.

2.4. Resumo

Neste capítulo, foi apresentado de modo resumido um estudo sobre filtros digitais e as características de algumas áreas de aplicação de processamento digital de sinais.

Capítulo 3

Aquisição e Representação de Conhecimento

Neste capítulo, são apresentados conceitos relativos à aquisição e representação de conhecimento.

3.1. Conceitos Básicos

Um Sistema Inteligente é definido como aquele que pode dispor apropriadamente do conhecimento relevante para solução de tarefas complexas. Assim, o sistema deve ter a sua disposição o conhecimento e recursos necessários para execução das tarefas [KING86].

Um Sistema Especialista ou Sistema Baseado em Conhecimento é definido como um programa de computador o qual reúne uma coleção de regras heurísticas e fatos de um domínio necessários à solução de problemas de uma área específica [RYCH88].

Uma Base de Conhecimento é uma porção de um sistema baseado em conhecimento ou sistema especialista que agrupa numa estrutura (forma de representação) o conhecimento de um domínio específico.

Os procedimentos envolvidos no desenvolvimento de sistemas especialistas estão relacionados a engenharia de conhecimento. O engenheiro de conhecimento é o elemento responsável pela aquisição de conhecimento junto ao especialista, bem como estruturação das informações obtidas [WATE86].

As atividades de aquisição e representação de conhecimento desenvolvidas neste trabalho serão abordadas com mais detalhes no capítulo 4, constituindo-se nas etapas críticas do

desenvolvimento de um sistema especialista.

3.2. Aquisição de Conhecimento

O objetivo do processo de aquisição de conhecimento é o de estruturar as informações que irão compor a base de conhecimento de um sistema especialista.

Trata-se de um processo através do qual é feita uma coleta de conhecimento com o objetivo de reunir informações sobre um determinado domínio possibilitando estruturar a base de conhecimento de um sistema.

Tal processo, no desenvolvimento de um sistema especialista, é visto como um "gargalo" (bottleneck), isto é, a etapa mais difícil. Esta dificuldade é observada, principalmente, quando a fonte de conhecimento é um ser humano (especialista) e também pela inexistência de métodos formais [COOK86].

Até o presente, não existem procedimentos automáticos para aquisição de conhecimento, exceto quando se tem um conjunto de exemplos [WATE86].

Os métodos de aquisição de conhecimento são abordados em 3.2.2.

3.2.1. Aspectos Relacionados à Aquisição de Conhecimento

Há vários aspectos inerentes na engenharia de conhecimento, conforme visto a seguir, que contribuem para dificultar a transferência de conhecimento do especialista para o engenheiro de conhecimento. O processo de aquisição de conhecimento do especialista não é bem definido e geralmente envolve interação

entre o engenheiro de conhecimento e o especialista.

O processo de entrevista conduzido pelo engenheiro de conhecimento junto ao especialista é esquematizado na fig. 3.1. Neste processo, o engenheiro de conhecimento apresenta questões e problemas para que o especialista revele soluções, conceitos, conhecimento sobre o domínio enfocado.

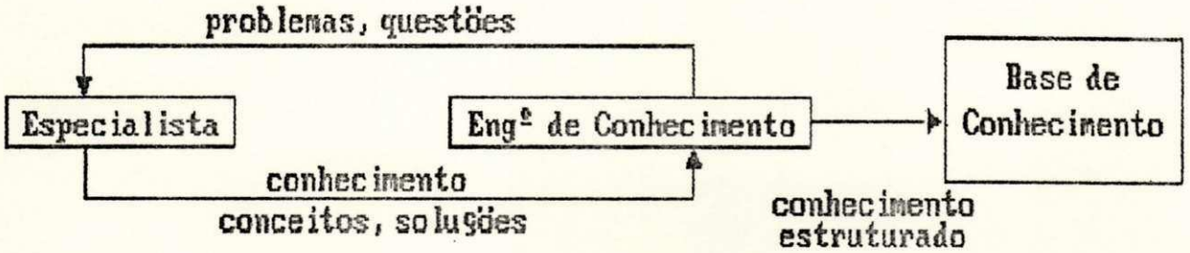

Fig. 3.1 - Consulta do Engº de Conhecimento com o Especialista

Outros aspectos a serem considerados são:

- o processo de aquisição de conhecimento não deve ser condicionado pelo processo de representação de conhecimento;
- o processo de aquisição de conhecimento através de entrevistas não é perfeito, pois envolve introspecção do especialista e interpretação subjetiva do engº de conhecimento;
- o aumento de conhecimentos do especialista diminui sua habilidade de espressar, verbalmente, conhecimento [COOK86];

3.2.2. Métodos de Aquisição de Conhecimento

a) Método Cognitivo - Baseado na competência de um mediador humano (engenheiro de conhecimento), conforme visto na fig. 3.1.

Para este método, deve-se considerar:

- interessante o especialista conhecer o sistema e suas finalidades a fim de que haja melhor interação entre o engenheiro de conhecimento e o especialista, facilitando o processo;
- é difícil para os especialistas expressarem a compilação de conhecimento, isto é, uma combinação e/ou redução nos passos utilizados no processo de inferência mental para obtenção de resultados;
- a perícia humana tem aspecto similar a intuição, tornando-a difícil sua codificação em regras [COOK86];
- a análise subjetiva do engenheiro de conhecimento pode gerar regras de conflito e deficiência no desempenho do sistema especialista;
- tem-se observado diferenças entre especialistas e aprendizes, no que diz respeito a habilidade de reconhecimento de padrões, organização de memória, solução de problemas e tomada de decisões [AKIN88].

b) Método Informatizado - Tem-se que o conhecimento é obtido de forma automática. Relativo a este método, deve-se considerar:

- evita/minimiza o contato com o engenheiro de conhecimento;
- direcionado a problemas de diagnóstico, classificação, interpretação;
- necessita de um conjunto de exemplos - APREND.

Como exemplos de ferramentas desenvolvidas para aquisição de informações para construção de bases de conhecimento, tem-se:

- TEIRESIAS - uma ferramenta que auxilia no processo de transferência de conhecimento do especialista para a base de conhecimento, interagindo com o especialista de um domínio específico a fim de estabelecer um conjunto de regras [WATE86];
- APREND - um sistema de aquisição automática de conhecimento, permitindo gerar um conjunto de regras; Para isto, necessita-se de um conjunto de exemplos [GOME89].

3.3. Representação de Conhecimento

Nesta seção, são considerados aspectos gerais da representação de conhecimento e os tipos mais comuns.

3.3.1. Aspectos Relacionados à Representação de Conhecimento

Na ciência da computação, uma boa solução, frequentemente, está relacionada a uma boa representação da informação.

Segundo Woods, um sistema de representação de conhecimento deveria permitir as capacidades de percepção, raciocínio, planejamento e controle de ações sobre um domínio específico [WOOD86].

Para a solução de problemas, necessitamos de conhecimentos que estão no domínio desses problemas e mecanismos para armazenar e manipular esses conhecimentos. Um sistema de representação de conhecimento deve ser capaz de considerar as informações e particularidades do domínio em questão.

Além disso, um sistema de representação de conhecimento deve executar tarefas cognitivas, identificando e manipulando

informações para buscar a solução do problema proposto. Dessa forma, segue-se às etapas abaixo quando da representação de conhecimento:

a) adquire mais conhecimentos;

b) recupera informações da base de conhecimento relacionadas com o problema;

c) raciocina sobre essas informações para encontrar a solução.

A partir do processo de aquisição de conhecimento, observa-se as características pertinentes ao domínio que permite identificar o modelo de representação de conhecimento [ARIT86].

3.3.2. Tipos de Representação de Conhecimento

Tem-se que as três formas mais conhecidas de se representar o conhecimento são: **regras, quadros** (frames) e **rede semântica** [WATE86].

A representação de conhecimento através de **regras de produção** é a técnica mais difundida. Esse tipo oferece uma forma de representar estratégias ou diretivas. É apropriado quando o conhecimento do domínio é resultado de associações empíricas desenvolvida com a solução de problemas.

As regras são expressas através de procedimentos do tipo:

SE condição ENTAO ação

Uma coleção de regras de produção que é interpretada para produzir uma ação (comportamento) de acordo com um procedimento específico é chamada de **sistema de produção**.

Normalmente, tem-se um conjunto de fatos contra os quais um conjunto de regras é "casado" (matched). Este processo pode ser chamado também de "resoluçao de conflito", na qual uma ou mais regras são selecionadas para casar com a situação corrente (vide fig. 3.2).

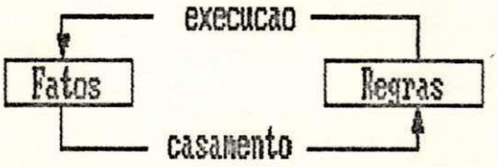

Fig. 3.2 - Resolução de conflito

A **rede semântica** é um tipo de representação na qual os conceitos ou objetos (nós) se encontram interconectados por arcos que expressam uma relação entre os mesmos. Na fig. 3.3, tem-se um exemplo genérico de uma rede semântica.

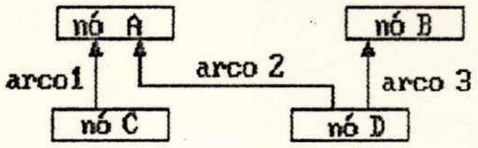

Fig. 3.3 - Exemplo de rede semântica

A representação baseada em **quadros** (frame) é um modo de expressar o conhecimento declarativo, no qual um objeto é representado por uma estrutura de dados contendo um número de **slots** (representando atributos ou relações do objeto), sendo cada slot preenchido com um ou mais valores (representando valores específicos de atributos ou outros objetos ao qual está relacionado).

A fig. 3.4 exemplifica um nó (coleção de atributos) em um frame.

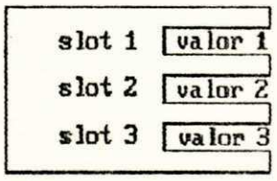

Fig. 3.4 - Exemplo de conceito de um nó frame

3.4. Resumo

Neste capítulo, apresentou-se um estudo sobre os processos de aquisição e de representação conhecimento.

Capítulo 4

Aquisição e Representação de Conhecimento: Modelagem e Implementação

No capítulo 3, foram apresentados os principais conceitos sobre aquisição e representação de conhecimento. Neste capítulo, é apresentado o desenvolvimento feito neste trabalho. Assim, inicialmente, são apresentados os procedimentos utilizados para o processo de aquisição de conhecimento.

4.1. Metodologia para Modelagem da Base de Conhecimento

O conteúdo da base de conhecimento engloba informações sobre processamento digital de sinais, suas áreas de aplicações e sobre os algoritmos utilizados para projetos de filtros digitais.

Face às observações mencionadas no capítulo 3, procurou-se seguir a uma seqüência de procedimentos, conforme descrito abaixo, com o objetivo de realizar a modelagem da base de conhecimento.

Inicialmente, fez-se estudos sobre Processamento Digital de Sinais (PDS), áreas de aplicação de PDS e algoritmos para projetos de filtros digitais, conforme apresentado no capítulo 2.

Em seguida, foi apresentada à colaboradores, especialistas nas áreas abordadas, a estrutura do sistema, a funcionalidade de seus blocos, bem como seu objetivo final.

Dessa forma, procurou-se utilizar a seqüência de procedimentos, descrita a seguir, com a finalidade de adquirir e modelar o conhecimento (a fig. 4.1 mostras esses procedimentos).

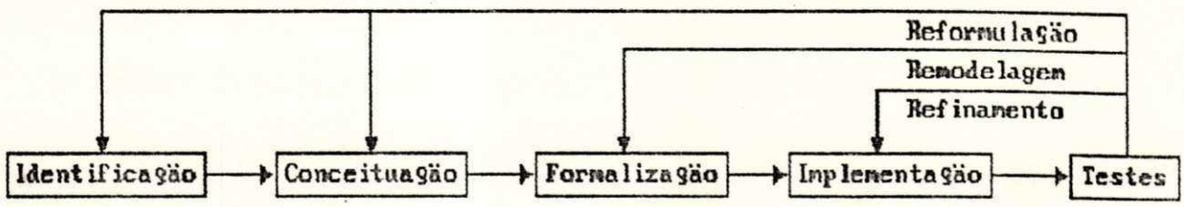

Fig. 4.1 - Processo de Modelagem da Base de Conhecimento

a) **Fase de Identificação e Conceituação**

Procurou-se identificar conceitos e aspectos relevantes a PDS, às áreas de aplicação de PDS e aos algoritmos para projetos de filtros digitais, bem como o casamento entre as informações, objetivando auxiliar na seleção de algoritmos para projetos de filtros.

Tal processo foi realizado, fazendo-se leitura de livros/artigos pertinentes ao domínio de interesse, consultas a colaboradores(especialistas) e análise numérica e gráfica do comportamento dos algoritmos a serem utilizados no sistema especialista.

As informações pertinentes a área de processamento de voz que permitiu estruturar a base de conhecimento se encontram sumarizadas no capítulo 2. Relativo ao conjunto de algoritmos propostos, foram usados um conjunto de algoritmos selecionados pelo DSP Committee da IEEE ASSP Society [IEEE79]. Estes algoritmos são usados no sistema SIPREX. Além disso, tem-se que os mesmos são divididos em duas categorias, classificadas quanto às suas respostas ao impulso, sendo elas: FIR e IIR. Os algoritmos-soluções são apresentados a seguir.

a.1. Algoritmos para Síntese de Filtros FIR

Algoritmo FIR1 - O algoritmo implementa o método de trocas de Remez para projeto de filtros digitais FIR de fase linear utilizando a aproximação mínima do erro ponderado de Chebyshev para a resposta em freqüência desejada [McCL79].

Este algoritmo permite projetar os seguintes filtros passa-baixa, passa-alta, passa-faixa(s), rejeita-faixa(s), diferenciador e transformador de Hilbert.

Para N (tamanho do filtro) constante, se a largura da faixa de transição aumentar o ripple (faixa de passagem) diminui; Para largura da faixa de transição (L) constante, se N aumentar o ripple $\delta 1$ na faixa de passagem diminui. Esta observação é válida para os demais algoritmos.

Dentre os algoritmos selecionados, este é o único algoritmo FIR que implementa filtros multi-passa-faixa e multi-rejeita-faixa, ou seja, filtros passa-faixa e rejeita-faixa com multiplas faixas de passagem e rejeição.

Algoritmo FIR2 - O algoritmo implementa o método de janelas para o projeto de filtros digitais de fase linear [RABI79].

Projeta os seguintes filtros: passa-baixa, passa-alta, passa-faixa e rejeita-faixa.

Para o projeto de um filtro, este algoritmo permite a escolha de um tipo de janela, dentre as seguintes janelas: retangular, triangular, Hamming, generalizada de Hamming, Hanning, Kaiser e Chebyshev.

Algoritmo FIR3 - O algoritmo sintetiza filtro digital

simétrico com característica de magnitude "maximally flat" (maximamente plana) na faixa de passagem e rejeição [KAIS79].

Este algoritmo permite apenas projeto de filtros passa-baixa.

Durante a utilização deste algoritmo, verificou-se que ele apresentava resultados apenas quando da especificação das freqüências de corte do filtro, estas produziam valores da largura da faixa de transição normalizada superiores a 0.1.

Algoritmo FIR4 - Este algoritmo sintetiza filtros digitais que satisfaçam às especificações de tolerância nas faixas de passagem e rejeição ($\delta 1$ e $\delta 2$, respectivamente). Além disso, este algoritmo procura minimizar o tamanho da palavra necessária a quantização dos coeficientes, realizando uma modificação (rounding) dos coeficientes , levando em consideração as tolerâncias pré-especificadas [HEUT79].

Projeta os seguintes filtros: passa-baixa, passa-alta, passa-faixa, rejeita-faixa e transformador de Hilbert.

O resultado é filtro FIR de fase linear com palavra de tamanho mínimo, tamanho este dependente dos valores de tolerância desejados.

a.2. Algoritmos para Síntese de Filtros IIR

Algoritmo IIR1 - O algoritmo sintetiza filtros digitais IIR recursivos [DEHN79].

Permite projetar os seguintes filtros: passa-baixa, passa-alta, passa-faixa e rejeita-faixa, sendo estes dois últimos

simétricos.

As aproximações disponíveis são as de Butterworth (magnitude maximamente plana), Chebyshev (tipos I e II) e elíptico.

Quando da utilização deste algoritmo para síntese de filtros digitais, verificou-se que a aproximação que apresentou os melhores resultados foi a elíptica. Esta observação se fez quando o resultado final desejado no projeto de um filtro era uma combinação dos seguintes fatores: um valor reduzido na largura da faixa de transição (isto é, um corte abrupto - sharp cutoff) e um valor reduzido da tolerância $\delta 1$ (ripple na faixa de passagem).

Esta análise considerou as aproximações mencionadas acima.

Como resultado, este algoritmo fornece um conjunto de pólos e zeros, além de fornecer um conjunto de coeficientes.

Algoritmo IIR2 - Este algoritmo sintetiza filtros digitais IIR recursivos utilizando o critério de erro **p**-mínimo e o método de Fletcher e Powell para minimização da função [DECZ79].

O critério de erro p-mínimo e o método de Fletcher e Powell são discutidos por Deczky [DECZ72].

Projeta os seguintes filtros: passa-baixa, passa-alta, passa-faixa(s) e rejeita-faixa(s).

Este algoritmo permite que se façam aproximações apenas da magnitude, apenas do atraso de grupo (group delay), equalização do atraso de grupo e aproximação de magnitude e atraso de grupo combinada.

Possibilita a síntese de filtros de fase linear.

No caso de projeto multi-faixas, essas estão limitadas a 10 (dez).

O algoritmo fornece os coeficientes do filtro a partir de um conjunto de pólos e zeros.

Para a determinação deste conjunto de pólos e zeros, pode-se utilizar o algoritmo apresentado anteriormente IIR1.

Algoritmo IIR3 - Este algoritmo varia os coeficientes do filtro até que sejam atendidas as especificações desejada para a magnitude da resposta em freqüência [DOLA79]. Dessa forma, tem-se que o algoritmo realiza uma otimização, ou seja, obtenha o resultado no domínio da freqüência que satisfaça às especificações de projeto.

Projeta os seguintes filtros: passa-baixa, passa-alta, passa-faixa(s) e rejeita-faixa(s).

A otimização é feita para filtros de ordem (N) par. Para filtros de ordem ímpar, usa-se a ordem par seguinte, fazendo os coeficientes restantes iguais a zero.

Algoritmo IIR4 - Este algoritmo sintetiza filtros digitais IIR com palavras de tamanho finito, satisfazendo às especificações da magnitude no domínio da freqüência [STEI79].

Possibilita projetar os seguintes filtros: passa-baixa, passa-alta, passa-faixa(s) e rejeita-faixa(s).

Este algoritmo fornece um conjunto de coeficientes arredondados a partir de um conjunto de coeficientes de alta precisão, satisfazendo aos requisitos de tolerância nas faixas de passagem e rejeição, tomando por base os tamanhos de palavras definidas pelo projetista. Esse conjunto de coeficientes de alta precisão seriam fornecidos por outro algoritmo. Por exemplo, poderia coletar os resultados dos algoritmos anteriores como

opção.

A análise numérica e gráfica feita com o objetivo de identificar o comportamento dos algoritmos que compõem o sistema SIPREX se constituiu de uma bateria de exemplos de projetos de filtros. Para tanto, fez-se a análise dos resultados fornecidos pelos algoritmos quando da variação dos parâmetros de especificação do filtro. Em função da análise dos parâmetros de especificação do filtro e dos resultados fornecidos pelos algoritmos, identificou-se faixas para os valores da ordem do filtro, atenuação desejada na faixa de rejeição do filtro e da largura normalizada da faixa de transição do filtro (esta derivada do conjunto de freqüências de corte que definem o filtro). Estas faixas são mostradas no **quadro de qualificação da árvore de conhecimento** apresentado na seção 4.4 deste capítulo.

Esta análise, aliada ao estudo preliminar feito, permitiu estruturar o conhecimento relativo aos algoritmos que associado ao conhecimento sobre processamento digital de sinais e suas aplicações resultou no conjunto de informações para a base de conhecimento.

b)**Fase de Formalização**

As informações (conhecimento/dados/conceitos) adquiridos pelo engenheiro de conhecimento (mestrando) teve seu conteúdo analisado quanto a consistência e abrangência (completeness), bem como o tipo de representação de conhecimento adequada para estrutura encontrada. Esse conjunto de informações obtidas permitiu estruturar a árvore de conhecimento apresentada na seção

4.4.

c)**Fase de Implementação**

Após a representação de conhecimento, o engenheiro de conhecimento apresenta ao colaborador (especialista) o conhecimento estruturado para que este avalie as estratégias de controle usadas para solução de problemas dentro do domínio específico.

No capítulo 5, faz-se um discussão sobre esta fase.

d)**Fase de Testes**

Como última etapa nesse conjunto de procedimentos para modelagem e implementação da base de conhecimento, o sistema é submetido a uma série de testes a fim de avaliar seu desempenho, podendo o conjunto de problemas resolvidos ser apresentado a outros especialistas para verificar diferenças nas estratégias empregadas para a solução do problema proposto.

Com a avaliação dos resultados, é observada a necessidade de se refinar ainda mais o sistema. Dependendo dos resultados obtidos, pode-se ter a necessidade de se remodelar ou até reformular o conteúdo da base de conhecimento. Nos capítulos 5 e 6, é feita uma discussão maior sobre essa etapa.

4.2. Dificuldades Encontradas na Aquisição de Conhecimento

O processo de Aquisição de Conhecimento, como citado anteriormente, tem-se mostrado como a fase mais difícil quando se objetiva o desenvolvimento de um sistema especialista.

Na fase de identificação e conceituação da modelagem da base de conhecimento, encontraram-se algumas dificuldades apresentadas abaixo.

- inicialmente, realizou-se um estudo sobre processamento digital de sinais, sobre filtros digitais, sobre as áreas de aplicação de processamento digital de sinais e finalmente sobre os algoritmos para projetos de filtros digitais FIR e IIR. Entretanto, encontrou-se dificultade de identificar informações que pudessem integrar a base de conhecimento, porque inicialmente se fez um estudo geral não direcionado ao uso dos algoritmos pertinentes à implementação deste sistema; a partir dessa observação, então se procurou identificar as informações pertinentes a base de conhecimento a partir de um estudo específico do comportamento dos algoritmos selecionados e dos possíveis resultados que estes oferecem.
- Inexistência de especialistas com experiência a nível prático ou de implementação de filtros digitais obtidos com o uso dos algoritmos envolvidos. para cobrir essa deficiência, procurou-se submeter o conjunto de algoritmos a uma bateria de testes, simulando vários projetos, com a finalidade de se fazer uma análise do comportamento e diferenças entre eles (foi feita uma análise numérica e gráfica dos resultados obtidos).
- As consultas/entrevistas feitas junto aos colaboradores (especialistas) auxiliaram no que diz respeito às informações sobre processamento digital de sinais, comportamento dos tipos de filtros e algoritmos, porém se

identificando poucas informações que relacionassem os referidos algoritmos e as áreas de aplicação de processamento digital de sinais.

4.3. Considereções Sobre o Uso de Regras de Produção

Como visto anteriormente, um sistema de produção é uma representação modular de conhecimento na qual a base de conhecimento é constituída por regras, chamadas de regras de produção da forma:

SE condições **ENTAO** ações

Para os sistemas de regras de produção, tem-se dois mecanismos de inferência: **encadeamento progressivo** (forward chaining) e **encadeamento regressivo** (backward chaining).

O encadeamento progressivo é uma estratégia de busca problema-solução pela qual é feita uma inferência a partir dos dados iniciais, fazendo com que novos dados sejam gerados e seja(m) feita(s) nova(s) inferência(s) até que uma meta seja alcançada.

O encadeamento regressivo é uma estratégia de busca problema-solução na qual se parte de uma meta ou ação e se procura fazer inferência a fim de estabelecer as submetas ou dados necessários para conseguir provar a realização da referida meta.

Face ao caráter dedutivo do conhecimento adquirido, observado em sua forma de inferência, o tipo de representação de conhecimento utilizado é baseado em regras com encadeamento

progressivo. Isto é mostrado na seção 4.4 quando da apresentação da árvore de conhecimento.

Uma forma de automatizar o processo dedutivo é o uso da linguagem Prolog (Programação em Lógica), por se tratar de uma linguagem formal e precisa, constituída por cláusulas de Horn [ARAR89].

Assim, necessita-se dos recursos abaixo para permitir uma implementação:

i) uma ferramenta para edição (construção) da base de conhecimento;

ii) um mecanismo de inferência que permita acesso ao conhecimento, possibilitando o raciocínio sobre o conteúdo da base.

Esses 2 (dois) recursos são apresentados no capítulo 5.

4.4. Apresentação da Arvore de Decisão

A árvore de decisão mostrada a seguir é uma representação gráfica do conteúdo de informações da base de conhecimento.

Associado a esta árvore de decisão, tem-se no apêndice **A** a base de conhecimento do sistema dividida em 2 células referentes às áreas de processamento de **voz** e **outras**. Para cada célula, tem-se 4 (quatro) arquivos, os quais são discutidos na seção 5.4.1 do capítulo 5.

Na fig. 4.2, a árvore de decisão é mostrada de forma particionada.

A numeração existente na referida árvore tem a função de auxiliar na identificação da interligaçao das ramificações da

mesma.

Para as informações pertinentes à árvore de decisão, foram utilizadas termos de modo sintético, sendo apresentado os significados dos mesmos no quadro a seguir.

QUADRO DE QUALIFICAÇÃO DA ARVORE DE CONHECIMENTO

área - área de aplicação;

voz - área de processamento de sinais de voz

áudio - área de processamento de sinais de áudio

bioméd - área de processamento de sinais biomédicos

comunic - área de processmento de sinais comunicações

estrutura - tipo de estrutura a ser usada para o filtro

filtro - tipo de filtro

p_baixa - passa baixa

p_alta - passa alta

p_faixa - passa faixa

r_faixa - rejeita faixa

m_faixa - multi faixa

difer - diferenciador

t_hilb - transformador de Hilbert

p_tudo - passa tudo

tipo_aprox - tipo de aproximação;

butt - aproximação de Butterworth

cheb1 - aproximação Chebyshev tipo 1

cheb2 - aproximação Chebyshev tipo 2

elip - aproximação elíptica

programa(fir) - programa tipo fir

programa(iir_bw) - programa iir,aproximação Butterworth

programa(iir_c1) - programa iir,aproximação Chebyshev,tipo1

programa(iir_c2) - programa iir,aproximação Chebyshev,tipo2

programa(iir_el) - programa iir,aproximação elíptico

ordem - ordem do filtro (N)

ord1 - valor de N, $1 < N \leq 11$

ord2 - valor de N, $11 < N \leq 31$

ord3 - valor de N, $31 < N \leq 41$

ord4 - valor de N, $41 < N \leq 51$

ord5 - valor de N, $51 < N \leq 128$

largura - largura normalizada da faixa de transição (L)

larg1 - $0.00 < L < 0.03$

larg2 - $0.03 \leq L < 0.05$

larg3 - $0.05 \leq L < 0.065$

larg4 - $0.065 \leq L$

aten - valor de atenuação na faixa de rejeição (A)

att1 - 0 dB $< A <$ 20 dB

att2 - 20 dB $\leq A <$ 30 dB

att3 - 30 dB $\leq A <$ 44 dB

att4 - 44 dB $\leq A$

FIR1 - solução programa fir1

HAMM - solução programa fir2, utilizando a janela de Hamming

GHAM -solução programa fir2, utilizando a janela generalizada de Hamming

HANN - solução programa fir2, utilizando a janela de Hanning

KAIS - solução programa fir2, utilizando a janela de Kaiser

CHEB - solução programa fir2, utilizando a janela de Chebyshev

FIR4 - solução programa fir4

IIR12BW_LIN - solução utilizando os programas iir1 com aproximação Butterworth e iir2 com fase linear

IIR12BW_NLIN - solução utilizando os programas iir1 com aproximação Butterworth e iir2 sem fase linear

IIR124BW_LIN - solução utilizando os programas iir1 com aproximação Butterworth, iir2 com fase linear e iir4

IIR124BW_NLIN - solução utilizando os programas iir1 com aproximação Butterworth, iir2 sem fase linear e iir4

IIR12C1_LIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 1) e iir2 com fase linear

IIR12C1_NLIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo1) e iir2 sem fase linear

IIR124C1_LIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo1), iir2 com fase linear e iir4

IIR124C1_NLIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 1), iir2 sem fase linear e iir4

IIR12C2_LIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 2) e iir2 com fase linear

IIR12C2_NLIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 2) e iir2 sem fase linear

IIR124C2_LIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 2), iir2 com fase linear e iir4

IIR124C2_NLIN - solução utilizando os programas iir1 com aproximação Chebyshev (tipo 2), iir2 sem fase linear e iir4

IIR12EL_LIN - solução utilizando os programas iir1 com

aproximação elíptica e iir2 com fase linear

IIR12EL_NLIN - solução utilizando os programas iir1 com aproximação elíptica e iir2 sem fase linear

IIR124EL_LIN - solução utilizando os programas iir1 com aproximação elíptica, iir2 com fase linear e iir4

IIR124EL_NLIN - solução utilizando os programas iir1 com aproximação elíptica, iir2 sem fase linear e iir4

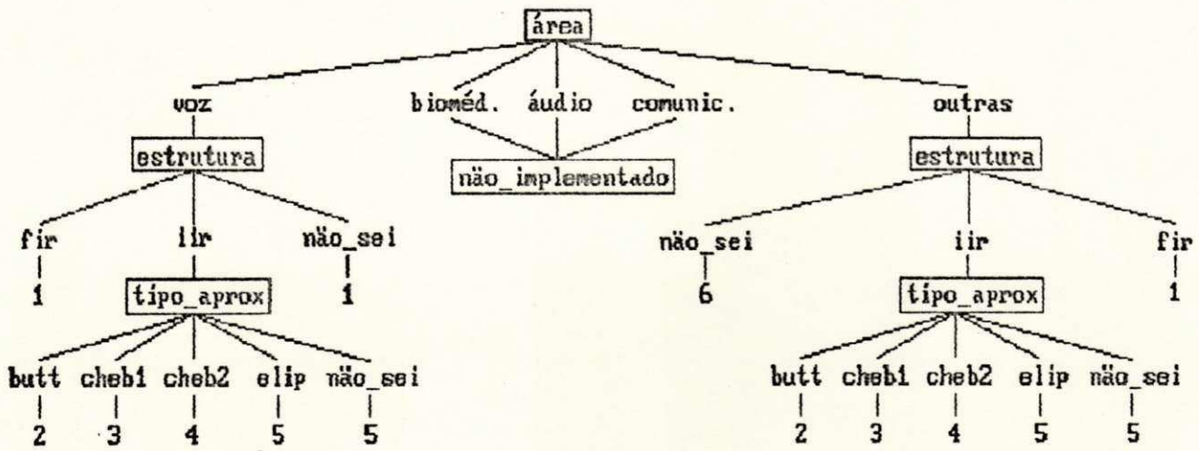

Fig. 4.2.a - Arvore de decisão, identificando a área, estrutura e tipo de aproximação.

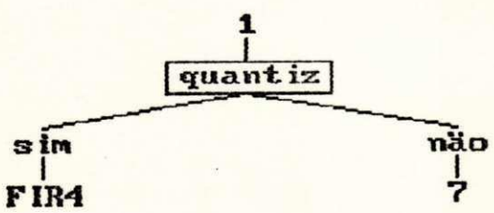

Fig. 4.2.b - identificação da opção de quantização.

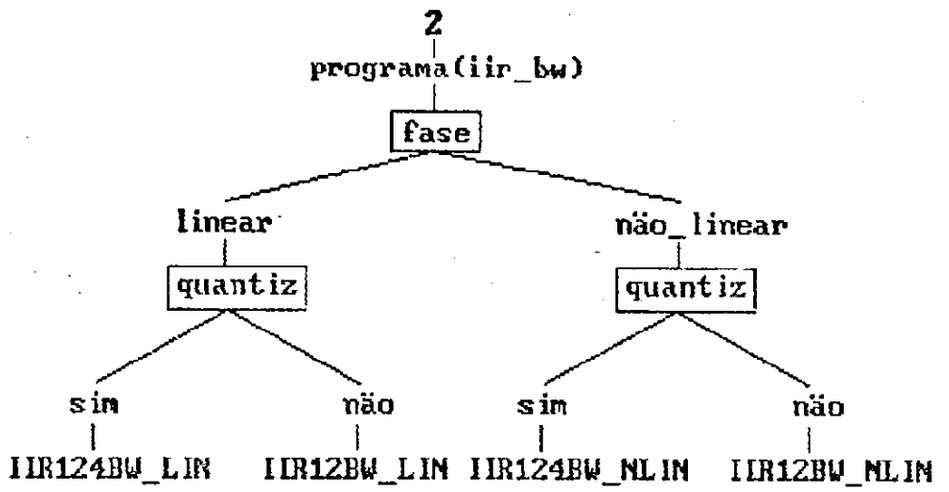

Fig. 4.2.c - Decisão para programa iir com aproximação de Butterworth.

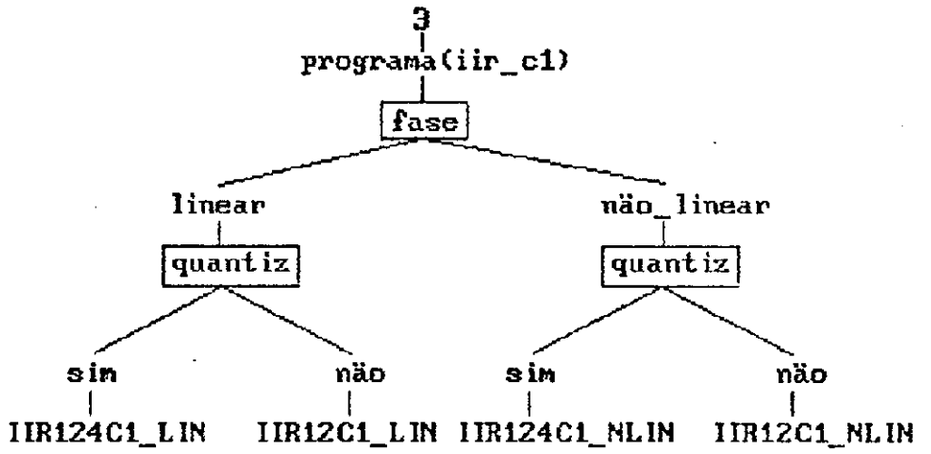

Fig. 4.2.d - Decisão para programa iir com aproximação Chebyshev, tipo 1.

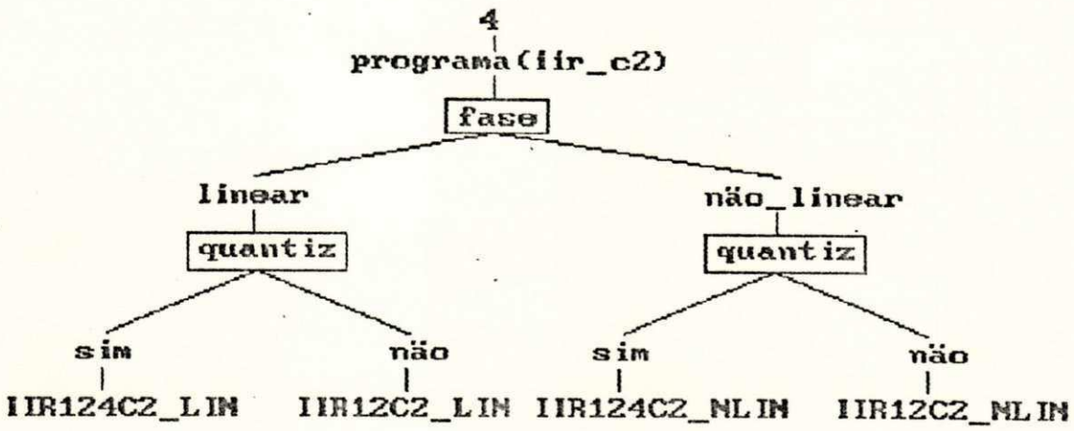

Fig. 4.2.e - Decisão para programa iir com aproximação Chebyshev, tipo 2.

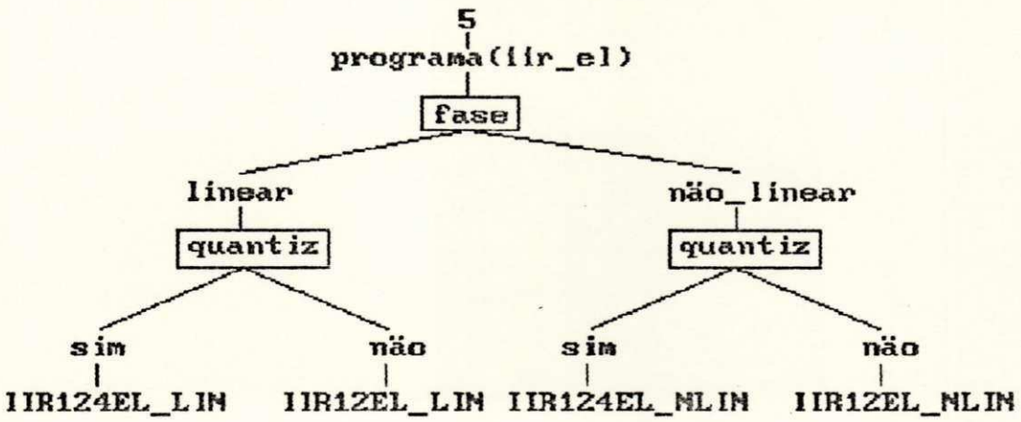

Fig. 4.2.f - Decisão para programa iir com aproximação elíptica.

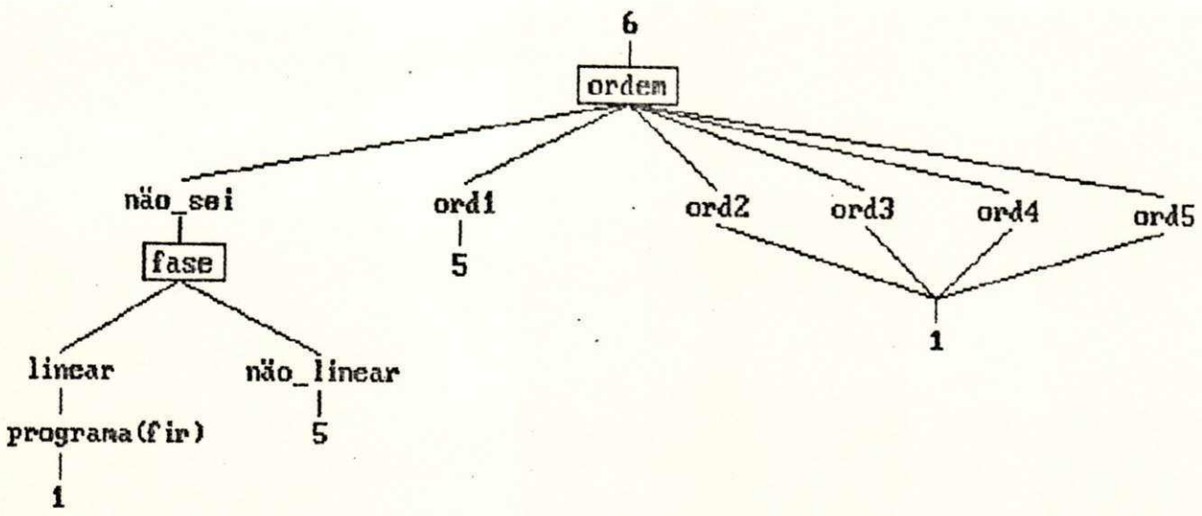

Fig. 4.2.g - Classificação da ordem do filtro dentro de uma das 5 (cinco) faixas existente nesta proposta.

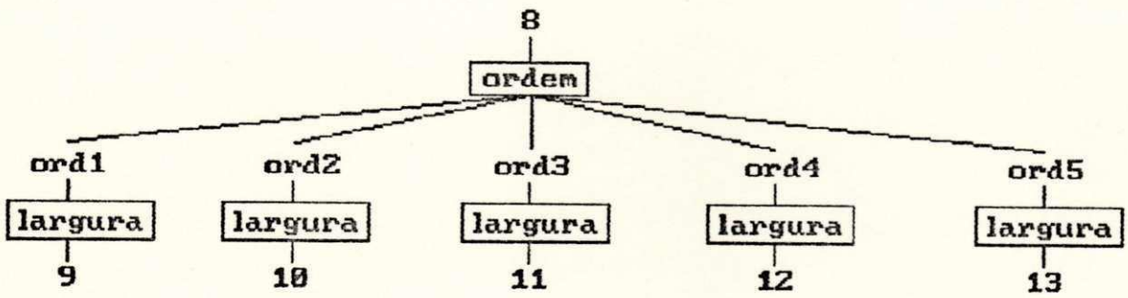

Fig. 4.2.h - identificação da largura após classificação da ordem do filtro.

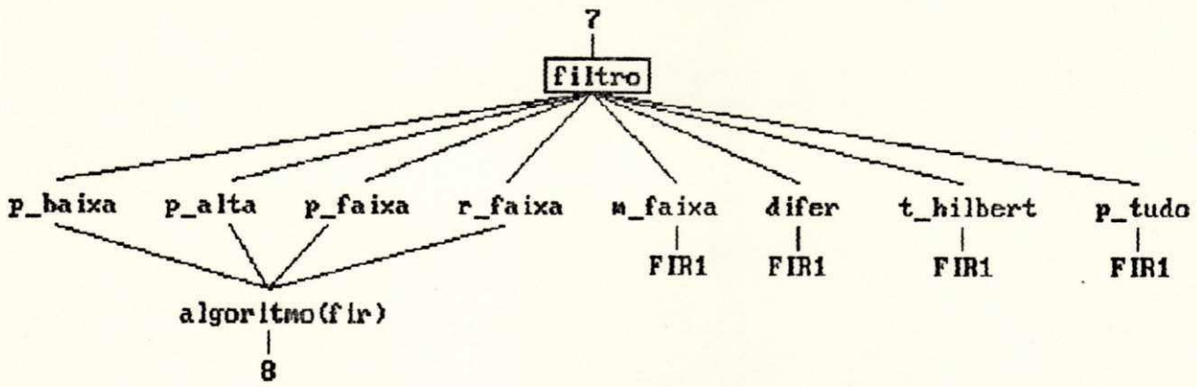

Fig. 4.2.i - Identificação do tipo de filtro.

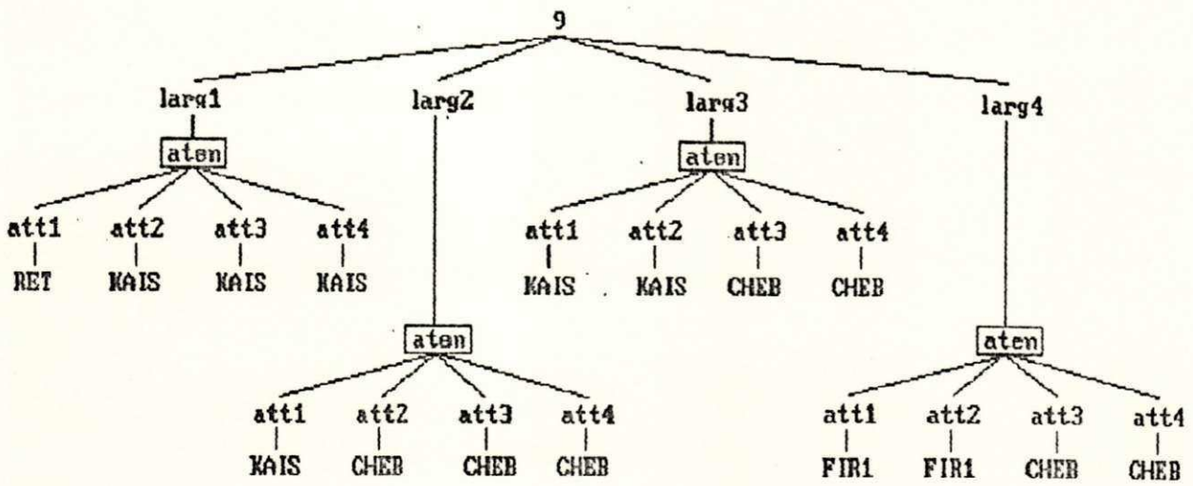

Fig. 4.2.j - Identificação da solução FIR em função da faixa do valor de atenuação, ordem e da largura da faixa de transição (ord1).

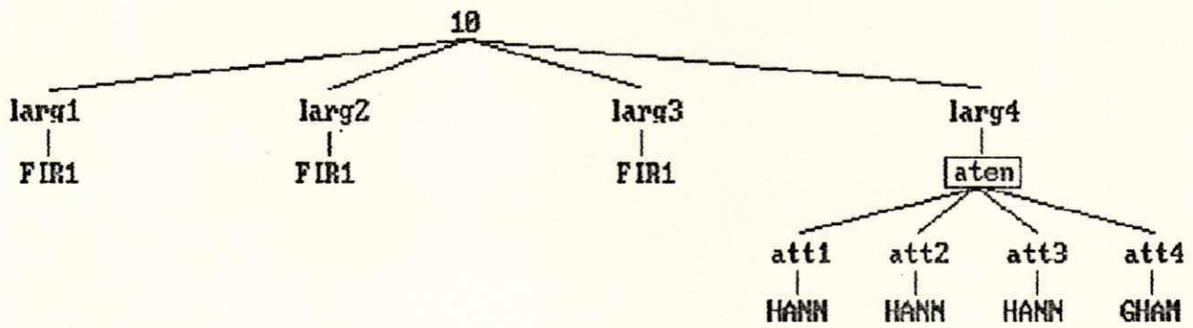

Fig. 4.2.k - Identificação da solução FIR em função da faixa do valor de atenuação, ordem e da largura da faixa de transição (ord2).

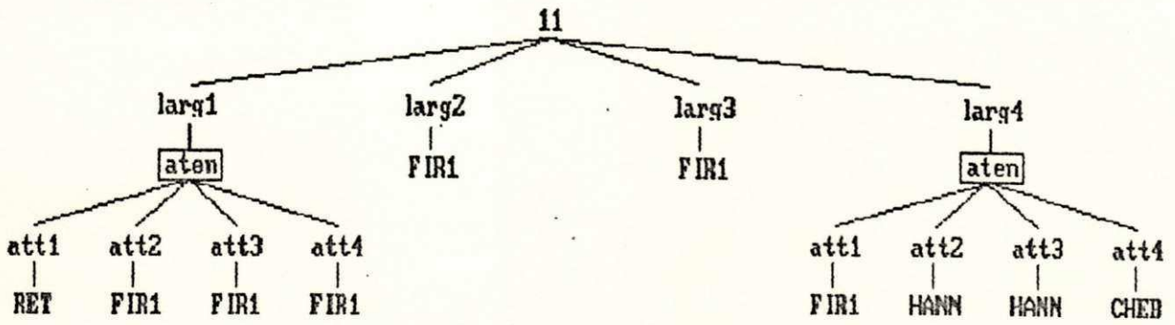

Fig. 4.2.l - Identificação da solução FIR em função da faixa do valor de atenuação, ordem e da largura da faixa de transição (ord3).

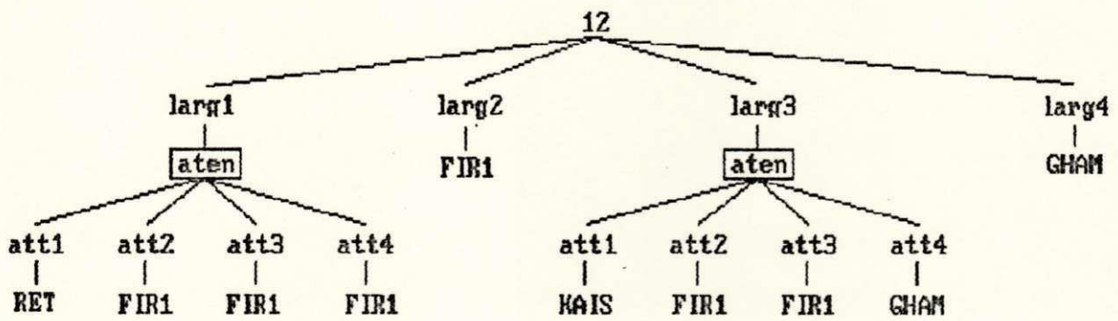

Fig. 4.2.m - Identificação da solução FIR em função da faixa do valor de atenuação, ordem e da largura da faixa de transição (ord4).

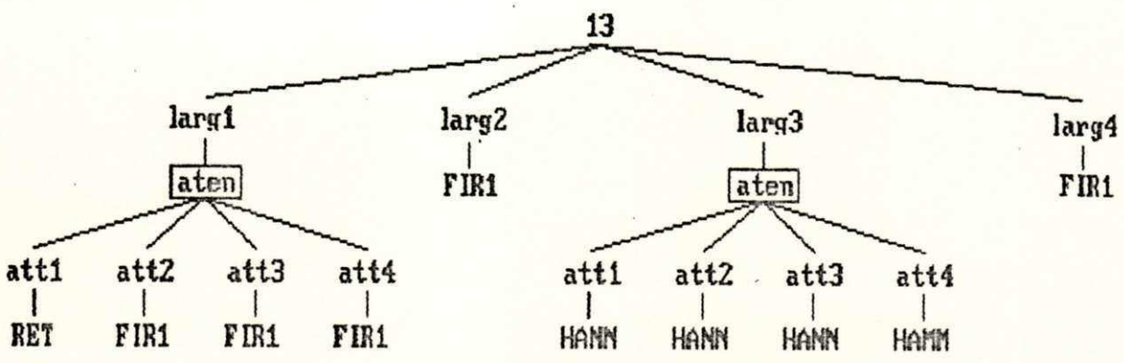

Fig. 4.2.n - Identificação da solução FIR em função da faixa do valor de atenuação, ordem e da largura da faixa de transição (ord5).

4.4. Resumo

Neste capítulo foi apresentada a metodologia usada para modelar a base de conhecimento, bem como foi justificado o uso de regras e apresentada a árvore de conhecimento.

Capítulo 5

Unidade Central - Estrutura para Seleção de Algoritmos para Projetos de Filtros Digitais

5.1. Introdução

A proposta desta unidade é a de voluntariar junto ao usuário um conjunto de fatos, os quais se constituem nos fatos iniciais, para em seguida ativar o mecanismo de inferência que interage com a base de conhecimento a fim de identificar novos fatos e/ou diagnosticar uma solução para o problema em questão.

O problema proposto é o de identificar um ou mais algoritmo(s) a partir de um conjunto de informações fornecidas por um usuário, objetivando o projeto de um filtro digital.

A seguir, é apresentada a arquitetura da unidade central.

5.2. Arquitetura da Unidade Central

É mostrada na fig. 5.1 a arquitetura usada nesta unidade.

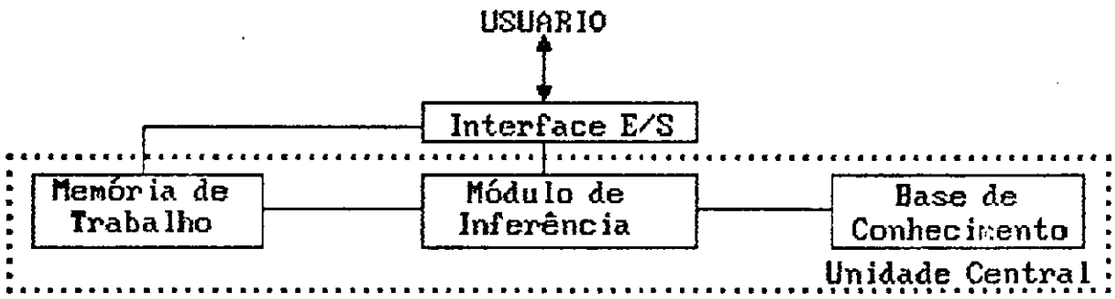

Fig. 5.1 - Arquitetura da Unidade Central

A interface de E/S realiza o voluntariamento junto ao

usuário obtendo um conjunto de fatos, chamados fatos iniciais, os quais são armazenados na memória de trabalho.

Ao término da voluntariamento, o módulo de inferência é ativado; este, por sua vez, armazena na memória de trabalho a(s) célula(s) da base de conhecimento relacionada ao problema em questão, iniciando o processo de inferência.

Após encontrada uma solução, é permitido ao usuário verificar as regras usadas para solução do problema.

A fig. 5.2 mostra um diagrama deste processo [ARAK90].

O processo de voluntariamento e diagnóstico sumarizado na fig. 5.2 é detalhado nos sub-ítens seguintes apresentados neste capítulo.

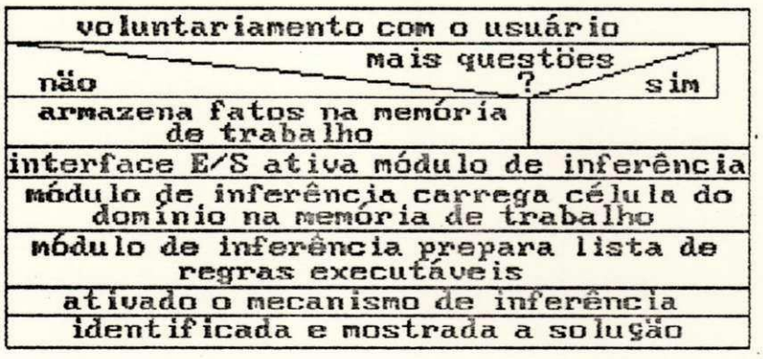

Fig. 5.2 - Diagrama Nassi Schneiderman do processo de voluntariamento e diagnóstico.

5.3. Interface para Voluntariamento

Com o objetivo de efetuar testes na base de conhecimento desenvolvida, foi implementada uma interface de E/S, interagindo com o módulo de inferência e com o usuário.

Para tanto, utilizou-se a linguagem Arity/Prolog [ARIT88] e [ARIT87], facilitando essa interação.

A finalidade desta interface é a de voluntariar junto ao usuário um conjunto de fatos iniciais, os quais são armazenados na memória de trabalho, possibilitando o módulo de inferência ter acesso a esses fatos para que seja feito o diagnóstico, encontrando-se a solução para o problema proposto.

A interface possui incorporado um módulo auxiliar de informação (**help-on-line**) a fim de auxiliar o usuário durante o processo de voluntariamento.

A seqüência de figs. 5.3 mostram as questões 1, 2, 3, 4, 5, 6 e algumas questões complementares feitas ao usuário no processo de voluntariamento.

O programa que implementa este processo se encontra no apêndice B1.

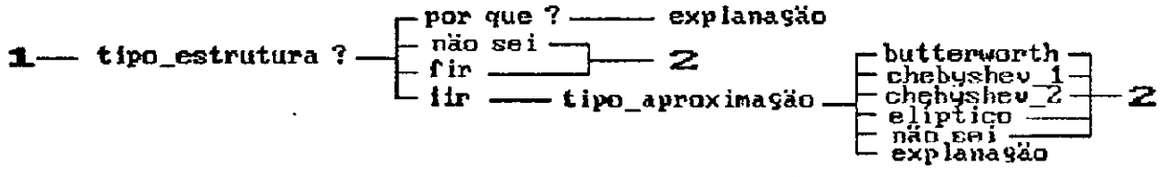

Fig. 5.3.a - Identificação do tipo de estrutura (e tipo de aproximação, se iir).

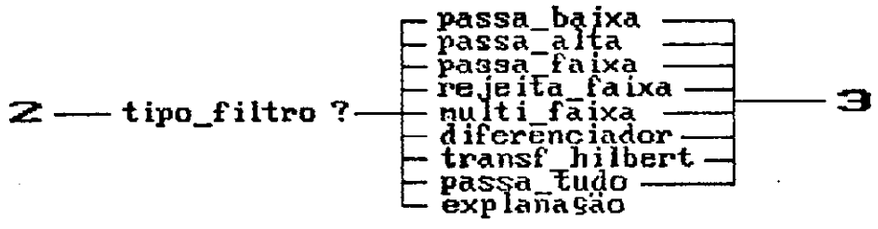

Fig. 5.3.b - Identificação do tipo de filtro.

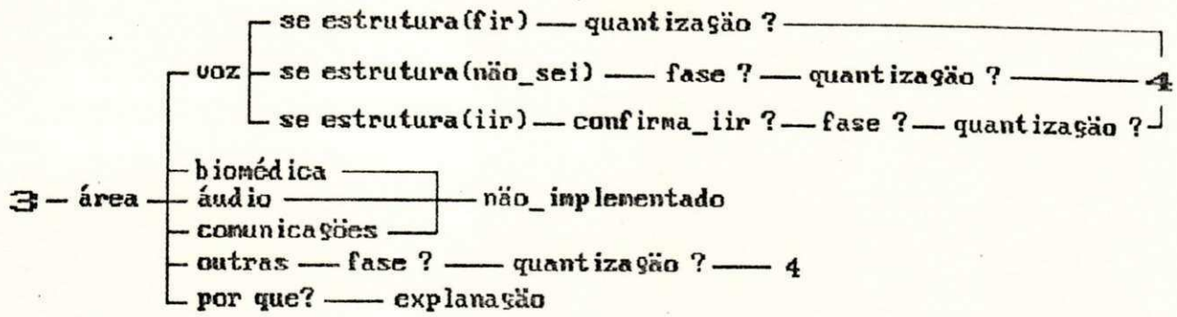

Fig. 5.3.c - Identificação da área de aplicação, tipo de fase (linear/não_linear) e se é desejada a quantização dos coeficientes.

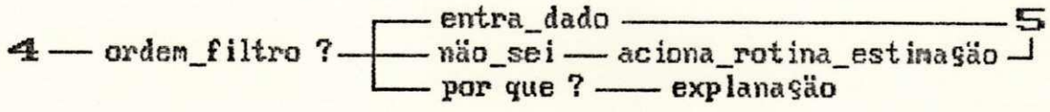

Fig. 5.3.d - Identificação da ordem desejada do filtro.

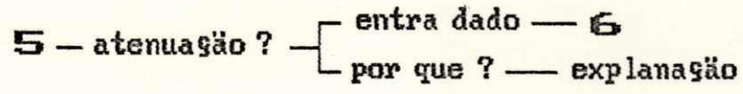

Fig. 5.3.e - Identificação da atenuação desejada na faixa de rejeição.

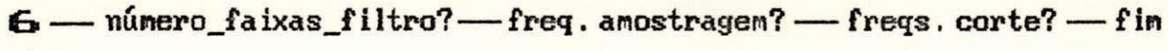

Fig. 5.3.f - Identificação do número de faixas de freqüências, da freqüência de amostragem e do conjunto de freqüências de corte que definem o filtro desejado.

5.4. Unidade Central

Nesta seção, são vistos os blocos que compõem a Unidade Central, isto é, a base de conhecimento, a memória de trabalho e o módulo de inferência.

5.4.1. Base de Conhecimento

A base de conhecimento está estruturada de forma celular, cada célula contendo informações pertinentes a uma das áreas de aplicação de PDS e informações sobre os algoritmos de projetos de filtros digitais.

Cada célula é identificada por uma área de aplicação, reunindo as informações mencionadas acima.

Tem-se mostrado na fig. 5.4, o caráter celular desta base de conhecimento.

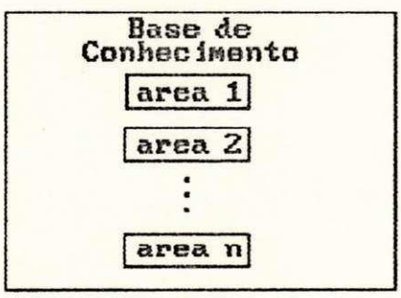

Fig. 5.4 - Base de conhecimento celular

Tendo sido representado o conhecimento através do uso de regras, conforme descrito capítulo 4, identificou-se uma ferramenta que facilitasse a edição da base de conhecimento na forma de predicados Prolog. Para este fim, utilizou-se um editor de regras [PERK90], o qual possibilitou não apenas a edição do

arquivo [*.reg], como de outros arquivos mencionados a seguir, os quais interagem com o módulo de inferência.

Cada célula se apresenta composta de quatro arquivos: [*.reg], [*.con], [*.inf] e [*.pri]. Estes arquivos são caracterizados abaixo.

[*.reg] - Arquivo de regras, contendo um conjunto de regras, na forma de predicados Prolog, às quais reúnem informações de PDS e dos algoritmos de projetos de filtros digitais. As regras se apresentam no seguinte formato:

regra(número da regra, se(antecedente) então(consequente)).

[*.con] - arquivo contagem, indicando a quantidade de condições existentes no antecedente da regra, bem como o conectivo usado. Para esta base, tem-se o seguinte formato:

con(C1, C2, N, C) , onde:

N - número da regra

C - tipo de conectivo: se **C** = 1 -> conectivo OU
se **C** = 0 -> conectivo **E**

C1, C2 - quantidade de condições no antecedente da regra

[*.inf] - arquivo influência, indicando os fatos que influenciam em uma regra. Apresenta-se no seguinte formato:

influ(fato,N) , onde **N** - número da regra.

[*.pri] - arquivo prioridade, indicando a prioridade de cada regra. Mostra-se, a seguir, o formato usado:

prior(P, N) , onde **P** - prioridade da regra e

N - número da regra.

Os arquivos [*.reg], [*.con], [*.inf] e [*.pri] que compõem a base de conhecimento se encontram no apêndice A.

5.4.2. Memória de Trabalho

A memória de trabalho reúne os fatos informados pelo usuário quando do processo de voluntariamento e a célula correspondente ao problema em questão. A fig. 5.5 mostra, de forma simplificada, o conteúdo da memória de trabalho.

Fig. 5.5 - Conteúdo da memória de trabalho.

O módulo de inferência interage com a memória de trabalho, armazenando nesta os arquivos [*.reg], [*.con], [*.inf] e [*.pri] da célula do problema em questão.

5.4.3. Módulo de Inferência

Nesta seção é apresentado o funcionamento do módulo de inferência e sua interação com a base de conhecimento, memória de trabalho e interface de E/S.

O módulo de inferência é o elemento do sistema especialista que controla e avalia o conteúdo da base de conhecimento, direcionando o processo de inferência.

O mecanismo de inferência utilizado para integrar a arquitetura da unidade central, mostrada na fig. 5.1, é baseado no desenvolvido por Perkusich [PERK90], tendo sido adaptado às necessidades deste trabalho.

Como pode ser visto na fig. 5.6, o módulo de inferência é composto de 5 partes: **Supervisor**, **Escalonador**, **Acionador**, **Diagnosticador** e **Explanador**.

O **supervisor** armazena a célula (conjunto de 4 arquivos) referente ao domínio em questão na memória de trabalho e ativa o escalonador.

O **escalonador** é ativado pelo supervisor que lhe apresenta o conjunto de fatos (informados pelo usuário) para que sejam identificadas as regras influenciadas por este fatos, ou seja, as **regras executáveis**.

O **acionador** recebe a lista de regras preparadas pelo escalonador a fim de que seja feito o encadeamento progressivo a partir dos fatos informados pelo usuário com o objetivo de identificar novos fatos ou achar o diagnóstico. O acionador é o elemento responsável pelo mecanismo de inferência.

O **diagnosticador** lista a solução encontrada buscando o predicado **diagnóstico(solução)**.

O **explanador** fornece os fatos e regras usadas para encontrar a solução.

Fig. 5.6 - Partes componentes do módulo de inferência.

Como foi mostrado na fig. 5.1, o módulo de inferência interage com a interface de E/S, a memória de trabalho e a base de conhecimento.

Após o processo de voluntariamento, feito pela interface de E/S, o módulo de inferência é ativado para que armazene na memória de trabalho a célula correspondente ao domínio em questão.

Em seguida, o supervisor ativa o escalonador a fim de que este identifique as regras executáveis, ou seja, as regras influenciadas pelos fatos informados pelo usuário quando do voluntariamento.

Após serem identificadas as regras executáveis, este conjunto de regras é distribuído numa lista atendendo às prioridades constantes no arquivo [*.pri].

Estando esta lista de regras executáveis preparada, o acionador é ativado e então tem início o encadeamento progressivo, ou seja, parte-se das premissas das regras (fatos informados pelo usuário)e da lista de regras executáveis em ordem descrescente de prioridade.

Com a execução dessas regras, novos fatos poderão ser adicionados à memória de trabalho, fazendo com que o escalonador identifique novas regras executáveis.

Quando for encontrado o diagnóstico, o diagnosticador é chamado e então ele identifica a solução através do predicado **diagnóstico(solução)** e lista o resultado.

Esta seqüência de procedimentos se encontra na fig. 5.7.

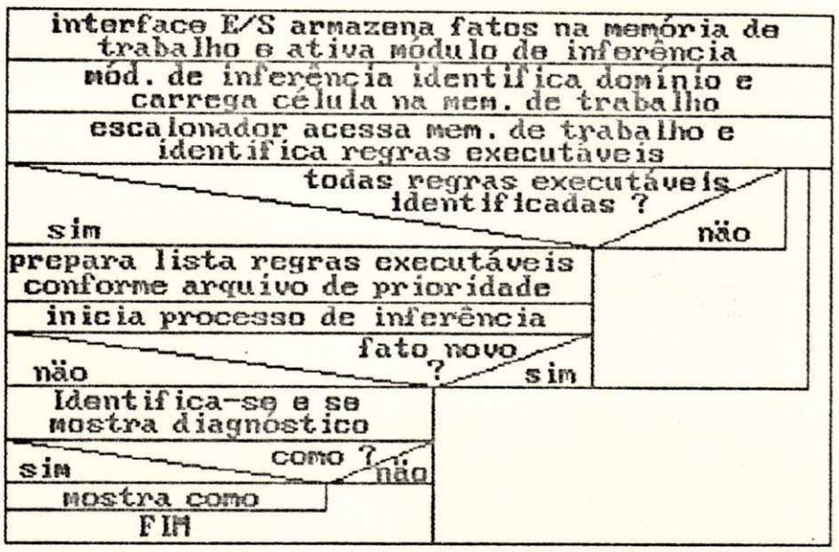

Fig. 5.7 - Diagrama Nassi Schneiderman do mecanismo de inferência.

No apêndice B2, tem-se o programa que implementa esse mecanismo.

5.5. Acoplamento da Unidade Central a um Sistema Especialista

Após o desenvolvimento da unidade central e submissão da mesma a testes, procurou-se integrar esta unidade a um sistema especialista. Para tanto, fez-se o a acoplamento desta unidade ao sistema SIPREX.

Este acoplamento se deu com a substituição da interface de E/S (um conjunto de menus que realizavam o processo de voluntariamento) pela interface do SIPREX.

Dessa forma, a unidade central lê um arquivo **base.dat**, que contém o conjunto de fatos referentes ao processo de

voluntariamento.

Após a leitura desse arquivo, o mecanismo de inferência da unidade central é ativado fornecendo uma solução e sua justificativa, isto é, é realizada a escrita no arquivo **base.out** da solução encontrada.

Essa seqüência de procedimentos é supervisionada pelo módulo de controle e está sumarizada na fig. 5.8.

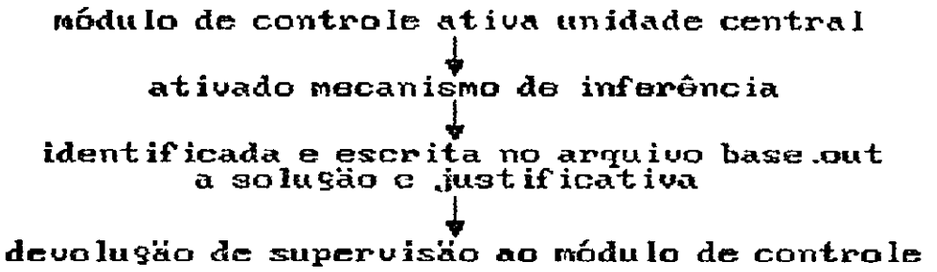

Fig. 5.8 - Controle da unidade central.

5.6. Resumo

Neste capítulo foi apresentada e detalhada a unidade central, bem como seu acoplamento ao sistema SIPREX.

Capítulo 6

Conclusões

Os objetivos traçados para este trabalho foram alcançados. Fez-se o estudo sobre Processamento Digital de Sinais e o sobre o comportamento dos algoritmos para projeto de filtros digitais. Este estudo permitiu a modelagem da base de conhecimento para um sistema especialista. Posteriormente, fez-se a implementação de um módulo de voluntariamento e se utilizou de um mecanismo de inferência, constituindo assim uma unidade de voluntariamento e diagnóstico, denominada neste trabalho de unidade central.

Alcançada a modelagem da base de conhecimento, teve-se a necessidade de se realizar um processo de aquisição e representação de conhecimento, orientando-se pela seqüência de procedimentos apresentada no capítulo 4.

Ainda, fez-se uso de uma ferramenta para a edição da base de conhecimento [PERK90] desenvolvida na UFPb, a qual minimizou o tempo de realização deste trabalho. Aliado a esse fato, a edição da base de conhecimento foi direcionado ao uso do mecanismo de inferência discutido no capítulo 5.

Também, com a utilização desse mecanismo de inferência, permitiu-se realizar testes a fim de validar a base de conhecimento, integrando-a por fim ao sistema especialista (SIPREX), conforme abordado na seção 5.5 do capítulo 5.

Quanto ao estudo feito sobre os algoritmos para projeto de filtros digitais, processamento digital de sinais (PDS) e áreas de aplicação de processamento digital de sinais, verificou-se

haver pequeno conteúdo de informações associando os referidos algoritmos às áreas de aplicação. Daí, ter-se um volume maior de informações sobre os algoritmos na base de conhecimento.

No que diz respeito a análise feita sobre o comportamento destes algoritmos, tem-se que esta se limitou à uma análise numérica e gráfica dos resultados fornecidos face a um conjunto de problemas-exemplo.

Para refinamento maior das informações sobre os algoritmos, é proposto implementar os filtros obtidos, avaliando-se seus desempenhos.

Referente às consultas feitas aos especialistas, identificou-se maior dificuldade deles expressarem mais conhecimentos relacionados à aplicação destes algoritmos em suas áreas de concentração por eles não terem familiaridade com os mesmos.

Como extensão deste trabalho, pode-se utilizar a metodologia, isto é, o conjunto de procedimentos usados neste trabalho quando do processo de aquisição e representação de conhecimento, permitindo assim estruturar outras células, correspondentes a outras áreas de aplicação, bem como no desenvolvimento de outros trabalhos, cuja finalidade seja o desenvolvimento de sistemas especialistas.

REFERENCIAS BIBLIOGRAFICAS:

[AKIN88] O. Akin, "Expertise of the Architect" in Expert Systems for Engineering Design, Ed., 1988;

[ARAK90] R. Arakaki et Al., Fundamentos de Programação C - Técnicas e Aplicações, LTC Ed., Rio de Janeiro, 1990;

[ARAR89] G. Araribóia, Inteligência Artificial - Um Curso Prático, LTC Ed., 1989;

[ARIT86] Arity Corporation, The Arity/Expert Development Package, Massachusetts, 1986;

[ARIT87] Arity Corporation, Using the Arity/Prolog Compiler and Interpreter, Massachusetts, 1987;

[ARIT88] Arity Corporation, The Arity/Prolog Language Reference Manual, Massachusetts, 1988;

[BELL86] M. G. Bellanger, New Applications of Digital Signal Processing in Communications, IEEE ASSP Magazine, Vol. 3, No. 3, July, 1986, pp. 06 - 11;

[BERK85] P. J. Berkhout and L. D. J. Eggermont, Digital Audio Systems, IEEE ASSP Magazine, Vol. 2., No. 4, October, 1985, pp. 45 - 67;

[BLES78] B. Blesser and J. M. Kates, "Digital Processing in Audio Signals" in Applications of Digital Signal Processing, Ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1978;

[BLOO85] P. J. Bloom, High - Quality Digital Audio in Entertainment Industry, IEEE ASSP Magazine, Vol. 2, No. 4, October, 1985, pp. 02 - 25;

[CARV91] J. M. de Carvalho, M. C. Pequeno, A. M. Silva Filho, M. T. Hattori, SIPREX: An Expert System for Digital Filter Design, 13th. GRETSI Symposium on Signal and Image Processing, Juan-les-Pins, França, September, 1991;

[CASA87] M. A. Casanova, F. A. C. Giorno e A. L. Furtado, Ed., Programação em Lógica e a Linguagem Prolog, Edgard Blucher Ltda, Rio de Janeiro, 1987;

[CAST79] K. R. Castleman, Digital Image Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979;

[COOK86] N. M. Cooke and J. E. McDonald, A Formal Methodology for Acquiring and Representing Expert Knowledge, Proceedings of the IEEE, Vol. 74, No. 10, October, 1986, pp. 1422 - 1430;

[DECZ72] A. G. Deczky, Synthesis of Recursive Digital Filters Using the Minimum p-Error Criterion, IEEE Trans. on Audio and Electroacoustics, Vol. AU-20, No. 4, pp. 257-263, october, 1972;

[DECZ79] A. G. Deczky, DSP Committee, IEEE ASSP Society Ed., Program for Minimum-p Synthesis of Recursive Digital Filters, Programs for DSP, IEEE Press, New York, 1979;

[DEHN79] G. F. Dehner, DSP Committee, IEEE ASSP Society Ed.,

Program for the Design of Recursive Digital Filters, Programs for DSP, IEEE Press, New York, 1979;

[DOLA79] M. T. Dolan and J. F. Kaiser, DSP Committee, IEEE ASSP Society Ed., An Optimization Program for the Design of Digital Filter Transfer Functions, Programs for DSP, IEEE Press, New York, 1979;

[FREE78] S. L. Freeney, J. F. Kaiser and H. S. McDonald, "Some Applications of DSP in Telecommunications" in Applications of Digital Signal Processing, Ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1978;

[GOME89] F. A. C. GOMES, APREND: Um Sistema de Aquisição Automática de Conhecimento a Partir de Exemplos, Dissertação de Mestrado, Campina Grande, 1989;

[HEUT79] U. Heute, DSP Committee, IEEE ASSP Society Ed., A Subroutine for Finite Wordlenth FIR Filter Design, Programs for DSP, IEEE Press, New York, 1979;

[IEEE79] DSP Committee, IEEE ASSP Society, Programs for DSP, IEEE Press, New York, 1979;

[KAIS79] J. F. Kaiser, DSP Committee, IEEE ASSP Society Ed., Design Subroutine for Simmetric FIR Low Pass Digital Filters with Maximally-Flat Pass and Stop Bands, Programs for DSP, IEEE Press, New York, 1979;

[KING86] M. King and M. Rosner, The Special Issue on Knowledge Representation, Proceedings of the IEEE, Vol. 74, No.

10, October, 1986, pp. 1299 - 1303;

[LATH79] B. P. Lathi, Sistemas de Comunicação, Guanabara Dois, Rio de Janeiro, 1979;

[LOVR86] Al Lovrich and Ray Simar Jr., Implementation of FIR/IIR Filters with the TMS32010/TMS32020, Applications of Digital Signal Processing with TMS320, Texas Instruments, 1986;

[McCL79] J. H. McClellan, T. W. Parks and L. R. Rabiner, DSP Committee, IEEE ASSP Society Ed.,FIR Linear Phase Filter Design Program, Programs for DSP, IEEE Press, New York, 1979;

[OPPE75] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing, Ed., Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1975;

[OPPE78] A. V. Oppenheim, "Digital Processing of Speech" in Applications of Digital Signal Processing, Ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1978;

[OPPE83] A. V. Oppenheim, A. S. Willsky and I. T. Young, Signals and Systems, Ed., Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983;

[PEQU88] M. C. Pequeno, EXBITAN - Um Sistema Especialista Numérico, XI CNMAC, Ouro Preto, 1988;

[PEQU89] M. C. Pequeno, FFTEX - Um Sistema Especialista para o Cálculo de Transformadas Rápidas de Fourier em Uma e

Duas Dimensões, II Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, Aguas de Lindóia, 1989;

[PERK90] M. L. B. Perkusich, Contribuição para a Construção de Sistemas Baseados em Conhecimento Voltados para o Conhecimento Profundo, Dissertação de Mestrado, Campina Grande, 1990;

[RABI74] L. R. Rabiner, J. F. Kaiser, O. Herrmann and M. T. Dolan, Some Comparisons Between FIR and IIR Digital Filters, The Bell System Technical Journal, Vol. 53, No. 2, February, 1974;

[RABI75] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975;

[RABI79] L. R. Rabiner, C. A. McGonegal and D. Paul, DSP Committee, IEEE ASSP Society Ed., FIR Windowed Filter Design Program-Window, Programs for DSP, IEEE Press, New York, 1979;

[RYCH88] M. D. Rychener, Research in Expert Systems for Engineering Design in Expert Systems for Engineering Design, Ed., 1988;

[STEI79] K. Steiglitz and B. D. Ladendorf, DSP Committee, IEEE ASSP Society Ed., A Program for Designing Finite Word-Lenth IIR Digital Filters, Programs for DSP. IEEE Press, New York, 1979;

[WATE86] D. A. Waterman, A Guide to Expert Systems, Addison-Wesley Publishing Co., 1986;

[WEBS78] J. G. Webster et Al., Medical Instrumentation - Application and Design, Ed., Houghton Mifflin Co., 1978;

[WILL88] A. B. Williams and F. J. Taylor, Electronic Filter Design Handbook, Mcgraw-Hill Publishing Co., New York, 1988;

[WOOD86] W. A. Woods, Important Issues in Knowledge Representation, Proceedings of the IEEE, Vol. 74, No. 10, October, 1986, pp. 1322 - 1334;

APENDICE A1

ARQUIVO VOZ.REG

```
regra( 1, se ( area(voz) , estrutura(filtro_fir) , quantiz(sim) )
entao (  guarde_d(fir4))).
regra( 2, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(passa_baixa) )
entao (  guarde_f(algoritmo(fir)))).
regra( 3, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(passa_alta) )
entao (  guarde_f(algoritmo(fir)))).
regra( 4, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(passa_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra( 5, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(rejeita_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra( 6, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(multi_faixa) )
entao (  guarde_d(fir1))).
regra( 7, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(diferenciador) )
entao (  guarde_d(fir1))).
regra( 8, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(transf_hilbert) )
entao (  guarde_d(fir1))).
regra( 9, se ( area(voz) , estrutura(filtro_fir) , quantiz(nao) ,
filtro(passa_tudo) )
entao (  guarde_d(fir1))).
regra( 10, se ( area(voz) , estrutura(nao_sei) , quantiz(sim) )
entao (  guarde_d(fir4))).
regra(  11, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(passa_baixa) )
entao (  guarde_f(algoritmo(fir)))).
regra(  12, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(passa_alta) )
entao (  guarde_f(algoritmo(fir)))).
regra(  13, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(passa_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra(  14, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(rejeita_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra(  15, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(multi_faixa) )
entao (  guarde_d(fir1))).
regra(  16, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(diferenciador) )
entao (  guarde_d(fir1))).
regra(  17, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(transf_hilbert) )
entao (  guarde_d(fir1))).
```

```
regra(  18, se ( area(voz) , estrutura(nao_sei) , quantiz(nao)  ,
filtro(passa_tudo) )
entao (  guarde_d(fir1))).
regra(   19,   se   (   area(voz)   ,   estrutura(filtro_iir)   ,
tipo_aprox(butterworth) )
entao (  guarde_f(programa(iir_bw)))).
regra(   20,   se   (   area(voz)   ,   estrutura(filtro_iir)   ,
tipo_aprox(chebyshev_1) )
entao (  guarde_f(programa(iir_c1)))).
regra(   21,   se   (   area(voz)   ,   estrutura(filtro_iir)   ,
tipo_aprox(chebyshev_2) )
entao (  guarde_f(programa(iir_c2)))).
regra(   22,   se   (   area(voz)   ,   estrutura(filtro_iir)   ,
tipo_aprox(eliptico) )
entao (  guarde_f(programa(iir_el)))).
regra(   23,   se   (   area(voz)   ,   estrutura(filtro_iir)   ,
tipo_aprox(nao_sei) )
entao (  guarde_f(programa(iir_el)))).
regra( 24, se ( programa(iir_bw) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124bw_lin))).
regra( 25, se ( programa(iir_bw) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12bw_lin))).
regra(   26,   se   (   programa(iir_bw)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124bw_nlin))).
regra(   27,   se   (   programa(iir_bw)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12bw_nlin))).
regra( 28, se ( programa(iir_c1) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124c1_lin))).
regra( 29, se ( programa(iir_c1) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12c1_lin))).
regra(   30,   se   (   programa(iir_c1)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124c1_nlin))).
regra(   31,   se   (   programa(iir_c1)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12c1_nlin))).
regra( 32, se ( programa(iir_c2) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124c2_lin))).
regra( 33, se ( programa(iir_c2) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12c2_lin))).
regra(   34,   se   (   programa(iir_c2)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124c2_nlin))).
regra(   35,   se   (   programa(iir_c2)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12c2_nlin))).
regra( 36, se ( programa(iir_el) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124el_lin))).
regra( 37, se ( programa(iir_el) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12el_lin))).
regra(   38,   se   (   programa(iir_el)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124el_nlin))).
```

```
regra(   39,   se  (  programa(iir_el)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12el_nlin))).
regra(  40, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  41, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(kaiser))).
regra(  42, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(kaiser))).
regra(  43, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(kaiser))).
regra(  44, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
regra(  45, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att2) )
entao (  guarde_d(chebyshev))).
regra(  46, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att3) )
entao (  guarde_d(chebyshev))).
regra(  47, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra(  48, se ( algoritmo(fir) , ordem(ord1) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
regra(  49, se ( algoritmo(fir) , ordem(ord1) , largura(larg3)  ,
atenuacao(att2) )
entao (  guarde_d(kaiser))).
regra(  50, se ( algoritmo(fir) , ordem(ord1) , largura(larg3)  ,
atenuacao(att3) )
entao (  guarde_d(chebyshev))).
regra(  51, se ( algoritmo(fir) , ordem(ord1) , largura(larg3)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra(  52, se ( algoritmo(fir) , ordem(ord1) , largura(larg4)  ,
atenuacao(att1) )
entao (  guarde_d(fir1))).
regra(  53, se ( algoritmo(fir) , ordem(ord1) , largura(larg4)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  54, se ( algoritmo(fir) , ordem(ord1) , largura(larg4)  ,
atenuacao(att3) )
entao (  guarde_d(chebyshev))).
regra(  55, se ( algoritmo(fir) , ordem(ord1) , largura(larg4)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra( 56, se ( algoritmo(fir) , ordem(ord2) , largura(larg1) )
entao (  guarde_d(fir1))).
regra( 57, se ( algoritmo(fir) , ordem(ord2) , largura(larg2) )
entao (  guarde_d(fir1))).
```

```
regra( 58, se ( algoritmo(fir) , ordem(ord2) , largura(larg3) )
entao (  guarde_d(fir1))).
regra(  59, se ( algoritmo(fir) , ordem(ord2) , largura(larg4)  ,
atenuacao(att1) )
entao (  guarde_d(hanning))).
regra(  60, se ( algoritmo(fir) , ordem(ord2) , largura(larg4)  ,
atenuacao(att2) )
entao (  guarde_d(hanning))).
regra(  61, se ( algoritmo(fir) , ordem(ord2) , largura(larg4)  ,
atenuacao(att3) )
entao (  guarde_d(hanning))).
regra(  62, se ( algoritmo(fir) , ordem(ord2) , largura(larg4)  ,
atenuacao(att4) )
entao (  guarde_d(gen_hamming))).
regra(  63, se ( algoritmo(fir) , ordem(ord3) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  64, se ( algoritmo(fir) , ordem(ord3) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  65, se ( algoritmo(fir) , ordem(ord3) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  66, se ( algoritmo(fir) , ordem(ord3) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(fir1))).
regra( 67, se ( algoritmo(fir) , ordem(ord3) , largura(larg2) )
entao (  guarde_d(fir1))).
regra( 68, se ( algoritmo(fir) , ordem(ord3) , largura(larg3) )
entao (  guarde_d(fir1))).
regra(  69, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att1) )
entao (  guarde_d(fir1))).
regra(  70, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att2) )
entao (  guarde_d(hanning))).
regra(  71, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att3) )
entao (  guarde_d(hanning))).
regra(  72, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra(  73, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  74, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  75, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  76, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(fir1))).
regra( 77, se ( algoritmo(fir) , ordem(ord4) , largura(larg2) )
```

```
entao (  guarde_d(fir1))).
regra(  78, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
regra(  79, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  80, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  81, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att4) )
entao (  guarde_d(gen_hamming))).
regra( 82, se ( algoritmo(fir) , ordem(ord4) , largura(larg4) )
entao (  guarde_d(gen_hamming))).
regra(  83, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  84, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  85, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  86, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(fir1))).
regra( 87, se ( algoritmo(fir) , ordem(ord5) , largura(larg2) )
entao (  guarde_d(fir1))).
regra(  88, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(hanning))).
regra(  89, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att2) )
entao (  guarde_d(hanning))).
regra(  90, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att3) )
entao (  guarde_d(hanning))).
regra(  91, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att4) )
entao (  guarde_d(hamming))).
regra( 92, se ( algoritmo(fir) , ordem(ord5) , largura(larg4) )
entao (  guarde_d(fir1))).
```

APENDICE A2

ARQUIVO VOZ.CON

```
contagem( 3 , 3 , 1 , 0 ).
contagem( 4 , 4 , 2 , 0 ).
contagem( 4 , 4 , 3 , 0 ).
contagem( 4 , 4 , 4 , 0 ).
contagem( 4 , 4 , 5 , 0 ).
contagem( 4 , 4 , 6 , 0 ).
contagem( 4 , 4 , 7 , 0 ).
contagem( 4 , 4 , 8 , 0 ).
contagem( 4 , 4 , 9 , 0 ).
contagem( 3 , 3 , 10 , 0 ).
contagem( 4 , 4 , 11 , 0 ).
contagem( 4 , 4 , 12 , 0 ).
contagem( 4 , 4 , 13 , 0 ).
contagem( 4 , 4 , 14 , 0 ).
contagem( 4 , 4 , 15 , 0 ).
contagem( 4 , 4 , 16 , 0 ).
contagem( 4 , 4 , 17 , 0 ).
contagem( 4 , 4 , 18 , 0 ).
contagem( 3 , 3 , 19 , 0 ).
contagem( 3 , 3 , 20 , 0 ).
contagem( 3 , 3 , 21 , 0 ).
contagem( 3 , 3 , 22 , 0 ).
contagem( 3 , 3 , 23 , 0 ).
contagem( 3 , 3 , 24 , 0 ).
contagem( 3 , 3 , 25 , 0 ).
contagem( 3 , 3 , 26 , 0 ).
contagem( 3 , 3 , 27 , 0 ).
contagem( 3 , 3 , 28 , 0 ).
contagem( 3 , 3 , 29 , 0 ).
contagem( 3 , 3 , 30 , 0 ).
contagem( 3 , 3 , 31 , 0 ).
contagem( 3 , 3 , 32 , 0 ).
contagem( 3 , 3 , 33 , 0 ).
contagem( 3 , 3 , 34 , 0 ).
contagem( 3 , 3 , 35 , 0 ).
contagem( 3 , 3 , 36 , 0 ).
contagem( 3 , 3 , 37 , 0 ).
contagem( 3 , 3 , 38 , 0 ).
contagem( 3 , 3 , 39 , 0 ).
contagem( 4 , 4 , 40 , 0 ).
contagem( 4 , 4 , 41 , 0 ).
contagem( 4 , 4 , 42 , 0 ).
contagem( 4 , 4 , 43 , 0 ).
contagem( 4 , 4 , 44 , 0 ).
contagem( 4 , 4 , 45 , 0 ).
contagem( 4 , 4 , 46 , 0 ).
contagem( 4 , 4 , 47 , 0 ).
contagem( 4 , 4 , 48 , 0 ).
contagem( 4 , 4 , 49 , 0 ).
contagem( 4 , 4 , 50 , 0 ).
contagem( 4 , 4 , 51 , 0 ).
```

```
contagem( 4 , 4 , 52 , 0 ).
contagem( 4 , 4 , 53 , 0 ).
contagem( 4 , 4 , 54 , 0 ).
contagem( 4 , 4 , 55 , 0 ).
contagem( 3 , 3 , 56 , 0 ).
contagem( 3 , 3 , 57 , 0 ).
contagem( 3 , 3 , 58 , 0 ).
contagem( 4 , 4 , 59 , 0 ).
contagem( 4 , 4 , 60 , 0 ).
contagem( 4 , 4 , 61 , 0 ).
contagem( 4 , 4 , 62 , 0 ).
contagem( 4 , 4 , 63 , 0 ).
contagem( 4 , 4 , 64 , 0 ).
contagem( 4 , 4 , 65 , 0 ).
contagem( 4 , 4 , 66 , 0 ).
contagem( 3 , 3 , 67 , 0 ).
contagem( 3 , 3 , 68 , 0 ).
contagem( 4 , 4 , 69 , 0 ).
contagem( 4 , 4 , 70 , 0 ).
contagem( 4 , 4 , 71 , 0 ).
contagem( 4 , 4 , 72 , 0 ).
contagem( 4 , 4 , 73 , 0 ).
contagem( 4 , 4 , 74 , 0 ).
contagem( 4 , 4 , 75 , 0 ).
contagem( 4 , 4 , 76 , 0 ).
contagem( 3 , 3 , 77 , 0 ).
contagem( 4 , 4 , 78 , 0 ).
contagem( 4 , 4 , 79 , 0 ).
contagem( 4 , 4 , 80 , 0 ).
contagem( 4 , 4 , 81 , 0 ).
contagem( 3 , 3 , 82 , 0 ).
contagem( 4 , 4 , 83 , 0 ).
contagem( 4 , 4 , 84 , 0 ).
contagem( 4 , 4 , 85 , 0 ).
contagem( 4 , 4 , 86 , 0 ).
contagem( 3 , 3 , 87 , 0 ).
contagem( 4 , 4 , 88 , 0 ).
contagem( 4 , 4 , 89 , 0 ).
contagem( 4 , 4 , 90 , 0 ).
contagem( 4 , 4 , 91 , 0 ).
contagem( 3 , 3 , 92 , 0 ).
```

APENDICE A3

ARQUIVO VOZ.INF

```
influ_cont( area(voz) , 1 ).
influ_cont( estrutura(filtro_fir) , 1 ).
influ_cont( quantiz(sim) , 1 ).
influ_cont( area(voz) , 2 ).
influ_cont( estrutura(filtro_fir) , 2 ).
influ_cont( quantiz(nao) , 2 ).
influ_cont( filtro(passa_baixa) , 2 ).
influ_cont( area(voz) , 3 ).
influ_cont( estrutura(filtro_fir) , 3 ).
influ_cont( quantiz(nao) , 3 ).
influ_cont( filtro(passa_alta) , 3 ).
influ_cont( area(voz) , 4 ).
influ_cont( estrutura(filtro_fir) , 4 ).
influ_cont( quantiz(nao) , 4 ).
influ_cont( filtro(passa_faixa) , 4 ).
influ_cont( area(voz) , 5 ).
influ_cont( estrutura(filtro_fir) , 5 ).
influ_cont( quantiz(nao) , 5 ).
influ_cont( filtro(rejeita_faixa) , 5 ).
influ_cont( area(voz) , 6 ).
influ_cont( estrutura(filtro_fir) , 6 ).
influ_cont( quantiz(nao) , 6 ).
influ_cont( filtro(multi_faixa) , 6 ).
influ_cont( area(voz) , 7 ).
influ_cont( estrutura(filtro_fir) , 7 ).
influ_cont( quantiz(nao) , 7 ).
influ_cont( filtro(diferenciador) , 7 ).
influ_cont( area(voz) , 8 ).
influ_cont( estrutura(filtro_fir) , 8 ).
influ_cont( quantiz(nao) , 8 ).
influ_cont( filtro(transf_hilbert) , 8 ).
influ_cont( area(voz) , 9 ).
influ_cont( estrutura(filtro_fir) , 9 ).
influ_cont( quantiz(nao) , 9 ).
influ_cont( filtro(passa_tudo) , 9 ).
influ_cont( area(voz) , 10 ).
influ_cont( estrutura(nao_sei) , 10 ).
influ_cont( quantiz(sim) , 10 ).
influ_cont( area(voz) , 11 ).
influ_cont( estrutura(nao_sei) , 11 ).
influ_cont( quantiz(nao) , 11 ).
influ_cont( filtro(passa_baixa) , 11 ).
influ_cont( area(voz) , 12 ).
influ_cont( estrutura(nao_sei) , 12 ).
influ_cont( quantiz(nao) , 12 ).
influ_cont( filtro(passa_alta) , 12 ).
influ_cont( area(voz) , 13 ).
influ_cont( estrutura(nao_sei) , 13 ).
influ_cont( quantiz(nao) , 13 ).
influ_cont( filtro(passa_faixa) , 13 ).
influ_cont( area(voz) , 14 ).
```

```
influ_cont( estrutura(nao_sei) , 14 ).
influ_cont( quantiz(nao) , 14 ).
influ_cont( filtro(rejeita_faixa) , 14 ).
influ_cont( area(voz) , 15 ).
influ_cont( estrutura(nao_sei) , 15 ).
influ_cont( quantiz(nao) , 15 ).
influ_cont( filtro(multi_faixa) , 15 ).
influ_cont( area(voz) , 16 ).
influ_cont( estrutura(nao_sei) , 16 ).
influ_cont( quantiz(nao) , 16 ).
influ_cont( filtro(diferenciador) , 16 ).
influ_cont( area(voz) , 17 ).
influ_cont( estrutura(nao_sei) , 17 ).
influ_cont( quantiz(nao) , 17 ).
influ_cont( filtro(transf_hilbert) , 17 ).
influ_cont( area(voz) , 18 ).
influ_cont( estrutura(nao_sei) , 18 ).
influ_cont( quantiz(nao) , 18 ).
influ_cont( filtro(passa_tudo) , 18 ).
influ_cont( area(voz) , 19 ).
influ_cont( estrutura(filtro_iir) , 19 ).
influ_cont( tipo_aprox(butterworth) , 19 ).
influ_cont( area(voz) , 20 ).
influ_cont( estrutura(filtro_iir) , 20 ).
influ_cont( tipo_aprox(chebyshev_1) , 20 ).
influ_cont( area(voz) , 21 ).
influ_cont( estrutura(filtro_iir) , 21 ).
influ_cont( tipo_aprox(chebyshev_2) , 21 ).
influ_cont( area(voz) , 22 ).
influ_cont( estrutura(filtro_iir) , 22 ).
influ_cont( tipo_aprox(eliptico) , 22 ).
influ_cont( area(voz) , 23 ).
influ_cont( estrutura(filtro_iir) , 23 ).
influ_cont( tipo_aprox(nao_sei) , 23 ).
influ_cont( programa(iir_bw) , 24 ).
influ_cont( fase(linear) , 24 ).
influ_cont( quantiz(sim) , 24 ).
influ_cont( programa(iir_bw) , 25 ).
influ_cont( fase(linear) , 25 ).
influ_cont( quantiz(nao) , 25 ).
influ_cont( programa(iir_bw) , 26 ).
influ_cont( fase(nao_linear) , 26 ).
influ_cont( quantiz(sim) , 26 ).
influ_cont( programa(iir_bw) , 27 ).
influ_cont( fase(nao_linear) , 27 ).
influ_cont( quantiz(nao) , 27 ).
influ_cont( programa(iir_c1) , 28 ).
influ_cont( fase(linear) , 28 ).
influ_cont( quantiz(sim) , 28 ).
influ_cont( programa(iir_c1) , 29 ).
influ_cont( fase(linear) , 29 ).
influ_cont( quantiz(nao) , 29 ).
influ_cont( programa(iir_c1) , 30 ).
influ_cont( fase(nao_linear) , 30 ).
influ_cont( quantiz(sim) , 30 ).
```

```
influ_cont( programa(iir_c1) , 31 ).
influ_cont( fase(nao_linear) , 31 ).
influ_cont( quantiz(nao) , 31 ).
influ_cont( programa(iir_c2) , 32 ).
influ_cont( fase(linear) , 32 ).
influ_cont( quantiz(sim) , 32 ).
influ_cont( programa(iir_c2) , 33 ).
influ_cont( fase(linear) , 33 ).
influ_cont( quantiz(nao) , 33 ).
influ_cont( programa(iir_c2) , 34 ).
influ_cont( fase(nao_linear) , 34 ).
influ_cont( quantiz(sim) , 34 ).
influ_cont( programa(iir_c2) , 35 ).
influ_cont( fase(nao_linear) , 35 ).
influ_cont( quantiz(nao) , 35 ).
influ_cont( programa(iir_el) , 36 ).
influ_cont( fase(linear) , 36 ).
influ_cont( quantiz(sim) , 36 ).
influ_cont( programa(iir_el) , 37 ).
influ_cont( fase(linear) , 37 ).
influ_cont( quantiz(nao) , 37 ).
influ_cont( programa(iir_el) , 38 ).
influ_cont( fase(nao_linear) , 38 ).
influ_cont( quantiz(sim) , 38 ).
influ_cont( programa(iir_el) , 39 ).
influ_cont( fase(nao_linear) , 39 ).
influ_cont( quantiz(nao) , 39 ).
influ_cont( algoritmo(fir) , 40 ).
influ_cont( ordem(ord1) , 40 ).
influ_cont( largura(larg1) , 40 ).
influ_cont( atenuacao(att1) , 40 ).
influ_cont( algoritmo(fir) , 41 ).
influ_cont( ordem(ord1) , 41 ).
influ_cont( largura(larg1) , 41 ).
influ_cont( atenuacao(att2) , 41 ).
influ_cont( algoritmo(fir) , 42 ).
influ_cont( ordem(ord1) , 42 ).
influ_cont( largura(larg1) , 42 ).
influ_cont( atenuacao(att3) , 42 ).
influ_cont( algoritmo(fir) , 43 ).
influ_cont( ordem(ord1) , 43 ).
influ_cont( largura(larg1) , 43 ).
influ_cont( atenuacao(att4) , 43 ).
influ_cont( algoritmo(fir) , 44 ).
influ_cont( ordem(ord1) , 44 ).
influ_cont( largura(larg2) , 44 ).
influ_cont( atenuacao(att1) , 44 ).
influ_cont( algoritmo(fir) , 45 ).
influ_cont( ordem(ord1) , 45 ).
influ_cont( largura(larg2) , 45 ).
influ_cont( atenuacao(att2) , 45 ).
influ_cont( algoritmo(fir) , 46 ).
influ_cont( ordem(ord1) , 46 ).
influ_cont( largura(larg2) , 46 ).
influ_cont( atenuacao(att3) , 46 ).
```

```
influ_cont( algoritmo(fir) , 47 ).
influ_cont( ordem(ord1) , 47 ).
influ_cont( largura(larg2) , 47 ).
influ_cont( atenuacao(att4) , 47 ).
influ_cont( algoritmo(fir) , 48 ).
influ_cont( ordem(ord1) , 48 ).
influ_cont( largura(larg3) , 48 ).
influ_cont( atenuacao(att1) , 48 ).
influ_cont( algoritmo(fir) , 49 ).
influ_cont( ordem(ord1) , 49 ).
influ_cont( largura(larg3) , 49 ).
influ_cont( atenuacao(att2) , 49 ).
influ_cont( algoritmo(fir) , 50 ).
influ_cont( ordem(ord1) , 50 ).
influ_cont( largura(larg3) , 50 ).
influ_cont( atenuacao(att3) , 50 ).
influ_cont( algoritmo(fir) , 51 ).
influ_cont( ordem(ord1) , 51 ).
influ_cont( largura(larg3) , 51 ).
influ_cont( atenuacao(att4) , 51 ).
influ_cont( algoritmo(fir) , 52 ).
influ_cont( ordem(ord1) , 52 ).
influ_cont( largura(larg4) , 52 ).
influ_cont( atenuacao(att1) , 52 ).
influ_cont( algoritmo(fir) , 53 ).
influ_cont( ordem(ord1) , 53 ).
influ_cont( largura(larg4) , 53 ).
influ_cont( atenuacao(att2) , 53 ).
influ_cont( algoritmo(fir) , 54 ).
influ_cont( ordem(ord1) , 54 ).
influ_cont( largura(larg4) , 54 ).
influ_cont( atenuacao(att3) , 54 ).
influ_cont( algoritmo(fir) , 55 ).
influ_cont( ordem(ord1) , 55 ).
influ_cont( largura(larg4) , 55 ).
influ_cont( atenuacao(att4) , 55 ).
influ_cont( algoritmo(fir) , 56 ).
influ_cont( ordem(ord2) , 56 ).
influ_cont( largura(larg1) , 56 ).
influ_cont( algoritmo(fir) , 57 ).
influ_cont( ordem(ord2) , 57 ).
influ_cont( largura(larg2) , 57 ).
influ_cont( algoritmo(fir) , 58 ).
influ_cont( ordem(ord2) , 58 ).
influ_cont( largura(larg3) , 58 ).
influ_cont( algoritmo(fir) , 59 ).
influ_cont( ordem(ord2) , 59 ).
influ_cont( largura(larg4) , 59 ).
influ_cont( atenuacao(att1) , 59 ).
influ_cont( algoritmo(fir) , 60 ).
influ_cont( ordem(ord2) , 60 ).
influ_cont( largura(larg4) , 60 ).
influ_cont( atenuacao(att2) , 60 ).
influ_cont( algoritmo(fir) , 61 ).
influ_cont( ordem(ord2) , 61 ).
```

```
influ_cont( largura(larg4) , 61 ).
influ_cont( atenuacao(att3) , 61 ).
influ_cont( algoritmo(fir) , 62 ).
influ_cont( ordem(ord2) , 62 ).
influ_cont( largura(larg4) , 62 ).
influ_cont( atenuacao(att4) , 62 ).
influ_cont( algoritmo(fir) , 63 ).
influ_cont( ordem(ord3) , 63 ).
influ_cont( largura(larg1) , 63 ).
influ_cont( atenuacao(att1) , 63 ).
influ_cont( algoritmo(fir) , 64 ).
influ_cont( ordem(ord3) , 64 ).
influ_cont( largura(larg1) , 64 ).
influ_cont( atenuacao(att2) , 64 ).
influ_cont( algoritmo(fir) , 65 ).
influ_cont( ordem(ord3) , 65 ).
influ_cont( largura(larg1) , 65 ).
influ_cont( atenuacao(att3) , 65 ).
influ_cont( algoritmo(fir) , 66 ).
influ_cont( ordem(ord3) , 66 ).
influ_cont( largura(larg1) , 66 ).
influ_cont( atenuacao(att4) , 66 ).
influ_cont( algoritmo(fir) , 67 ).
influ_cont( ordem(ord3) , 67 ).
influ_cont( largura(larg2) , 67 ).
influ_cont( algoritmo(fir) , 68 ).
influ_cont( ordem(ord3) , 68 ).
influ_cont( largura(larg3) , 68 ).
influ_cont( algoritmo(fir) , 69 ).
influ_cont( ordem(ord3) , 69 ).
influ_cont( largura(larg4) , 69 ).
influ_cont( atenuacao(att1) , 69 ).
influ_cont( algoritmo(fir) , 70 ).
influ_cont( ordem(ord3) , 70 ).
influ_cont( largura(larg4) , 70 ).
influ_cont( atenuacao(att2) , 70 ).
influ_cont( algoritmo(fir) , 71 ).
influ_cont( ordem(ord3) , 71 ).
influ_cont( largura(larg4) , 71 ).
influ_cont( atenuacao(att3) , 71 ).
influ_cont( algoritmo(fir) , 72 ).
influ_cont( ordem(ord3) , 72 ).
influ_cont( largura(larg4) , 72 ).
influ_cont( atenuacao(att4) , 72 ).
influ_cont( algoritmo(fir) , 73 ).
influ_cont( ordem(ord4) , 73 ).
influ_cont( largura(larg1) , 73 ).
influ_cont( atenuacao(att1) , 73 ).
influ_cont( algoritmo(fir) , 74 ).
influ_cont( ordem(ord4) , 74 ).
influ_cont( largura(larg1) , 74 ).
influ_cont( atenuacao(att2) , 74 ).
influ_cont( algoritmo(fir) , 75 ).
influ_cont( ordem(ord4) , 75 ).
influ_cont( largura(larg1) , 75 ).
```

```
influ_cont( atenuacao(att3) , 75 ).
influ_cont( algoritmo(fir) , 76 ).
influ_cont( ordem(ord4) , 76 ).
influ_cont( largura(larg1) , 76 ).
influ_cont( atenuacao(att4) , 76 ).
influ_cont( algoritmo(fir) , 77 ).
influ_cont( ordem(ord4) , 77 ).
influ_cont( largura(larg2) , 77 ).
influ_cont( algoritmo(fir) , 78 ).
influ_cont( ordem(ord4) , 78 ).
influ_cont( largura(larg3) , 78 ).
influ_cont( atenuacao(att1) , 78 ).
influ_cont( algoritmo(fir) , 79 ).
influ_cont( ordem(ord4) , 79 ).
influ_cont( largura(larg3) , 79 ).
influ_cont( atenuacao(att2) , 79 ).
influ_cont( algoritmo(fir) , 80 ).
influ_cont( ordem(ord4) , 80 ).
influ_cont( largura(larg3) , 80 ).
influ_cont( atenuacao(att3) , 80 ).
influ_cont( algoritmo(fir) , 81 ).
influ_cont( ordem(ord4) , 81 ).
influ_cont( largura(larg3) , 81 ).
influ_cont( atenuacao(att4) , 81 ).
influ_cont( algoritmo(fir) , 82 ).
influ_cont( ordem(ord4) , 82 ).
influ_cont( largura(larg4) , 82 ).
influ_cont( algoritmo(fir) , 83 ).
influ_cont( ordem(ord5) , 83 ).
influ_cont( largura(larg1) , 83 ).
influ_cont( atenuacao(att1) , 83 ).
influ_cont( algoritmo(fir) , 84 ).
influ_cont( ordem(ord5) , 84 ).
influ_cont( largura(larg1) , 84 ).
influ_cont( atenuacao(att2) , 84 ).
influ_cont( algoritmo(fir) , 85 ).
influ_cont( ordem(ord5) , 85 ).
influ_cont( largura(larg1) , 85 ).
influ_cont( atenuacao(att3) , 85 ).
influ_cont( algoritmo(fir) , 86 ).
influ_cont( ordem(ord5) , 86 ).
influ_cont( largura(larg1) , 86 ).
influ_cont( atenuacao(att4) , 86 ).
influ_cont( algoritmo(fir) , 87 ).
influ_cont( ordem(ord5) , 87 ).
influ_cont( largura(larg2) , 87 ).
influ_cont( algoritmo(fir) , 88 ).
influ_cont( ordem(ord5) , 88 ).
influ_cont( largura(larg3) , 88 ).
influ_cont( atenuacao(att1) , 88 ).
influ_cont( algoritmo(fir) , 89 ).
influ_cont( ordem(ord5) , 89 ).
influ_cont( largura(larg3) , 89 ).
influ_cont( atenuacao(att2) , 89 ).
influ_cont( algoritmo(fir) , 90 ).
```

```
influ_cont( ordem(ord5) , 90 ).
influ_cont( largura(larg3) , 90 ).
influ_cont( atenuacao(att3) , 90 ).
influ_cont( algoritmo(fir) , 91 ).
influ_cont( ordem(ord5) , 91 ).
influ_cont( largura(larg3) , 91 ).
influ_cont( atenuacao(att4) , 91 ).
influ_cont( algoritmo(fir) , 92 ).
influ_cont( ordem(ord5) , 92 ).
influ_cont( largura(larg4) , 92 ).
```

APENDICE A4

ARQUIVO VOZ.PRI

```
prior(1,0).
prior(2,0).
prior(3,0).
prior(4,0).
prior(5,0).
prior(6,0).
prior(7,0).
prior(8,0).
prior(9,0).
prior(10,0).
prior(11,0).
prior(12,0).
prior(13,0).
prior(14,0).
prior(15,0).
prior(16,0).
prior(17,0).
prior(18,0).
prior(19,0).
prior(20,0).
prior(21,0).
prior(22,0).
prior(23,0).
prior(24,0).
prior(25,0).
prior(26,0).
prior(27,0).
prior(28,0).
prior(29,0).
prior(30,0).
prior(31,0).
prior(32,0).
prior(33,0).
prior(34,0).
prior(35,0).
prior(36,0).
prior(37,0).
prior(38,0).
prior(39,0).
prior(40,0).
prior(41,0).
prior(42,0).
prior(43,0).
prior(44,0).
prior(45,0).
prior(46,0).
prior(47,0).
prior(48,0).
prior(49,0).
prior(50,0).
prior(51,0).
```

```
prior(52,0).
prior(53,0).
prior(54,0).
prior(55,0).
prior(56,0).
prior(57,0).
prior(58,0).
prior(59,0).
prior(60,0).
prior(61,0).
prior(62,0).
prior(63,0).
prior(64,0).
prior(65,0).
prior(66,0).
prior(67,0).
prior(68,0).
prior(69,0).
prior(70,0).
prior(71,0).
prior(72,0).
prior(73,0).
prior(74,0).
prior(75,0).
prior(76,0).
prior(77,0).
prior(78,0).
prior(79,0).
prior(80,0).
prior(81,0).
prior(82,0).
prior(83,0).
prior(84,0).
prior(85,0).
prior(86,0).
prior(87,0).
prior(88,0).
prior(89,0).
prior(90,0).
prior(91,0).
prior(92,0).
```

APENDICE A5

ARQUIVO OUTRAS.REG

```
regra( 1, se ( area(outras) , estrutura(filtro_fir) )
entao (  guarde_f(programa(fir)))).
regra(   2,   se  (  area(outras)   ,   estrutura(filtro_iir)   ,
tipo_aprox(butterworth) )
entao (  guarde_f(programa(iir_bw)))).
regra(   3,   se  (  area(outras)   ,   estrutura(filtro_iir)   ,
tipo_aprox(chebyshev_1) )
entao (  guarde_f(programa(iir_c1)))).
regra(   4,   se  (  area(outras)   ,   estrutura(filtro_iir)   ,
tipo_aprox(chebyshev_2) )
entao (  guarde_f(programa(iir_c2)))).
regra(   5,   se  (  area(outras)   ,   estrutura(filtro_iir)   ,
tipo_aprox(eliptico) )
entao (  guarde_f(programa(iir_el)))).
regra(   6,   se  (  area(outras)   ,   estrutura(filtro_iir)   ,
tipo_aprox(nao_sei) )
entao (  guarde_f(programa(iir_el)))).
regra( 7, se ( area(outras) , estrutura(nao_sei) , fase(linear) ,
ordem(nao_sei) )
entao (  guarde_f(programa(fir)))).
regra(   8,   se   (   area(outras)   ,   estrutura(nao_sei)   ,
fase(nao_linear) , ordem(nao_sei) )
entao (  guarde_f(programa(iir_el)))).
regra( 9, se ( area(outras) , estrutura(nao_sei) , fase(linear) ,
ordem(ord1) )
entao (  guarde_f(programa(iir_el)))).
regra(   10,   se   (   area(outras)   ,   estrutura(nao_sei)   ,
fase(nao_linear) , ordem(ord1) )
entao (  guarde_f(programa(iir_el)))).
regra( 11, se ( area(outras) , estrutura(nao_sei) , ordem(ord2) )
entao (  guarde_f(programa(fir)))).
regra( 12, se ( area(outras) , estrutura(nao_sei) , ordem(ord3) )
entao (  guarde_f(programa(fir)))).
regra( 13, se ( area(outras) , estrutura(nao_sei) , ordem(ord4) )
entao (  guarde_f(programa(fir)))).
regra( 14, se ( area(outras) , estrutura(nao_sei) , ordem(ord5) )
entao (  guarde_f(programa(fir)))).
regra( 15, se ( programa(iir_bw) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124bw_lin))).
regra( 16, se ( programa(iir_bw) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12bw_lin))).
regra(   17,   se  (  programa(iir_bw)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124bw_nlin))).
regra(   18,   se  (  programa(iir_bw)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12bw_nlin))).
regra( 19, se ( programa(iir_c1) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124c1_lin))).
regra( 20, se ( programa(iir_c1) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12c1_lin))).
```

```
regra(   21,   se  (  programa(iir_c1)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124c1_nlin))).
regra(   22,   se  (  programa(iir_c1)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12c1_nlin))).
regra( 23, se ( programa(iir_c2) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124c2_lin))).
regra( 24, se ( programa(iir_c2) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12c2_lin))).
regra(   25,   se  (  programa(iir_c2)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124c2_nlin))).
regra(   26,   se  (  programa(iir_c2)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12c2_nlin))).
regra( 27, se ( programa(iir_el) , fase(linear) , quantiz(sim) )
entao (  guarde_d(iir124el_lin))).
regra( 28, se ( programa(iir_el) , fase(linear) , quantiz(nao) )
entao (  guarde_d(iir12el_lin))).
regra(   29,   se  (  programa(iir_el)   ,   fase(nao_linear)   ,
quantiz(sim) )
entao (  guarde_d(iir124el_nlin))).
regra(   30,   se  (  programa(iir_el)   ,   fase(nao_linear)   ,
quantiz(nao) )
entao (  guarde_d(iir12el_nlin))).
regra( 31, se ( programa(fir) , quantiz(sim) )
entao (  guarde_d(fir4))).
regra(    32,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(passa_baixa) )
entao (  guarde_f(algoritmo(fir)))).
regra( 33, se ( programa(fir) , quantiz(nao) , filtro(passa_alta)
)
entao (  guarde_f(algoritmo(fir)))).
regra(    34,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(passa_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra(    35,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(rejeita_faixa) )
entao (  guarde_f(algoritmo(fir)))).
regra(    36,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(multi_faixa) )
entao (  guarde_d(fir1))).
regra(    37,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(diferenciador) )
entao (  guarde_d(fir1))).
regra(    38,    se   (   programa(fir)    ,    quantiz(nao)    ,
filtro(transf_hilbert) )
entao (  guarde_d(fir1))).
regra( 39, se ( programa(fir) , quantiz(nao) , filtro(passa_tudo)
)
entao (  guarde_d(fir1))).
regra(  40, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
```

```
regra(  41, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(kaiser))).
regra(  42, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(kaiser))).
regra(  43, se ( algoritmo(fir) , ordem(ord1) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(kaiser))).
regra(  44, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
regra(  45, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att2) )
entao (  guarde_d(chebyshev))).
regra(  46, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att3) )
entao (  guarde_d(chebyshev))).
regra(  47, se ( algoritmo(fir) , ordem(ord1) , largura(larg2)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra(  48, se ( algoritmo(fir) , ordem(ord1) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
rd(fir1))).
regra( 67, se ( algoritmo(fir) , ordem(ord3) , largura(larg2) )
entao (  guarde_d(fir1))).
regra( 68, se ( algoritmo(fir) , ordem(ord3) , largura(larg3) )
entao (  guarde_d(fir1))).
regra(  69, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att1) )
entao (  guarde_d(fir1))).
regra(  70, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att2) )
entao (  guarde_d(hanning))).
regra(  71, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att3) )
entao (  guarde_d(hanning))).
regra(  72, se ( algoritmo(fir) , ordem(ord3) , largura(larg4)  ,
atenuacao(att4) )
entao (  guarde_d(chebyshev))).
regra(  73, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  74, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  75, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  76, se ( algoritmo(fir) , ordem(ord4) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(fir1))).
regra( 77, se ( algoritmo(fir) , ordem(ord4) , largura(larg2) )
entao (  guarde_d(fir1))).
```

```
regra(  78, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(kaiser))).
regra(  79, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  80, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  81, se ( algoritmo(fir) , ordem(ord4) , largura(larg3)  ,
atenuacao(att4) )
entao (  guarde_d(gen_hamming))).
regra( 82, se ( algoritmo(fir) , ordem(ord4) , largura(larg4) )
entao (  guarde_d(gen_hamming))).
regra(  83, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att1) )
entao (  guarde_d(retangular))).
regra(  84, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att2) )
entao (  guarde_d(fir1))).
regra(  85, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att3) )
entao (  guarde_d(fir1))).
regra(  86, se ( algoritmo(fir) , ordem(ord5) , largura(larg1)  ,
atenuacao(att4) )
entao (  guarde_d(fir1))).
regra( 87, se ( algoritmo(fir) , ordem(ord5) , largura(larg2) )
entao (  guarde_d(fir1))).
regra(  88, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att1) )
entao (  guarde_d(hanning))).
regra(  89, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att2) )
entao (  guarde_d(hanning))).
regra(  90, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att3) )
entao (  guarde_d(hanning))).
regra(  91, se ( algoritmo(fir) , ordem(ord5) , largura(larg3)  ,
atenuacao(att4) )
entao (  guarde_d(hamming))).
regra( 92, se ( algoritmo(fir) , ordem(ord5) , largura(larg4) )
entao (  guarde_d(fir1))).
```

APENDICE A6

ARQUIVO OUTRAS.CON

```
contagem( 2 , 2 , 1 , 0 ).
contagem( 3 , 3 , 2 , 0 ).
contagem( 3 , 3 , 3 , 0 ).
contagem( 3 , 3 , 4 , 0 ).
contagem( 3 , 3 , 5 , 0 ).
contagem( 3 , 3 , 6 , 0 ).
contagem( 4 , 4 , 7 , 0 ).
contagem( 4 , 4 , 8 , 0 ).
contagem( 4 , 4 , 9 , 0 ).
contagem( 4 , 4 , 10 , 0 ).
contagem( 3 , 3 , 11 , 0 ).
contagem( 3 , 3 , 12 , 0 ).
contagem( 3 , 3 , 13 , 0 ).
contagem( 3 , 3 , 14 , 0 ).
contagem( 3 , 3 , 15 , 0 ).
contagem( 3 , 3 , 16 , 0 ).
contagem( 3 , 3 , 17 , 0 ).
contagem( 3 , 3 , 18 , 0 ).
contagem( 3 , 3 , 19 , 0 ).
contagem( 3 , 3 , 20 , 0 ).
contagem( 3 , 3 , 21 , 0 ).
contagem( 3 , 3 , 22 , 0 ).
contagem( 3 , 3 , 23 , 0 ).
contagem( 3 , 3 , 24 , 0 ).
contagem( 3 , 3 , 25 , 0 ).
contagem( 3 , 3 , 26 , 0 ).
contagem( 3 , 3 , 27 , 0 ).
contagem( 3 , 3 , 28 , 0 ).
contagem( 3 , 3 , 29 , 0 ).
contagem( 3 , 3 , 30 , 0 ).
contagem( 2 , 2 , 31 , 0 ).
contagem( 3 , 3 , 32 , 0 ).
contagem( 3 , 3 , 33 , 0 ).
contagem( 3 , 3 , 34 , 0 ).
contagem( 3 , 3 , 35 , 0 ).
contagem( 3 , 3 , 36 , 0 ).
contagem( 3 , 3 , 37 , 0 ).
contagem( 3 , 3 , 38 , 0 ).
contagem( 3 , 3 , 39 , 0 ).
contagem( 4 , 4 , 40 , 0 ).
contagem( 4 , 4 , 41 , 0 ).
contagem( 4 , 4 , 42 , 0 ).
contagem( 4 , 4 , 43 , 0 ).
contagem( 4 , 4 , 44 , 0 ).
contagem( 4 , 4 , 45 , 0 ).
contagem( 4 , 4 , 46 , 0 ).
contagem( 4 , 4 , 47 , 0 ).
contagem( 4 , 4 , 48 , 0 ).
contagem( 4 , 4 , 49 , 0 ).
contagem( 4 , 4 , 50 , 0 ).
contagem( 4 , 4 , 51 , 0 ).
```

```
contagem( 4 , 4 , 52 , 0 ).
contagem( 4 , 4 , 53 , 0 ).
contagem( 4 , 4 , 54 , 0 ).
contagem( 4 , 4 , 55 , 0 ).
contagem( 3 , 3 , 56 , 0 ).
contagem( 3 , 3 , 57 , 0 ).
contagem( 3 , 3 , 58 , 0 ).
contagem( 4 , 4 , 59 , 0 ).
contagem( 4 , 4 , 60 , 0 ).
contagem( 4 , 4 , 61 , 0 ).
contagem( 4 , 4 , 62 , 0 ).
contagem( 4 , 4 , 63 , 0 ).
contagem( 4 , 4 , 64 , 0 ).
contagem( 4 , 4 , 65 , 0 ).
contagem( 4 , 4 , 66 , 0 ).
contagem( 3 , 3 , 67 , 0 ).
contagem( 3 , 3 , 68 , 0 ).
contagem( 4 , 4 , 69 , 0 ).
contagem( 4 , 4 , 70 , 0 ).
contagem( 4 , 4 , 71 , 0 ).
contagem( 4 , 4 , 72 , 0 ).
contagem( 4 , 4 , 73 , 0 ).
contagem( 4 , 4 , 74 , 0 ).
contagem( 4 , 4 , 75 , 0 ).
contagem( 4 , 4 , 76 , 0 ).
contagem( 3 , 3 , 77 , 0 ).
contagem( 4 , 4 , 78 , 0 ).
contagem( 4 , 4 , 79 , 0 ).
contagem( 4 , 4 , 80 , 0 ).
contagem( 4 , 4 , 81 , 0 ).
contagem( 3 , 3 , 82 , 0 ).
contagem( 4 , 4 , 83 , 0 ).
contagem( 4 , 4 , 84 , 0 ).
contagem( 4 , 4 , 85 , 0 ).
contagem( 4 , 4 , 86 , 0 ).
contagem( 3 , 3 , 87 , 0 ).
contagem( 4 , 4 , 88 , 0 ).
contagem( 4 , 4 , 89 , 0 ).
contagem( 4 , 4 , 90 , 0 ).
contagem( 4 , 4 , 91 , 0 ).
contagem( 3 , 3 , 92 , 0 ).
```

APENDICE A7

ARQUIVO OUTRAS.INF

```
influ_cont( area(outras) , 1 ).
influ_cont( estrutura(filtro_fir) , 1 ).
influ_cont( area(outras) , 2 ).
influ_cont( estrutura(filtro_iir) , 2 ).
influ_cont( tipo_aprox(butterworth) , 2 ).
influ_cont( area(outras) , 3 ).
influ_cont( estrutura(filtro_iir) , 3 ).
influ_cont( tipo_aprox(chebyshev_1) , 3 ).
influ_cont( area(outras) , 4 ).
influ_cont( estrutura(filtro_iir) , 4 ).
influ_cont( tipo_aprox(chebyshev_2) , 4 ).
influ_cont( area(outras) , 5 ).
influ_cont( estrutura(filtro_iir) , 5 ).
influ_cont( tipo_aprox(eliptico) , 5 ).
influ_cont( area(outras) , 6 ).
influ_cont( estrutura(filtro_iir) , 6 ).
influ_cont( tipo_aprox(nao_sei) , 6 ).
influ_cont( area(outras) , 7 ).
influ_cont( estrutura(nao_sei) , 7 ).
influ_cont( fase(linear) , 7 ).
influ_cont( ordem(nao_sei) , 7 ).
influ_cont( area(outras) , 8 ).
influ_cont( estrutura(nao_sei) , 8 ).
influ_cont( fase(nao_linear) , 8 ).
influ_cont( ordem(nao_sei) , 8 ).
influ_cont( area(outras) , 9 ).
influ_cont( estrutura(nao_sei) , 9 ).
influ_cont( fase(linear) , 9 ).
influ_cont( ordem(ord1) , 9 ).
influ_cont( area(outras) , 10 ).
influ_cont( estrutura(nao_sei) , 10 ).
influ_cont( fase(nao_linear) , 10 ).
influ_cont( ordem(ord1) , 10 ).
influ_cont( area(outras) , 11 ).
influ_cont( estrutura(nao_sei) , 11 ).
influ_cont( ordem(ord2) , 11 ).
influ_cont( area(outras) , 12 ).
influ_cont( estrutura(nao_sei) , 12 ).
influ_cont( ordem(ord3) , 12 ).
influ_cont( area(outras) , 13 ).
influ_cont( estrutura(nao_sei) , 13 ).
influ_cont( ordem(ord4) , 13 ).
influ_cont( area(outras) , 14 ).
influ_cont( estrutura(nao_sei) , 14 ).
influ_cont( ordem(ord5) , 14 ).
influ_cont( programa(iir_bw) , 15 ).
influ_cont( fase(linear) , 15 ).
influ_cont( quantiz(sim) , 15 ).
influ_cont( programa(iir_bw) , 16 ).
influ_cont( fase(linear) , 16 ).
influ_cont( quantiz(nao) , 16 ).
```

```
influ_cont( programa(iir_bw) , 17 ).
influ_cont( fase(nao_linear) , 17 ).
influ_cont( quantiz(sim) , 17 ).
influ_cont( programa(iir_bw) , 18 ).
influ_cont( fase(nao_linear) , 18 ).
influ_cont( quantiz(nao) , 18 ).
influ_cont( programa(iir_c1) , 19 ).
influ_cont( fase(linear) , 19 ).
influ_cont( quantiz(sim) , 19 ).
influ_cont( programa(iir_c1) , 20 ).
influ_cont( fase(linear) , 20 ).
influ_cont( quantiz(nao) , 20 ).
influ_cont( programa(iir_c1) , 21 ).
influ_cont( fase(nao_linear) , 21 ).
influ_cont( quantiz(sim) , 21 ).
influ_cont( programa(iir_c1) , 22 ).
influ_cont( fase(nao_linear) , 22 ).
influ_cont( quantiz(nao) , 22 ).
influ_cont( programa(iir_c2) , 23 ).
influ_cont( fase(linear) , 23 ).
influ_cont( quantiz(sim) , 23 ).
influ_cont( programa(iir_c2) , 24 ).
influ_cont( fase(linear) , 24 ).
influ_cont( quantiz(nao) , 24 ).
influ_cont( programa(iir_c2) , 25 ).
influ_cont( fase(nao_linear) , 25 ).
influ_cont( quantiz(sim) , 25 ).
influ_cont( programa(iir_c2) , 26 ).
influ_cont( fase(nao_linear) , 26 ).
influ_cont( quantiz(nao) , 26 ).
influ_cont( programa(iir_el) , 27 ).
influ_cont( fase(linear) , 27 ).
influ_cont( quantiz(sim) , 27 ).
influ_cont( programa(iir_el) , 28 ).
influ_cont( fase(linear) , 28 ).
influ_cont( quantiz(nao) , 28 ).
influ_cont( programa(iir_el) , 29 ).
influ_cont( fase(nao_linear) , 29 ).
influ_cont( quantiz(sim) , 29 ).
influ_cont( programa(iir_el) , 30 ).
influ_cont( fase(nao_linear) , 30 ).
influ_cont( quantiz(nao) , 30 ).
influ_cont( programa(fir) , 31 ).
influ_cont( quantiz(sim) , 31 ).
influ_cont( programa(fir) , 32 ).
influ_cont( quantiz(nao) , 32 ).
influ_cont( filtro(passa_baixa) , 32 ).
influ_cont( programa(fir) , 33 ).
influ_cont( quantiz(nao) , 33 ).
influ_cont( filtro(passa_alta) , 33 ).
influ_cont( programa(fir) , 34 ).
influ_cont( quantiz(nao) , 34 ).
influ_cont( filtro(passa_faixa) , 34 ).
influ_cont( programa(fir) , 35 ).
influ_cont( quantiz(nao) , 35 ).
```

```
influ_cont( filtro(rejeita_faixa) , 35 ).
influ_cont( programa(fir) , 36 ).
influ_cont( quantiz(nao) , 36 ).
influ_cont( filtro(multi_faixa) , 36 ).
influ_cont( programa(fir) , 37 ).
influ_cont( quantiz(nao) , 37 ).
influ_cont( filtro(diferenciador) , 37 ).
influ_cont( programa(fir) , 38 ).
influ_cont( quantiz(nao) , 38 ).
influ_cont( filtro(transf_hilbert) , 38 ).
influ_cont( programa(fir) , 39 ).
influ_cont( quantiz(nao) , 39 ).
influ_cont( filtro(passa_tudo) , 39 ).
influ_cont( algoritmo(fir) , 40 ).
influ_cont( ordem(ord1) , 40 ).
influ_cont( largura(larg1) , 40 ).
influ_cont( atenuacao(att1) , 40 ).
influ_cont( algoritmo(fir) , 41 ).
influ_cont( ordem(ord1) , 41 ).
influ_cont( largura(larg1) , 41 ).
influ_cont( atenuacao(att2) , 41 ).
influ_cont( algoritmo(fir) , 42 ).
influ_cont( ordem(ord1) , 42 ).
influ_cont( largura(larg1) , 42 ).
influ_cont( atenuacao(att3) , 42 ).
influ_cont( algoritmo(fir) , 43 ).
influ_cont( ordem(ord1) , 43 ).
influ_cont( largura(larg1) , 43 ).
influ_cont( atenuacao(att4) , 43 ).
influ_cont( algoritmo(fir) , 44 ).
influ_cont( ordem(ord1) , 44 ).
influ_cont( largura(larg2) , 44 ).
influ_cont( atenuacao(att1) , 44 ).
influ_cont( algoritmo(fir) , 45 ).
influ_cont( ordem(ord1) , 45 ).
influ_cont( largura(larg2) , 45 ).
influ_cont( atenuacao(att2) , 45 ).
influ_cont( algoritmo(fir) , 46 ).
influ_cont( ordem(ord1) , 46 ).
influ_cont( largura(larg2) , 46 ).
influ_cont( atenuacao(att3) , 46 ).
influ_cont( algoritmo(fir) , 47 ).
influ_cont( ordem(ord1) , 47 ).
influ_cont( largura(larg2) , 47 ).
influ_cont( atenuacao(att4) , 47 ).
influ_cont( algoritmo(fir) , 48 ).
influ_cont( ordem(ord1) , 48 ).
influ_cont( largura(larg3) , 48 ).
influ_cont( atenuacao(att1) , 48 ).
influ_cont( algoritmo(fir) , 49 ).
influ_cont( ordem(ord1) , 49 ).
influ_cont( largura(larg3) , 49 ).
influ_cont( atenuacao(att2) , 49 ).
influ_cont( algoritmo(fir) , 50 ).
influ_cont( ordem(ord1) , 50 ).
```

```
influ_cont( largura(larg3) , 50 ).
influ_cont( atenuacao(att3) , 50 ).
influ_cont( algoritmo(fir) , 51 ).
influ_cont( ordem(ord1) , 51 ).
influ_cont( largura(larg3) , 51 ).
influ_cont( atenuacao(att4) , 51 ).
influ_cont( algoritmo(fir) , 52 ).
influ_cont( ordem(ord1) , 52 ).
influ_cont( largura(larg4) , 52 ).
influ_cont( atenuacao(att1) , 52 ).
influ_cont( algoritmo(fir) , 53 ).
influ_cont( ordem(ord1) , 53 ).
influ_cont( largura(larg4) , 53 ).
influ_cont( atenuacao(att2) , 53 ).
influ_cont( algoritmo(fir) , 54 ).
influ_cont( ordem(ord1) , 54 ).
influ_cont( largura(larg4) , 54 ).
influ_cont( atenuacao(att3) , 54 ).
influ_cont( algoritmo(fir) , 55 ).
influ_cont( ordem(ord1) , 55 ).
influ_cont( largura(larg4) , 55 ).
influ_cont( atenuacao(att4) , 55 ).
influ_cont( algoritmo(fir) , 56 ).
influ_cont( ordem(ord2) , 56 ).
influ_cont( largura(larg1) , 56 ).
influ_cont( algoritmo(fir) , 57 ).
influ_cont( ordem(ord2) , 57 ).
influ_cont( largura(larg2) , 57 ).
influ_cont( algoritmo(fir) , 58 ).
influ_cont( ordem(ord2) , 58 ).
influ_cont( largura(larg3) , 58 ).
influ_cont( algoritmo(fir) , 59 ).
influ_cont( ordem(ord2) , 59 ).
influ_cont( largura(larg4) , 59 ).
influ_cont( atenuacao(att1) , 59 ).
influ_cont( algoritmo(fir) , 60 ).
influ_cont( ordem(ord2) , 60 ).
influ_cont( largura(larg4) , 60 ).
influ_cont( atenuacao(att2) , 60 ).
influ_cont( algoritmo(fir) , 61 ).
influ_cont( ordem(ord2) , 61 ).
influ_cont( largura(larg4) , 61 ).
influ_cont( atenuacao(att3) , 61 ).
influ_cont( algoritmo(fir) , 62 ).
influ_cont( ordem(ord2) , 62 ).
influ_cont( largura(larg4) , 62 ).
influ_cont( atenuacao(att4) , 62 ).
influ_cont( algoritmo(fir) , 63 ).
influ_cont( ordem(ord3) , 63 ).
influ_cont( largura(larg1) , 63 ).
influ_cont( atenuacao(att1) , 63 ).
influ_cont( algoritmo(fir) , 64 ).
influ_cont( ordem(ord3) , 64 ).
influ_cont( largura(larg1) , 64 ).
influ_cont( atenuacao(att2) , 64 ).
```

```
influ_cont( algoritmo(fir) , 65 ).
influ_cont( ordem(ord3) , 65 ).
influ_cont( largura(larg1) , 65 ).
influ_cont( atenuacao(att3) , 65 ).
influ_cont( algoritmo(fir) , 66 ).
influ_cont( ordem(ord3) , 66 ).
influ_cont( largura(larg1) , 66 ).
influ_cont( atenuacao(att4) , 66 ).
influ_cont( algoritmo(fir) , 67 ).
influ_cont( ordem(ord3) , 67 ).
influ_cont( largura(larg2) , 67 ).
influ_cont( algoritmo(fir) , 68 ).
influ_cont( ordem(ord3) , 68 ).
influ_cont( largura(larg3) , 68 ).
influ_cont( algoritmo(fir) , 69 ).
influ_cont( ordem(ord3) , 69 ).
influ_cont( largura(larg4) , 69 ).
influ_cont( atenuacao(att1) , 69 ).
influ_cont( algoritmo(fir) , 70 ).
influ_cont( ordem(ord3) , 70 ).
influ_cont( largura(larg4) , 70 ).
influ_cont( atenuacao(att2) , 70 ).
influ_cont( algoritmo(fir) , 71 ).
influ_cont( ordem(ord3) , 71 ).
influ_cont( largura(larg4) , 71 ).
influ_cont( atenuacao(att3) , 71 ).
influ_cont( algoritmo(fir) , 72 ).
influ_cont( ordem(ord3) , 72 ).
influ_cont( largura(larg4) , 72 ).
influ_cont( atenuacao(att4) , 72 ).
influ_cont( algoritmo(fir) , 73 ).
influ_cont( ordem(ord4) , 73 ).
influ_cont( largura(larg1) , 73 ).
influ_cont( atenuacao(att1) , 73 ).
influ_cont( algoritmo(fir) , 74 ).
influ_cont( ordem(ord4) , 74 ).
influ_cont( largura(larg1) , 74 ).
influ_cont( atenuacao(att2) , 74 ).
influ_cont( algoritmo(fir) , 75 ).
influ_cont( ordem(ord4) , 75 ).
influ_cont( largura(larg1) , 75 ).
influ_cont( atenuacao(att3) , 75 ).
influ_cont( algoritmo(fir) , 76 ).
influ_cont( ordem(ord4) , 76 ).
influ_cont( largura(larg1) , 76 ).
influ_cont( atenuacao(att4) , 76 ).
influ_cont( algoritmo(fir) , 77 ).
influ_cont( ordem(ord4) , 77 ).
influ_cont( largura(larg2) , 77 ).
influ_cont( algoritmo(fir) , 78 ).
influ_cont( ordem(ord4) , 78 ).
influ_cont( largura(larg3) , 78 ).
influ_cont( atenuacao(att1) , 78 ).
influ_cont( algoritmo(fir) , 79 ).
influ_cont( ordem(ord4) , 79 ).
```

```
influ_cont( largura(larg3) , 79 ).
influ_cont( atenuacao(att2) , 79 ).
influ_cont( algoritmo(fir) , 80 ).
influ_cont( ordem(ord4) , 80 ).
influ_cont( largura(larg3) , 80 ).
influ_cont( atenuacao(att3) , 80 ).
influ_cont( algoritmo(fir) , 81 ).
influ_cont( ordem(ord4) , 81 ).
influ_cont( largura(larg3) , 81 ).
influ_cont( atenuacao(att4) , 81 ).
influ_cont( algoritmo(fir) , 82 ).
influ_cont( ordem(ord4) , 82 ).
influ_cont( largura(larg4) , 82 ).
influ_cont( algoritmo(fir) , 83 ).
influ_cont( ordem(ord5) , 83 ).
influ_cont( largura(larg1) , 83 ).
influ_cont( atenuacao(att1) , 83 ).
influ_cont( algoritmo(fir) , 84 ).
influ_cont( ordem(ord5) , 84 ).
influ_cont( largura(larg1) , 84 ).
influ_cont( atenuacao(att2) , 84 ).
influ_cont( algoritmo(fir) , 85 ).
influ_cont( ordem(ord5) , 85 ).
influ_cont( largura(larg1) , 85 ).
influ_cont( atenuacao(att3) , 85 ).
influ_cont( algoritmo(fir) , 86 ).
influ_cont( ordem(ord5) , 86 ).
influ_cont( largura(larg1) , 86 ).
influ_cont( atenuacao(att4) , 86 ).
influ_cont( algoritmo(fir) , 87 ).
influ_cont( ordem(ord5) , 87 ).
influ_cont( largura(larg2) , 87 ).
influ_cont( algoritmo(fir) , 88 ).
influ_cont( ordem(ord5) , 88 ).
influ_cont( largura(larg3) , 88 ).
influ_cont( atenuacao(att1) , 88 ).
influ_cont( algoritmo(fir) , 89 ).
influ_cont( ordem(ord5) , 89 ).
influ_cont( largura(larg3) , 89 ).
influ_cont( atenuacao(att2) , 89 ).
influ_cont( algoritmo(fir) , 90 ).
influ_cont( ordem(ord5) , 90 ).
influ_cont( largura(larg3) , 90 ).
influ_cont( atenuacao(att3) , 90 ).
influ_cont( algoritmo(fir) , 91 ).
influ_cont( ordem(ord5) , 91 ).
influ_cont( largura(larg3) , 91 ).
influ_cont( atenuacao(att4) , 91 ).
influ_cont( algoritmo(fir) , 92 ).
influ_cont( ordem(ord5) , 92 ).
influ_cont( largura(larg4) , 92 ).
```

APENDICE A8

ARQUIVO OUTRAS.PRI

```
prior(1,0).
prior(2,0).
prior(3,0).
prior(4,0).
prior(5,0).
prior(6,0).
prior(7,0).
prior(8,0).
prior(9,0).
prior(10,0).
prior(11,0).
prior(12,0).
prior(13,0).
prior(14,0).
prior(15,0).
prior(16,0).
prior(17,0).
prior(18,0).
prior(19,0).
prior(20,0).
prior(21,0).
prior(22,0).
prior(23,0).
prior(24,0).
prior(25,0).
prior(26,0).
prior(27,0).
prior(28,0).
prior(29,0).
prior(30,0).
prior(31,0).
prior(32,0).
prior(33,0).
prior(34,0).
prior(35,0).
prior(36,0).
prior(37,0).
prior(38,0).
prior(39,0).
prior(40,0).
prior(41,0).
prior(42,0).
prior(43,0).
prior(44,0).
prior(45,0).
prior(46,0).
prior(47,0).
prior(48,0).
prior(49,0).
prior(50,0).
prior(51,0).
```

```
prior(52,0).
prior(53,0).
prior(54,0).
prior(55,0).
prior(56,0).
prior(57,0).
prior(58,0).
prior(59,0).
prior(60,0).
prior(61,0).
prior(62,0).
prior(63,0).
prior(64,0).
prior(65,0).
prior(66,0).
prior(67,0).
prior(68,0).
prior(69,0).
prior(70,0).
prior(71,0).
prior(72,0).
prior(73,0).
prior(74,0).
prior(75,0).
prior(76,0).
prior(77,0).
prior(78,0).
prior(79,0).
prior(80,0).
prior(81,0).
prior(82,0).
prior(83,0).
prior(84,0).
prior(85,0).
prior(86,0).
prior(87,0).
prior(88,0).
prior(89,0).
prior(90,0).
prior(91,0).
prior(92,0).
```

APENDICE B1

PROGRAMA DA INTERFACE DE VOLUNTARIAMENTO

```
%
%          Interface de Voluntariamento
%

:- segment(farqseg).

:- public questoes/0:far.

/* Conjunto de questões do voluntariamento */

questoes :-
        questao1,
        questao2,
        questao3,
        questao4,
        questao5,
        questao6.


questao1 :-
        create_popup(´´,(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Indique   o   tipo   de   estrutura   adequada   ao
projeto:$),
        tmove(3,4),
        write($[1] FIR$),
        tmove(4,4),
        write($[2] IIR$),
        tmove(5,4),
        write($[3] Nao sei (o sistema identificar  a estrutura)$),
        tmove(6,4),
        write($[4] Por que ? (ativa o módulo de explanação)$),
        repeat,
        tmove(8,4),
        write($Escolha opção ---> $),
        read_line(0,Q1),
        case([Q1 == $1$ -> armazena1f,
              Q1 == $2$ -> armazena1i,
              Q1 == $3$ -> armazena1s,
              Q1 == $4$ -> explanacao1,
              (Q1 == $S$ ; Q1 == $s$) -> halt
              ¦fail]).


armazena1f :-
        assertz(fato(estrutura(filtro_fir))),
        assertz(dado_entrada(estrutura(filtro_fir))),
        exit_popup.

armazena1i :-
        assertz(fato(estrutura(filtro_iir))),
```

```
        assertz(dado_entrada(estrutura(filtro_iir))),
        exit_popup,
        questao11.

armazenals :-
        assertz(fato(estrutura(nao_sei))),
        assertz(dado_entrada(estrutura(desconhecida))),
        exit_popup.

explanacao1 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
indique um tipo  de$),
        tmove(2,2),
        write($estrutura  adequada para o filtro, pois este  fato
tem importancia na$),
        tmove(3,2),
        write($determinação  do tipo de algoritmo para  o  filtro
desejado.No entanto,$),
        tmove(4,2),
        write($se  o  usuário desconhecer a  estrutura  adequada,
digite <3>, ou seja,$),
        tmove(5,2),
        write($nao  sei,  fazendo com que o sistema  determine  a
solução  baseado  em$),
        tmove(6,2),
        write($outros  fatos.  A estrutura FIR é  mais  utilizada
quando se deseja pre-$),
        tmove(7,2),
        write($servar a linearidade de fase enquanto que a IIR  é
usado quando   se$),
        tmove(8,2),
        write($deseja obter um filtro com de baixa ordem.$),
        tmove(10,2),
        write($     *** Aperte uma tecle para voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao11 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Indique o tipo de aproximação desejada:$),
        tmove(3,4),
        write($[1] Butterworth$),
        tmove(4,4),
        write($[2] Chebyshev I$),
        tmove(5,4),
        write($[3] Chebyshev II$),
        tmove(6,4),
        write($[4] Elíptico$),
        tmove(7,4),
        write($[5] Nao sei$),
        tmove(8,4),
```

```
        write($[6] Por que ? (ativa o módulo de explanação)$),
        repeat,
        tmove(10,4),
        write($Escolha opção ---> $),
        read_line(0,Q11),
        case([Q11 == $1$ -> armazena11_butter,
              Q11 == $2$ -> armazena11_cheby1,
              Q11 == $3$ -> armazena11_cheby2,
              Q11 == $4$ -> armazena11_elip,
              Q11 == $5$ -> armazena11_nao_sei,
              Q11 == $6$ -> explanacao11,
              (Q11 == $S$ ; Q11 == $s$) -> halt
              !fail]).

armazena11_butter :-
        assertz(fato(tipo_aprox(butterworth))),
        assertz(dado_entrada(tipo_aproximacao(butterworth))),
        exit_popup.

armazena11_cheby1 :-
        assertz(fato(tipo_aprox(chebyshev_1))),
        assertz(dado_entrada(tipo_aproximacao(chebyshev_1))),
        exit_popup.

armazena11_cheby2 :-
        assertz(fato(tipo_aprox(chebyshev_2))),
        assertz(dado_entrada(tipo_aproximacao(chebyshev_2))),
        exit_popup.

armazena11_elip :-
        assertz(fato(tipo_aprox(eliptico))),
        assertz(dado_entrada(tipo_aproximacao(eliptico))),
        exit_popup.

armazena11_nao_sei :-
        assertz(fato(tipo_aprox(não_sei))),
        assertz(dado_entrada(tipo_aproximacao(desconhecida))),
        exit_popup.

explanacao11 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
indique o tipo   de$),
        tmove(2,2),
        write($aproximação  desejada. A  aproximação  Butterworth
tem a  borda  pouco$),
        tmove(3,2),
        write($íngreme na transição e magnitude maximamente plana
;  Chebyshev - I$),
        tmove(4,2),
        write($tem  borda  mais íngreme que a Butterworth  e  tem
ripple na  faixa  de$),
        tmove(5,2),
        write($passagem;  Chebyshev II e idêntico ao I,porém  tem
```

```
ripple na faixa de$),
        tmove(6,2),
        write($rejeição;    Elíptico   apresenta    comportamento
equiripple na faixa   de$),
        tmove(7,2),
        write($passagem similar ao Chebyshev I e possui uma borda
mais aguda   que$),
        tmove(8,2),
        write($as demais. No entanto,se você desconhece o  melhor
tipo de aproxima-$),
        tmove(9,2),
        write($ção, tecle < 5 > e o sistema identificar .$),
        tmove(11,2),
        write($    *** Aperte uma tecle para voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao2 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Indique tipo de filtro desejado:$),
        tmove(3,4),
        write($[1] Passa-baixa$),
        tmove(4,4),
        write($[2] Passa-alta$),
        tmove(5,4),
        write($[3] Passa-faixa$),
        tmove(6,4),
        write($[4] Rejeita-faixa$),
        tmove(7,4),
        write($[5] Multi-faixa$),
        tmove(8,4),
        write($[6] Passa-tudo$),
        tmove(9,4),
        write($[7] Diferenciador$),
        tmove(10,4),
        write($[8] Transformador de Hilbert$),
        repeat,
        tmove(12,6),
        write($Escolha opção ---> $),
        read_line(0,Q2),
        case([Q2 == $1$ -> armazena2pb,
              Q2 == $2$ -> armazena2pa,
              Q2 == $3$ -> armazena2pf,
              Q2 == $4$ -> armazena2rf,
              Q2 == $5$ -> (verifica2_estrutura,armazena2mf),
              Q2 == $6$ -> (verifica2_estrutura,armazena2pt),
              Q2 == $7$ -> (verifica2_estrutura,armazena2di),
              Q2 == $8$ -> (verifica2_estrutura,armazena2th),
              (Q2 == $S$ ; Q2 == $s$) -> halt
              |fail]).

verifica2_estrutura :-
        call(fato(estrutura(E2))),
```

```
        ifthen(E2 == filtro_iir,fail).

armazena2pb :-
        assertz(fato(filtro(passa_baixa))),
        assertz(dado_entrada(tipo_filtro(passa_baixa))),
        exit_popup.

armazena2pa :-
        assertz(fato(filtro(passa_alta))),
        assertz(dado_entrada(tipo_filtro(passa_alta))),
        exit_popup.

armazena2pf :-
        assertz(fato(filtro(passa_faixa))),
        assertz(dado_entrada(tipo_filtro(passa_faixa))),
        exit_popup.

armazena2rf :-
        assertz(fato(filtro(rejeita_faixa))),
        assertz(dado_entrada(tipo_filtro(rejeita_faixa))),
        exit_popup.

armazena2mf :-
        assertz(fato(filtro(multi_faixa))),
        assertz(dado_entrada(tipo_filtro(multi_faixa))),
        exit_popup.

armazena2pt :-
        assertz(fato(filtro(passa_tudo))),
        assertz(dado_entrada(tipo_filtro(passa_tudo))),
        exit_popup.

armazena2di :-
        assertz(fato(filtro(diferenciador))),
        assertz(dado_entrada(tipo_filtro(diferenciador))),
        exit_popup.

armazena2th :-
        assertz(fato(filtro(transf_hilbert))),
        assertz(dado_entrada(tipo_filtro(tranformador_hilbert))),
        exit_popup.

questao3 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Indique a área de aplicação:$),
        tmove(3,4),
        write($[1] Voz$),
        tmove(4,4),
        write($[2] Audio$),
        tmove(5,4),
        write($[3] Comunicações$),
        tmove(6,4),
        write($[4] Biomédica$),
        tmove(7,4),
```

```
        write($[5] Outras$),
        tmove(8,4),
        write($[6] Por que ? (ativa o módulo de explanação),
        repeat,
        tmove(10,6),
        write($Escolha opção ---> $),
        read_line(0,Q3),
        case([Q3 == $1$ -> armazena3v,
              Q3 == $2$ -> nao_implementado,
              Q3 == $3$ -> nao_implementado,
              Q3 == $4$ -> nao_implementado,
              Q3 == $5$ -> armazena3o,
              Q3 == $6$ -> explanacao3,
              (Q3 == $S$ ; Q3 == $s$) -> halt
              |fail]).

armazena3v :-
        assertz(fato(area(voz))),
        assertz(dado_entrada(area_aplicacao(voz))),
        repeat,
        call(fato(estrutura(M1))),
        case([M1 == filtro_iir -> (exit_popup,
                                   questao31,
                                   questao32,
                                   questao33),
              M1 == nao_sei -> (exit_popup,questao32,questao33),
              M1 == filtro_fir -> (exit_popup,questao33)
              |fail]).

armazena3o :-
        assertz(fato(area(outras))),
        assertz(dado_entrada(area_aplicacao(outras))),
        exit_popup,
        questao32,
        questao33.

nao_implementado :-
        create_popup('',(16,3),(22,76),(112,7)),
        tmove(1,2),
        write($A base de conhecimento deste sistema é dividida em
células.$),
        tmove(2,2),
        write($A célula correspondente a esta área não foi
implementada.$),
        tmove(4,2),
        write($      *** Aperte uma tecla p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

explanacao3 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
indique a área  de$),
```

```
        tmove(2,2),
        write($aplicação,  pois  este  fato  tem  importancia  na
determinação do algo -$),
        tmove(3,2),
        write($ritmo.Caso  a área de aplicação do usuário nao  se
enquadre dentre as$),
        tmove(4,2),
        write($existentes no menu, digite < 5 >.$),
        tmove(6,2),
        write($      *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao31 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Tem  certeza que a estrutura mais adequada  é  IIR
?$),
        tmove(3,4),
        write($[1] Sim$),
        tmove(4,4),
        write($[2] Não$),
        tmove(5,4),
        write($[3] Por que ? (ativa o módulo de explanação)$),
        repeat,
        tmove(7,6),
        write($Escolha opção ---> $),
        read_line(0,Q31),
        case([Q31 == $1$ -> exit_popup,
              Q31 == $2$ -> (altera31_estrutura,exit_popup),
              Q31 == $3$ -> explanacao31,
              (Q31 == $S$ ; Q31 == $s$) -> halt
              |fail]).

altera31_estrutura :-
        retract(fato(estrutura(filtro_iir))),
        retract(dado_entrada(estrutura(filtro_iir))),
        call(fato(tipo_aprox(E31))),
        retract(fato(tipo_aprox(E31))),
        call(dado_entrada(tipo_aproximacao(D31))),
        retract(dado_entrada(tipo_aproximacao(D31))),
        assertz(fato(estrutura(filtro_fir))),
        assertz(dado_entrada(estrutura(filtro_fir))).

explanacao31 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
confirme o tipo de$),
        tmove(2,2),
        write($estrutura  mais  adequada,  pois  na  maioria  das
aplicações de proces -$),
        tmove(3,2),
        write($samento de voz se utiliza estrutura FIR. Se para a
```

```
aplicação dese -$),
        tmove(4,2),
        write($jada a indicação for IIR, digite <1>; Caso
contrário, digite <2>.$),
        tmove(6,2),
        write($      *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao32 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Necessita de fase linear ?$),
        tmove(3,4),
        write($[1] Sim$),
        tmove(4,4),
        write($[2] Não$),
        tmove(5,4),
        write($[3] Por que ? (ativa o módulo de explanação)$),
        repeat,
        tmove(7,6),
        write($Escolha opção ---> $),
        read_line(0,Q32),
        case([Q32 == $1$ -> armazena32s,
              Q32 == $2$ -> armazena32n,
              Q32 == $3$ -> explanacao3,
              (Q32 == $S$ ; Q32 == $s$) -> halt
              ¦fail]).

armazena32s :-
        assertz(fato(fase(linear))),
        assertz(dado_entrada(fase(linear))),
        exit_popup.

armazena32n :-
        assertz(fato(fase(nao_linear))),
        assertz(dado_entrada(fase(nao_linear))),
        exit_popup.

explanacao32 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação: Está sendo solicitado que o usuário
informe se a linea-$),
        tmove(2,2),
        write($ridade de fase tem ou não importancia porque este
fato irá auxiliar$),
        tmove(3,2),
        write($na determinação do algoritmo que satisfaça essa
característica. Se$),
        tmove(4,2),
        write($para a aplicação houver necessidade de preservar a
linearidade de$),
        tmove(5,2),
```

```
        write($fase, digite <1>; Caso contrário, digite <2>.$),
        tmove(7,2),
        write($      *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao33 :-
        create_popup(´´,(5,2),(23,77),(7,7)),
        tmove(1,4),
        write($Deseja que os coeficientes sejam quantizados ?$),
        tmove(3,4),
        write($[1] Sim$),
        tmove(4,4),
        write($[2] Não$),
        tmove(5,4),
        write($[3] Por que ? (ativa o módulo de explanação)$),
        repeat,
        tmove(7,6),
        write($Escolha opção -- > $),
        read_line(0,Q33),
        case([Q33 == $1$ -> armazena33_sim,
              Q33 == $2$ -> armazena33_nao,
              Q33 == $3$ -> explanacao33,
              (Q33 == $S$ ; Q33 == $s$) -> halt
              ¦fail]).

armazena33_sim :-
        verifica33_filtro,
        verifica33_estrutura,
        exit_popup.

verifica33_filtro :-
        call(fato(filtro(F33))),
        ifthen((F33 == passa_tudo;
                F33 == multi_faixa;
                F33 == diferenciador;
                F33 == transf_hilbert),
                fail).

verifica33_estrutura :-
        call(fato(estrutura(E33))),
        case([E33 == filtro_fir ->
                    (assertz(fato(quantiz(sim))),
                     assertz(dado_entrada(deseja_quantizacao))),
              (E33 == filtro_iir;E33 == nao_sei) ->
                     solicita_num_bits
               ¦fail]).
solicita_num_bits :-
        tmove(9,4),
        write($Entre com o número de bits desejado --> $),
        read_line(0,A33),
        int_text(P33,A33),
        assertz(dado_entrada(num_bits(P33))),
        assertz(fato(quantiz(sim))),
```

```
        assertz(dado(deseja_quantizacao)).

armazena33_nao :-
        assertz(fato(quantiz(nao))),
        assertz(dado_entrada(nao_deseja_quantizacao)),
        exit_popup.

explanacao33 :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:Está sendo solicitado que o usuário
informe se deseja ou$),
        tmove(2,2),
        write($não ter os valores dos coeficientes do filtro
quantizado. Se a es -$),
        tmove(3,2),
        write($trutura indicada for FIR, então o sistema fará o
projeto em função$),
        tmove(4,2),
        write($do valor de N desejado, porém se for IIR, ele
solicitar ao usuário$),
        tmove(5,2),
        write($o tamanho da palavra (número de bits) para qual o
usuário deseja a$),
        tmove(6,2),
        write($quantização.$),
        tmove(8,2),
        write($     *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao4 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,2),
        write($Indique a ordem(N) desejada do filtro:$),
        tmove(5,2),
        write($Observação:$),
        tmove(6,2),
        write($Se filtro IIR: 1 < N =< 10 (número de seções de
2a.ordem)$),
        tmove(7,2),
        write($Se filtro FIR: 3 =< N =< 128 (tamanho do
filtro)$),
        tmove(8,2),
        write($Se desejar que o sistema determine o N, digite < m
> e <enter>$),
        tmove(9,2),
        write($(Essa determinação do valor de N é feita apenas
para filtros$),
        tmove(10,2),
        write($passa-baixa, alta, faixa e rejeita-faixa)$),
        tmove(12,2),
        write($Se desejar explanação, digite <0> e <enter>$),
        repeat,
```

```
        tmove(3,6),write($Ordem do filtro --> $),
        read_line(0,Q4),
        case([Q4 == $0$ -> explanacao4(P4),
              (Q4 == $m$ ; Q4 == $M$) ->
                     (verifica4_filtro,aciona_estinf),
              (Q4 == $S$ ; Q4 == $s$) -> halt
              ¦converte_ordem(Q4)]).

verifica4_filtro :-
        call(fato(filtro(T1))),
        ifthen((T1 == multi_faixa;
                T1 == diferenciador;
                T1 == transf_hilbert;
                T1 == passa_tudo),
                fail).

aciona_estinf :-
        create_popup(´´,(16,3),(22,76),(112,7)),
        tmove(1,2),
        write($Neste  ponto, ser acionado o  programa  estinf.exe
para que ele$),
        tmove(2,2),
        write($forneça o valor de N (ordem do filtro).$),
        tmove(4,2),
        write($       *** Aperte uma tecla p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

converte_ordem(Q4) :-
        int_text(P4,Q4),
        ifthenelse((P4 > 1 , P4 < 129),verifica_ordem(P4),fail).

verifica_ordem(P4) :-
        ver_ord1(P4);
        ver_ord2(P4);
        ver_ord3(P4);
        ver_ord4(P4);
        ver_ord5(P4).

ver_ord1(P4) :-
        (P4 < 11),
        call(fato(estrutura(T2))),
        (T2 == filtro_iir;T2 == nao_sei),
        assertz(fato(ordem(ord1))),
        assertz(dado_entrada(ordem(P4))),
        exit_popup.

ver_ord2(P4) :-
        ((P4 > 2 , P4 < 31) ; P4 == 31),
        call(fato(estrutura(T2))),
        (T2 == filtro_fir; T2 == nao_sei),
        assertz(fato(ordem(ord2))),
        assertz(dado_entrada(ordem(P4))),
        exit_popup.
```

```
ver_ord3(P4) :-
        ((P4 >2 , P4 < 41) ; P4 == 41),
        call(fato(estrutura(T2))),
        (T2 == filtro_fir; T2 == nao_sei),
        assertz(fato(ordem(ord3))),
        assertz(dado_entrada(ordem(P4))),
        exit_popup.

ver_ord4(P4) :-
        ((P4 > 2 , P4 < 51) ; P4 == 51),
        call(fato(estrutura(T2))),
        (T2 == filtro_fir; T2 == nao_sei),
        assertz(fato(ordem(ord4))),
        assertz(dado_entrada(ordem(P4))),
        exit_popup.

ver_ord5(P4) :-
        ifthenelse(((P4 > 2 , P4 < 129),
                    call(fato(estrutura(T2))),
                    (T2 == filtro_fir; T2 == nao_sei)),
                   (assertz(fato(ordem(ord5))),
                    assertz(dado_entrada(ordem(P4))),exit_popup),
                    fail).

explanacao4(P4) :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Se  a estrutura  escolhida  for  IIR,
dever ser  digitado$),
        tmove(2,2),
        write($um  número referente a ordem desejada, onde 1 <  N
=< 10, sendo N   o$),
        tmove(3,2),
        write($número  de  seções de 2a. ordem;  Se  a  estrutura
escolhida for FIR   ,$),
        tmove(4,2),
        write($dever ser digitado um número referente ao  tamanho
do filtro,  onde$),
        tmove(5,2),
        write($3 =< N =<128 ; Caso desconheça o dado solicitado ,
dever  digitar$),
        tmove(6,2),
        write($< m >, pois dessa forma o sistema ir determinar  o
valor de N, con-$),
        tmove(7,2),
        write($siderando os demais parâmetros.$),
        tmove(9,2),
        write($      *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao5 :-
        create_popup('',(5,2),(23,77),(7,7)),
```

```
        tmove(1,4),
        write($Indique a atenuação desejada na faixa de rejeição:$),
        tmove(5,4),
        write($Obs.: P/ explanação, digite < 0 > e$),
        tmove(6,10),
        write($p/ filtros s/ faixa de rejeição, digite < m >.$),
        repeat,
        tmove(3,4),
        write($atenuação(dB) --> $),
        read_line(0,Q5),
        case([Q5 == $0$ -> explanacao5(Q5),
              (Q5 == $M$ ; Q5 == $m$) -> verifica5_tipo_filtro,
              (Q5 == $S$ ; Q5 == $s$) -> halt
              !converte_atenuacao(Q5)]).

verifica5_tipo_filtro :-
        call(fato(filtro(V5))),
        ifthenelse((V5 == multi_faixa;
                    V5 == diferenciador;
                    V5 == transf_hilbert;
                    V5 == passa_tudo),
                    exit_popup,fail).

converte_atenuacao(Q5) :-
        int_text(P5,Q5),
        ifthenelse((P5 > 0 , P5 < 200),confirma5_tipo_filtro(P5),fail).

confirma5_tipo_filtro(P5) :-
        call(fato(filtro(M5))),
        ifthenelse((M5 == passa_baixa;
                    M5 == passa_alta;
                    M5 == passa_faixa;
                    M5 == rejeita_faixa),
                    verifica_atenuacao(P5),fail).

verifica_atenuacao(P5) :-
        ver_att1(P5);
        ver_att2(P5);
        ver_att3(P5);
        ver_att4(P5).

ver_att1(P5) :-
        (P5 < 20),
        assertz(fato(atenuacao(att1))),
        assertz(dado_entrada(atenuacao(P5))),
        exit_popup.

ver_att2(P5) :-
        (P5 < 30),
        assertz(fato(atenuacao(att2))),
        assertz(dado_entrada(atenuacao(P5))),
        exit_popup.

ver_att3(P5) :-
        (P5 < 44),
```

```
        assertz(fato(atenuacao(att3))),
        assertz(dado_entrada(atenuacao(P5))),
        exit_popup.

ver_att4(P5) :-
        (P5 > 44;P5 == 44),
        assertz(fato(atenuacao(att4))),
        assertz(dado_entrada(atenuacao(P5))),
        exit_popup.

explanacao5(P5) :-
        create_popup('',(6,3),(22,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
informe a atenuação$),
        tmove(2,2),
        write($desejada  na faixa de rejeição. No entanto,  se  o
filtro selecionado$),
        tmove(3,2),
        write($não   tiver   essa   característica   (passa-tudo,
diferenciador ou trans -$),
        tmove(4,2),
        write($formador de Hilbert), então digite  < m >.$),
        tmove(6,2),
        write($      *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.

questao6 :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(1,2),
        write($Quantas faixas de atuação tem o filtro ?$),
        tmove(5,2),
        write($OBS.: Entre c/ valores de frequência$),
        tmove(6,2),
        write($em kHz. Para explanação , digite <0>$),
        tmove(7,2),
        write($e <enter>.$),
        repeat,
        tmove(3,2),
        write($Número de faixas --> $),
        read_line(0,Ai6),
        int_text(Ao6,Ai6),
        case([(Ao6 > 0, Ao6 < 12) ->
                ((P6 is (2 * Ao6)),
                cria_arq_larg(P6),
                verifica_num_faixas(P6,H1)),
                Ao6 = 0 -> explanacao6
                ¦fail]).

cria_arq_larg(P6) :-
        create(H1,'freq.dat'),
        freq_amost(H1,P6),
        le_escreve_freq(H1,P6),
```

```
        confirma_freq(H1),
        close(H1).


verifica_num_faixas(P6,H1) :-
        ifthenelse((P6 > 2 , P6 < 11),
                   (calcula_largura(H1),exit_popup),
                   exit_popup).

freq_amost(H1,P6) :-
        repeat,
        mostra_freq_amostragem,
        read_line(0,Fsi),
        float_text(Fs,Fsi,_),
        writeq(H1,Fs).

mostra_freq_amostragem :-
        tmove(8,2),
        write($Entre com o valor da frequência$),
        tmove(9,2),
        write($de amostragem(kHz) ---> Fs = $).

le_escreve_freq(H1,P6) :-
        ctr_set(0,1),
        repeat,
        ctr_inc(0,Xc),
        Xl is Xc + 2,
        mostra_freq(P6,Xc,Xl),
        le_escreve_arq(H1,P6,Xc,Xl),
        Xc = P6.

mostra_freq(P6,Xc,Xl) :-
        (Xc =< 11, Xc =< P6),
        tmove(Xl,40),
        write(f(Xc)),
        write($=$).

mostra_freq(P6,Xc,Xl) :-
        (Xc > 11 , Xc =< 22 , Xc =< P6),
        Xl2 is Xl - 11,
        tmove(Xl2,56),
        write(f(Xc)),
        write($=$).

le_escreve_arq(H1,P6,Xc,Xl) :-
        (Xc =< 11 , Xc =< P6),
        tmove(Xl,46),
        read_line(0,Fi),
        float_text(Fo,Fi,_),
        nl(H1),
        writeq(H1,Fo).

le_escreve_arq(H1,P6,Xc,Xl) :-
        (Xc > 11 , Xc =< 22 , Xc =< P6),
        Xl2 is Xc - 9,
```

```
        tmove(X12,62),
        read_line(0,Fi),
        float_text(Fo,Fi,_),
        nl(H1),
        writeq(H1,Fo).

confirma_freq(H1) :-
        tmove(15,40),
        write($Confirma frequências(s/n) ->$),
        keyb(Cf,_),
        case([(Cf == 115 ; Cf == 83) -> true,
              (Cf == 110 ; Cf == 78) -> (seek(H1,1,bof,_),
                                         ctr_set(0,1),
                                         fail)
              ¦fail]).

calcula_largura(H1) :-
        open(H1,´freq.dat´,r),
        seek(H1,0,bof,_),
        read_line(H1,Fsi),
        float_text(Fs,Fsi,_),
        read_line(H1,F1i),
        float_text(F1,F1i,_),
        read_line(H1,F2i),
        float_text(F2,F2i,_),
        read_line(H1,F3i),
        float_text(F3,F3i,_),
        Df is (F3-F2),
        Dv is Df/Fs,
        L is abs(Dv),
        close(H1),
        verifica_largura(L).

verifica_largura(L) :-
        ver_larg1(L);
        ver_larg2(L);
        ver_larg3(L);
        ver_larg4(L).

ver_larg1(L) :-
        (L < 0.03),
        assertz(fato(largura(larg1))),
        assertz(dado_entrada(largura_faixa_transicao(L))).

ver_larg2(L) :-
        (L >= 0.03,L < 0.05),
        assertz(fato(largura(larg2))),
        assertz(dado_entrada(largura_faixa_transicao(L))).

ver_larg3(L) :-
        (L >= 0.05,L < 0.065),
        assertz(fato(largura(larg3))),
        assertz(dado_entrada(largura_faixa_transicao(L))).

ver_larg4(L) :-
```

```
        (L >= 0.065),
        assertz(fato(largura(larg4))),
        assertz(dado_entrada(largura_faixa_transicao(L))).

explanacao6 :-
        create_popup('',(6,3),(23,76),(112,7)),
        tmove(1,2),
        write($Explanação:  Está sendo solicitado que  o  usuário
informe  o  número$),
        tmove(2,2),
        write($de  faixas  existentes no  filtro(considerando  as
faixas de passagem e$),
        tmove(3,2),
        write($rejeição existentes). Para os filtros  passa-tudo,
diferenciador   e$),
        tmove(4,2),
        write($transformador   de   Hilbert,  digite   <   1   >,
correspondente a uma faixa.$),
        tmove(5,2),
        write($Para os filtros passa-baixa e passa-alta, digite <
2 >, correspon -$),
        tmove(6,2),
        write($dente  a duas faixas ( uma de passagem e outra  de
rejeição). Para os$),
        tmove(7,2),
        write($filtros  multi_faixas, tem-se que o número  máximo
de faixas de pas -$),
        tmove(8,2),
        write($sagem ou rejeição é 5 (cinco).$),
        tmove(10,2),
        write($     *** Aperte uma tecle p/voltar ***$),
        keyb(_,_),
        exit_popup,
        fail.
```

APENDICE B2

PROGRAMA DO MODULO DE INFERENCIA

```
%       Módulo de Inferência
%

:- module io.

:- segment(farioseg).

:- public main/0:far.

:- extrn questoes/0:far.

:- visible
        guarde_f/1,
        retira_f/1,
        guarde_d/1,
        como/0.

/* Definicao da tela de inicializacao */

main :-
        background_window,
        entrada_sys,
        delete_window(back).

background_window :-
        define_window(back,'',(0,0),(24,79),(7,0)),
        current_window(_,back).

entrada_sys :-
        create_popup('SIPREX',(0,0),(24,79),(7,-7)),
        reverse_text(0,10,$SELEÇAO DE ALGORITMOS PARA PROJETOS DE
FILTROS DIGITAIS$),
        reverse_text(2,10,$     Laboratorio   de   Automacao    e
Processamento de Sinais   $),
        reverse_text(3,10,$           Depto.    de    Engenharia
Eletrica/CCT/UFPb         $),
        reverse_text(4,10,$        Versao 1.0 - 1991 -  da  Silva
Filho, A. M.        $),
        tmove(19,12),
        write($Tecle <enter> p/ iniciar ou <Esc> p/ sair$),
        repeat,
        le_iniciar(P0,_),
        case([P0  ==  13  ->  (questoes,confirma_dados),  %   faz
voluntariamento
              P0  == 27 -> (exit_popup,halt)          %  abandona
sistema
              ¦fail]).

reverse_text(Y,X,Text) :-
        list_text(L,Text),
        length(L,N),
```

```
        tmove(Y,X),
        wa(N,112),
        write(Text).

le_iniciar(P0,_) :-
        repeat,
        keyb(P0,_),
        membro(P0,[13,27]).

membro(X,[X|_]) :- !,gc.
membro(X,[_|Xs]) :-
        membro(X,Xs).

confirma_dados :-
        create_popup('',(5,2),(23,77),(7,7)),
        tmove(2,0),
        listing(dado_entrada),
        tmove(0,2),
        write($Confirma os dados abaixo:  [s/n] $),
        repeat,
        le_confirma(F2,_),
        case([F2  == 115 -> ativa_nucleo,      % chama modulo  de
inferencia
              F2   ==  110  ->  retorna_questoes    %  faz   novo
voluntariamento
              |fail]).

le_confirma(F2,_) :-
        repeat,
        keyb(F2,_),
        membro(F2,[115,110]).

ativa_nucleo :-
        create_popup('',(6,3),(22,76),(7,7)),
        retract(fato(area(W))),
        assertz(W),
        assertz(fato(area(W))),
        gc,
        tmove(1,4),
        write($Solução = $),
        call(W),
        execute(W),
        tmove(4,4),
        write($[1]  Como  ? (O sistema mostrar as  regras  usadas
p/solução)$),
        tmove(5,4),
        write($[2] Fim (abandona o sistema)$),
        repeat,
        tmove(3,4),
        write($Selecione uma opção:$),
        le_escolha(Esc,_),
        case([Esc == 49 -> explanacao_como,
              Esc == 50 -> saida_sistema
              |fail]).
```

```
le_escolha(Esc,_) :-
        repeat,
        keyb(Esc,_),
        membro(Esc,[49,50]).
/* Mostra explanação ref. a solução encontrada e abandona o sistema */

explanacao_como :-
        create_popup('EXPLANAÇAO                DA                SOLUÇAO
ENCONTRADA',(9,4),(21,75),(112,7)),
        como,
        nl,
        write($              *** Aperte uma tecla p/ sair ***$),
        keyb(_,_),
        exit_popup,
        exit_popup,
        exit_popup,
        halt.

/* Abandona o sistema e limpa memória de trabalho */

saida_sistema :-
        abolish(fato/1),
        abolish(dado_entrada/1),
        abolish(dado/1),
        abolish(executavel/1),
        exit_popup,
        exit_popup,
        exit_popup,
        halt.

retorna_questoes :-
        abolish(fato/1),
        abolish(dado_entrada/1),
        delete('freq.dat'),
        exit_popup,
        exit_popup,
        entrada_sys.


/*  Mecanismo de Inferência  */


:- op(900,fx,se).
:- op(600,xfy,':').
:- op(800,xfx,entao).
:- op(750,xfy,e).
:- op(400,fx,i).
:- op(300,fx,[guarde_f,guarde_d]).



/*  Instancia um predicado fato ou diagnóstico, depois  instancia
```

este com os predicados **influ_cont** para identificar a regra infuenciada. Em seguida, atualiza o predicado **contagem** da regra correspondente e verifica o valor do conectivo C. Se C=1, cria uma cláusula executável, senão chamará a rotina disj. */

```
escalonador :-
        retract(fato(D)),
        assertz(dado(D)),
        call(influ_cont(D,N)),
        retract(influ_cont(D,N)),
        call(contagem(C1,C2,N,C)),
        ifthenelse(C == 1, (assertz(executavel(N)), exec1(D)),
        disj(D,N)).

escalonador :-
        call(diagnostico(D)),
        call(influ_cont(D,N)),
        assertz(dado(D)),
        retract(diagnostico(D)),
        retract(influ_cont(D,N)),
        call(contagem(C1,C2,N,C)),
        ifthenelse(C == 1, (assertz(executavel(N)), exec1(D)),
        disj(D,N)).

escalonador.


exec1(D) :-
        call(influ_cont(D,N)),
        retract(influ_cont(D,N)),
        call(contagem(C1,C2,N,C)),
        ifthenelse(C  == 1,  (assertz(executavel(N)),  exec1(D)),
        disj(D,N)).

exec1(X) :- escalonador,gc.


disj(D,N) :-
        retract(contagem(C1,C2,N,C)),
        C3 is C2 - 1,
        assertz(contagem(C1,C3,N,C)),
        ifthenelse(C3  == 0, (assertz(executavel(N)),  exec1(D)),
        exec1(D)),
        !,gc.


/*  Lê  uma  lista com número e prioridade da regra  e  aciona  a
primeira da lista */


acionador([[A,B]|Ys]) :-
        aciona(B),
        not(call(modificou)),
        acionador(Ys),!,gc.
```

```
acionador([]).


/* Aciona uma regra buscando os fatos na base e executando as
ações */

aciona(Y) :-
        not(call(executou(Y))),
        call(regra(Y,se Conds entao Acoes)),
        asserta(regra_corrente(Y)),
        call(Acoes),
        assertz(executou(Y)),
        retract(prior_exec(Y,X)).

aciona(Y) :- assertz(para).

/* Lista todos os diagnósticos que estão nos predicados
diagnóstico */

diagnosticador :-
        call(diagnostico(X)),
        call(X),
        write(X), nl,
        fail.

diagnosticador.

guarde_f(X) :- call(X).
guarde_f(X) :- assertz(fato(X)), call(modificou).
guarde_f(_) :- assertz(modificou).

retira_f(X) :- call(X).
retira_f(X) :- retract(fato(X)).

guarde_d(X) :- call(X).
guarde_d(X) :-
        assertz(diagnostico(X)),
        assertz(X),
        call(modificou).
guarde_d(_) :- assertz(modificou).


/* instancia a célula da base de conhecimento com o nome
correspondente a uma área a fim de carregar a referida célula,
ou seja, parte da base de conhecimento na memória de trabalho */


execute(W) :-
        concat(W,$.reg$,R),
        concat(W,$.inf$,I),
        concat(W,$.pri$,P),
        concat(W,$.con$,C),
        [R, I, P, C],
        assertz(modificou),
```

```
        passo.


continuo :-
        call(modificou),
        escalonador,
        exec_prior,
        findall([Y,X],
        prior_exec(X,Y), L),
        sort(L,L2),
        invertel(L2,[],L1),
        retract(modificou),
        call(acionador(L1)),
        continuo.


/* Escalona todas a regras executáveis, criando uma lista de
regras executáveis em ordem decrescente de prioridade e aciona
todas as regras da lista */


continuo :- diagnosticador, !,gc.

passo :-
        call(modificou),
        escalonador,
        exec_prior,
        passo2,!,gc.

passo :- diagnosticador,!,gc.

passo2 :-
        findall([Y,X],
        prior_exec(X,Y), L),
        sort(L,L2),
        invertel(L2,[],L1),
        retract(modificou),
        socabeca(L1),
        passo,!,gc.

passo2 :- diagnosticador, !,gc.

socabeca([[A,B]¦Ys]) :-
        aciona(B),
        socabeca2(Ys),!, gc.

socabeca([]) :- !,gc.

socabeca2([[A,B]¦Ys]) :-
        not(call(modificou)),
        aciona(B),
        socabeca2(Ys),!,gc.
socabeca2([[A,B]¦Ys]) :- !,gc.
```

```
socabeca2([]) :- !,gc.

exec_prior :-
        call(executavel(X)),
        call(prior(X,Y)),
        retract(executavel(X)),
        assertz(prior_exec(X,Y)),
        exec_prior.

exec_prior.

invertel([X|Xs], Acc,Ys) :-
        invertel(Xs,[X|Acc],Ys).

invertel([],Ys,Ys).

lista_executaveis :-
        call(prior_exec(X,Y)),
        write(X),
        retract(prior_exec(X,Y)),
        nl, fail.

/*  Mostra as regras e fatos que foram utilizados para  encontrar
um diagnóstico */

como :-
        call(executou(X)),
        write('Pela(s) regra(s) '),
        mostra_regras, nl,
        write('e baseado nos dados: '),
        mostra_dados, nl,
        write('concluiu-se que: '),
        diagnosticador.

como :-
        write('Atualmente  nao diponho de dados suficientes  para
        responder.').

mostra_regras :-
        call(executou(X)),
        call(regra(X,Y)),
        write('R'), write(X),
        write(','),
        write(Y), nl,
        write('e'), fail.

mostra_regras.

mostra_dados :-
        call(dado(X)),
        write(X),
        write(', '), fail.
mostra_dados.
```