Diagnose em Sistemas de Potência utilizando Lógica Não-monotônica e Redes Neurais

Victor Navarro Araújo Lemos da Silva

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por :

Gerson Zaverucha

Prof. Gerson Zaverucha, Ph. D.
(Presidente)

Luis Alfredo Vidal de Carvalho

Prof. Luís Alfredo Vidal de Carvalho, D. Sc.

Sheila Regina Murgel Veloso

Profa. Sheila Regina Murgel Veloso, D. Sc.

Marley Maria Bernardes Rebuzzi Vellasco

Profa. Marley Maria Bernardes Rebuzzi Vellasco, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1994

SILVA, VICTOR NAVARRO ARAÚJO LEMOS DA SILVA

Diagnose em Sistemas de Potência utilizando Lógica Não-monotônica e Redes Neurais [Rio de Janeiro] 1994

X, 190p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1994)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1- Lógica Não-monotônica 2- Redes Neurais 3- Sistemas de Potência 4- Diagnose de Falhas

I. COPPE/UFRJ II. Título (Série).

A meus pais

Agradecimentos

Agradeço a todos, que direta e indiretamente contribuiram para realização deste trabalho, dentre os quais destaco:

- Os amigos do CEPEL, em especial: Guilherme (meu orientador "e-mail"), Sérgio Lúcio, Aroldo, Fatima, Lima, Marcello e Valdir.
- Os engenheiros da ELETROSUL, Carlos Alberto Ferreira e Luis Francisco, pela ajuda na descrição da operação de sistemas de potência.
- Os professores Gerson e Luís Alfredo pela orientação acadêmica.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M. Sc.).

Diagnose em Sistemas de Potência utilizando Lógica Não-monotônica e Redes Neurais

Victor Navarro Araújo Lemos da Silva

Abril de 1994

Orientadores: Prof. Gerson Zaverucha e Prof. Luís Alfredo Vidal de Carvalho

Programa: Engenharia de Sistemas e Computação

A crescente complexidade dos Sistemas de Potência vem exigindo o desenvolvimento de técnicas cada vez mais sofisticadas de supervisão, controle e automação, entre as quais incluem-se os sistemas para diagnóstico de falhas.

Este trabalho analisa a utilização de técnicas de Inteligência Artificial na Diagnose de um Sistema de Potência, pretendendo auxiliar os operadores destes sistemas a determinar as causas dos distúrbios que possam surgir.

Com o propósito de verificar a viabilidade desta proposta, estudamos comparativamente a aplicação de duas técnicas: a Lógica Não-monotônica e a Rede Neural, no desenvolvimento de um sistema inteligente de diagnóstico de falhas.

Para tal, fizemos um estudo comparativo dos dois tipos mais conhecidos de Diagnose baseada em Lógica : a Abdutiva e a baseada na Consistência. A partir da escolha da Diagnose Abdutiva, utilizamos o Sistema Theorist como ferramenta para implementação deste tipo de diagnose. Com este objetivo, implementamos um interpretador e um compilador em Prolog para este Sistema.

Em relação a Rede Neural, escolhemos o modelo mais conhecido para aplicações em computação neural que é o modelo "Back-propagation". A partir desta escolha, utilizamos o software comercial "NeuralWorks" como ferramenta de simulação do comportamento deste modelo.

Estas duas técnicas foram então implementadas em um sistema elétrico simplificado de uma usina e os resultados obtidos comprovam a validade destes métodos e servem de base para o desenvolvimento de um sistema híbrido e para a implementação da Diagnose em um Sistema de Potência real.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

Nonmonotonic Logic and Artificial Neural Networks for Power Systems Diagnosis

Victor Navarro Araújo Lemos da Silva
April , 1994

Thesis supervisors: Prof. Gerson Zaverucha e Prof. Luís Alfredo Vidal de Carvalho

Department: Systems Engineering and Computer Science

The growing complexity in Power Systems has required the development of sophisticated techniques for supervision, control and automation. Among them, we find systems for fault diagnosis.

This work analyses the application of Artificial Intelligence techniques to help a power system operator to diagnose the fault during a disturbance.

In order to verify the feasibility of the proposed system, we make a comparative study about the application of two techniques in the development of an intelligent system for fault diagnosis: Nonmonotonic Logic and Artificial Neural Networks.

Towards this goal, we compare two logic-based models of diagnosis: Abductive Diagnosis and Consistency-based Diagnosis. By choosing the Abductive Diagnosis, we use Poole's Theorist System as a tool to implement this kind of diagnosis. We implement an interpreter and a compiler both written in Prolog for this system.

We use the "Back-propagation" model and the commercial software "NeuralWorks" to simulate the Neural Networks.

These two techniques are implemented in a simplified power system generation plant and the obtained results show the validity of both methods. It serves as a basis of development of a hybrid system towards the implementation of a real Power Systems Diagnosis.

Índice

Capítulo I

Introdução

Capítulo II

Operação de Sistemas de Potência

Capítulo III

Lógica aplicada à Diagnose

Capítulo IV

Redes Neurais

Capítulo V

Conclusões

Capítulo I

Introdução

I.1 - A Importância da Diagnose em Sistemas de Potência :

Os Sistemas Elétricos de Potência vêm passando por um grande desenvolvimento nas últimas décadas, procurando melhorar cada vez mais o processo de produção, transmissão e entrega de energia elétrica.

Esta evolução acarretou um aumento crescente da complexidade dos Sistemas Elétricos, exigindo o desenvolvimento de sistemas de supervisão, controle e automação cada vez mais sofisticados com o objetivo de garantir um funcionamento seguro e confiável.

Entretanto, ainda assim, os Sistemas Elétricos estão sujeitos a distúrbios e normalmente quando eles ocorrem, os operadores destes sistemas tentam solucioná-los desenvolvendo três atividades básicas : ***detecção dos problemas existentes, diagnóstico e determinação das ações corretivas.***

Na detecção dos problemas existentes, os operadores contam com o auxílio do sistema de supervisão que lhes informa todos os alarmes gerados (normalmente, a quantidade de alarmes gerados é elevada dificultando ainda mais o trabalho). Porém, em relação ao diagnóstico (estabelecimento dos motivos que geraram os alarmes) e a determinação das ações corretivas, os operadores em geral dependem quase que exclusivamente das suas experiências e conhecimentos.

Assim, vários fatores podem acabar levando os operadores a fazerem um diagnóstico errado, como a quantidade excessiva de alarmes, a fadiga, o estresse, a falta de experiência, a dificuldade natural em lidar com situações não usuais, entre outros.

Este diagnóstico falho acaba retardando a tomada das ações corretivas apropriadas, prejudicando a segurança e eficiência de operação do sistema.

Em conseqüência, tem-se notado um interesse cada vez maior no desenvolvimento de sistemas que possam ajudar o operador nas etapas de diagnóstico e determinação das ações corretivas.

I.2 - Objetivos :

Este trabalho pretende analisar a utilização de técnicas de Inteligência Artificial na diagnose de funcionamento dos Sistemas de Potência e, com isso, contribuir para melhorar o desempenho e reduzir o estresse do operador, dando-lhe mais segurança e agilidade no processo de tomada de decisão sobre a causa do distúrbio.

Além disso, ele também pode ser de grande valia na análise pós-despacho e no treinamento de operadores através da simulação de problemas nos Sistemas de Potência.

O objetivo principal deste trabalho é ***estudar comparativamente a aplicação de duas técnicas de Inteligência Artificial na diagnose de um Sistema de Potência***. Cada uma das técnicas utiliza uma concepção diferente quanto ao tratamento para a representação e aquisição do conhecimento, baseando-se no modelo simbolista e conexionista respectivamente.

Para a implementação destas duas técnicas de Inteligência Artificial, utilizaremos um caso exemplo de um sistema elétrico simplificado de uma usina.

I.2.1 - Diagnose utilizando o Modelo Simbolista :

O modelo simbolista apresenta duas formas de representação do conhecimento:

- ***Representação Declarativa*** → O conhecimento está representado de forma explícita através das expressões de uma linguagem bem definida com semântica e sintaxe claramente especificadas.
- ***Representação Procedural*** → O conhecimento está representado de forma implícita através de procedimentos ou algoritmos. Assim, para realizar uma determinada tarefa, alguém deve compreender como essa tarefa pode ser efetuada e desenvolver um procedimento para levá-la a efeito passo à passo.

Neste trabalho, o modelo simbolista será representado pela união das duas formas de representação onde :

- A ***Representação Declarativa*** utilizará a ***Lógica*** [Enderton 72] para descrever o nosso conhecimento sobre o Sistema Elétrico. Tal escolha deve-se ao fato da Lógica fornecer a base teórica matemática para a Inteligência Artificial [Geneserth *et. al.* 87 pp.vii,viii] e por ser um método de representação natural e intuitivo.
- A ***Representação Procedural*** usará a linguagem ***Prolog*** para processar este conhecimento já que o Prolog representa uma implementação da Lógica como linguagem computacional. Esta interpretação procedimental pode ser encontrada em detalhes em [Kowalski 74], [Casanova *et. al.* 86].

Entretanto, devemos ressaltar que existem várias maneiras de se realizar um diagnóstico nos modelos simbolistas, onde podemos destacar :

1. A diagnose baseada apenas na descrição sobre o funcionamento normal do sistema. Neste tipo de diagnose, o conflito entre as observações do sistema e a descrição sobre o seu comportamento normal caracterizam um problema, onde o objetivo do diagnóstico é

determinar as suas causas [Genesereth 84], [Reiter 87], e [de Kleer *et. al.* 87].

2. A diagnose baseada apenas no funcionamento anormal do sistema, ou seja, só temos informações sobre as falhas (doenças) e seus sintomas.
   Neste tipo de diagnose, o objetivo é obter explicações sobre as observações anormais do sistema [Cox *et. al.* 87] , [Poole *et. al.* 87] e [Poole 89].
   Uma análise deste tipo de diagnose foi realizada utilizando o *Sistema Theorist* ([Poole *et. al.* 87], [Poole 88b]) de *Raciocínio Não-monotônico* [Lukaszewicz 90].
3. A diagnose baseada principalmente nas experiências passadas por especialistas do sistema e em informações heurísticas, onde a descrição do sistema é fracamente representada. Um sistema bastante conhecido baseado neste tipo de diagnose é o MYCIN [Buchanan *et. al.* 84].

Uma análise comparativa destes tipos de diagnose pode ser vista em [Poole 88a], [Poole 89] e [Konolige 92].

Em nosso trabalho, a diagnose de falhas simples ou múltiplas do Sistema de Potência se baseará principalmente em um conhecimento sobre o comportamento anormal deste sistema.

Porém, somos também capazes de diagnosticar falhas mesmo quando o conjunto de alarmes que a descrevem estiver incompleto, graças a informações fornecidas por especialistas do Sistema Elétrico.

Deste modo, podemos concluir que utilizamos características desejáveis dos dois últimos tipos de diagnose comumente usadas nos modelos simbolistas.

Para atingir estes objetivos, desenvolvemos um interpretador (Apêndice A, [Poole *et. al.* 87]) e um compilador (Apêndice B, [Poole 91]) em Prolog para o Sistema Theorist que serão utilizados na implementação do caso exemplo.

I.2.2 - Diagnose utilizando o Modelo Conexionista :

Os modelos conexionistas se inspiram no sistema neuronal biológico. Nestes modelos, a informação inteligente deve ser processada como no cérebro, ou seja, paralelamente por um conjunto de elementos computacionais simples, denominados neurônios. A inteligência conexionista estaria armazenada nas conexões entre os neurônios.

Assim, tarefas como associação, generalização, percepção entre outras, que são muito difíceis de serem algoritmizadas, poderiam ser realizadas de modo tão natural quanto o efetuado pelo cérebro.

As ***Redes Neurais Artificiais*** ([Rumelhart *et. al.* 86], [Hecht-Nielsen 90]) representarão o modelo conexionista neste nosso trabalho e possuem um escopo muito grande de aplicações inclusive na área de maior interesse para nossos estudos que é a área de processamento de alarmes e diagnose.

Para os modelos conexionistas, a diagnose pode ser vista como um problema de classificação de padrões, pois ao identificarmos corretamente um conjunto particular de alarmes, estamos determinando qual(is) a(s) falha(s) que originou(aram) estes alarmes.

Deste modo, a diagnose pode ser descrita como uma tarefa hetero-associativa onde, caracterizamos um padrão de saída (falha) em função de um padrão de entrada (alarmes).

Sabemos que as tarefas hetero-associativas podem ser realizadas com facilidade pelas redes neurais, inclusive, devido à sua capacidade de generalização, ela consegue classificar padrões de entrada incompletos.

Neste trabalho, a escola conexionista será representada pelo modelo de rede neural denominado ***"Back-propagation"*** [Rumelhart *et. al.* 86], já que as tarefas hetero-associativas são bem realizadas pelas redes neurais sem realimentação (denominadas redes "feedforward") e o modelo "Back-

propagation" é a rede sem realimentação mais difundida e utilizada para este tipo de tarefa.

Utilizaremos o software comercial de simulação de Redes Neurais, NeuralWorks [NeuralWare 91] para implementação do caso exemplo utilizando o modelo "Back-propagation".

I.3 - Descrição dos Capítulos :

No capítulo II, faremos uma análise das razões que levam os operadores a enfrentarem tantas dificuldades na ocorrência de um distúrbio e apresentaremos as características gerais que um sistema deve possuir para auxiliar e agilizar o operador nesta tarefa.

Neste mesmo capítulo, será apresentado o caso exemplo de um Sistema Elétrico simplificado de uma usina que será utilizado na implementação das duas técnicas de Inteligência Artificial e por fim, mostraremos o estágio atual das aplicações de Inteligência Artificial em Sistemas de Potência.

No capítulo III, será mostrada uma abordagem para realizar um diagnóstico baseado em ***Lógica*** denominada ***Diagnose Abdutiva***. Faremos ainda uma descrição do ***Theorist*** (modelo desenvolvido para representação da diagnose abdutiva), apresentando a sua linguagem de programação e o seu funcionamento e, finalizando, o utilizaremos na implementação do caso exemplo.

No capítulo IV, mostraremos os conceitos básicos das ***Redes Neurais Artificiais***, dando destaque ao seu aprendizado. Apresentaremos em detalhe o modelo mais conhecido e utilizado que é o ***"Back-propagation"*** e, por fim, usaremos este modelo como ferramenta para solução do caso exemplo descrito no capítulo II.

No capítulo V, realizaremos uma comparação entre as duas técnicas de Inteligência Artificial (Lógica Não-monotônica e Redes Neurais Artificiais),

mostraremos as conclusões referentes ao trabalho como um todo e finalmente, faremos algumas sugestões de melhorias e pesquisas futuras.

As informações complementares consideradas relevantes são apresentadas nos apêndices.

Nos apêndices A e B, são apresentadas as listagens comentadas das versões interpretada [Poole *et. al.* 87] e compilada [Poole 91] do Theorist.

No apêndice C, descreveremos mais detalhadamente um dos métodos de diagnose baseada em Lógica bastante utilizado na literatura que é a Diagnose Baseada na Consistência.

No apêndice D, mostraremos a descrição completa das falhas do caso exemplo utilizando o Theorist.

No apêndice E, apresentaremos a listagem completa da implementação do caso exemplo utilizando Lógica Não-monotônica, representada pela Diagnose Abdutiva e o Theorist.

No apêndice F, mostraremos a listagem do interpretador de resultados gerados pelos padrões de teste de diagnóstico da rede neural.

Capítulo II

A Operação de Sistemas de Potência

II.1 - Estrutura Hierárquica :

O aumento da demanda de energia obrigou os Sistemas Elétricos de Potência a evoluirem continuamente ao longo das últimas décadas.

Com isso, a operação de Sistemas de Potência torna-se cada vez mais complexa e pode ser vista como uma série de ações de controle necessárias para atender ao suprimento de energia elétrica aos consumidores com qualidade, confiabilidade e economia.

Atualmente, esta operação é realizada por uma estrutura hierárquica disposta em vários níveis, onde cada um desses níveis possui atribuições, características e recursos específicos. Deste modo, cada empresa de energia elétrica de grande porte geralmente opera sua rede através dos seguintes níveis :

- ***Centro de Operação do Sistema (COS) ou Despacho Central*** → tem por objetivo o controle e a supervisão de todo o sistema elétrico da empresa de modo a assegurar a sua correta operação bem como os de seus despachos regionais.
- ***Centro de Operação Regional (COR) ou Despacho Regional*** → é responsável por coordenar a operação de uma determinada área (parte das usinas e subestações) do sistema elétrico da empresa.
- ***Usinas e Subestações*** → dispõe localmente de recursos para controle da própria usina ou subestação porém, alguns destes recursos estão subordinados aos níveis hierárquicos superiores.

As ***Usinas*** são responsáveis pela geração de energia elétrica, que é transmitida aos centros consumidores através de linhas de transmissão. Já as ***Subestações*** são responsáveis pelas conexões entre usinas, linhas de transmissão e centros consumidores permitindo realizar as manobras necessárias de acordo com as necessidades de operação.

Entretanto, a maioria das empresas de energia fazem parte de um sistema interligado (ou seja, diversas companhias conectadas entre si, formando uma grande rede de geração e transmissão) e com isso, devemos acrescentar mais um nível a hierarquia que é o centro de operação do "pool" destas empresas.

No Brasil, as principais concessionárias de energia da região sudeste-sul (como CEMIG, CESP, ELETROSUL, FURNAS e LIGHT) constituem o sistema interligado sudeste-sul, que é operado pelo COS de cada empresa sob a ***coordenação nacional do CNOS - Centro Nacional de Operação do Sistema da Eletrobrás*** (figura II.1).

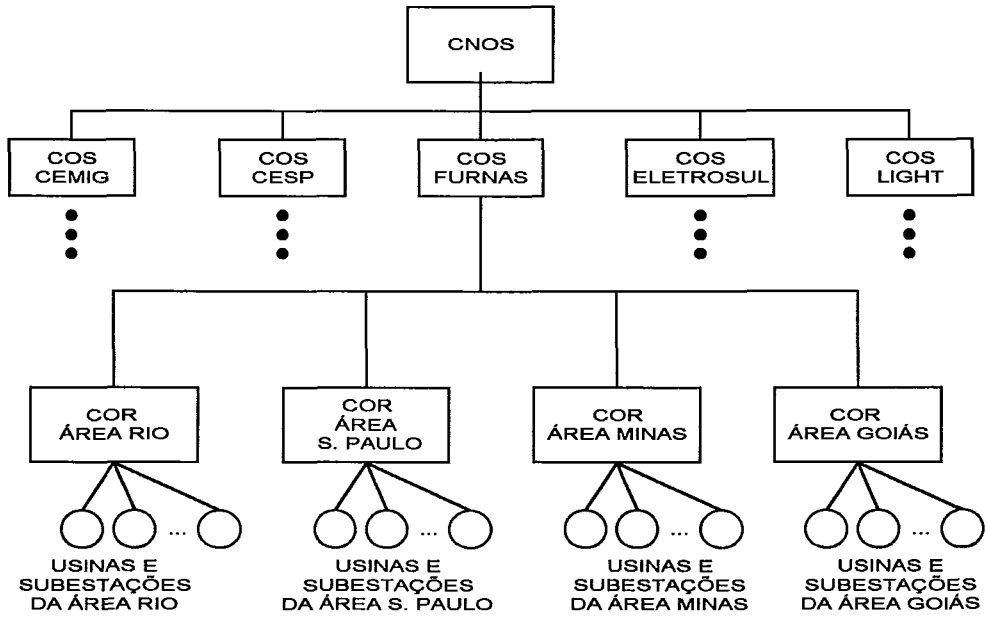


Figura II.1 : Estrutura Hierárquica da região sudeste-sul do Brasil

II. 2 - Sistemas de Supervisão e Controle :

Os ***Sistemas de Supervisão e Controle*** utilizados nos vários níveis hierárquicos de operação de um Sistema de Potência visam obter, de uma forma eficiente e segura, a monitoração, o comando e a otimização de parâmetros vitais à operação do sistema elétrico.

A explosão tecnológica na área de computadores e sistemas de comunicação provocou um grande desenvolvimento, permitindo a obtenção de Sistemas de Supervisão e Controle com alta confiabilidade, flexibilidade e melhores taxas de resposta.

Com isso, houve uma melhora significativa na eficiência e confiabilidade dos dados, provenientes das Unidades Terminais Remotas (UTR's) localizadas nas subestações e usinas ou transmitidos de outros centros de controle e recebidos pelos COS e COR de cada empresa.

Diante de tantas transformações, o papel do operador mudou drasticamente. Suas funções que até então eram na maior parte do tempo de registro, tornaram-se mais analíticas, deixando para o computador as tarefas de registrar e apresentar os resultados processados.

O operador passou, então, a analisar estes resultados, tomar decisões e executá-las com o auxílio dos computadores. Deste modo, houve um aumento da importância da monitoração e controle dos Sistemas Elétricos visando um funcionamento mais seguro e confiável.

Segundo [Dy Liacco 78], o objetivo da ***operação de um Sistema de Potência*** deve ser o ***controle da sua segurança***, onde segurança é a capacidade do sistema ficar em operação, mantendo seus equipamentos dentro de suas capacitações e atendendo a demanda de consumo de energia mesmo quando ocorre um distúrbio.

Porém, a ***operação de um Sistema Elétrico*** é bem complexa, sendo muito difícil a implementação de um controle único. Por este motivo, [Dy Liacco 67] propôs que a operação de um Sistema de Potência fosse ***decomposta em três estados*** ou modos de operação : ***normal, emergência e restaurativo.***

A figura II.2 mostra a relação entre estes três estados operativos, onde as transições pontilhadas correspondem a situações originadas por eventos e as transições cheias correspondem a ações corretivas do operador.

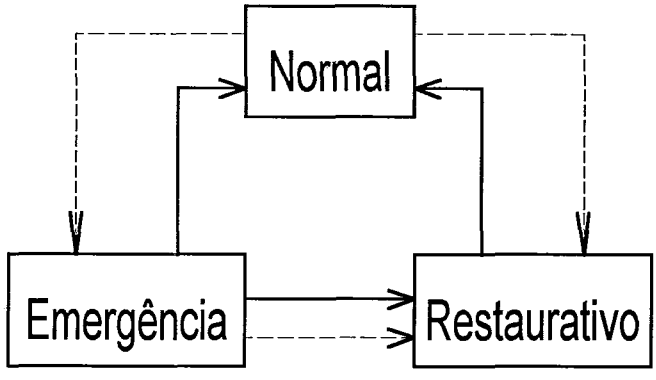


Figura II.2 : Estados de Operação de um Sistema de Potência

Deste modo, o problema da operação de um Sistema de Potência passa a ser subdividido em três subproblemas menores. Entretanto, o objetivo global de controle é manter o sistema sempre no estado normal.

Usualmente, os Sistemas de Potência operam a maior parte do tempo no estado normal, onde a intervenção do operador é pequena e os procedimentos de operação são bem conhecidos.

Quando o Sistema Elétrico enfrenta problemas mais sérios, ele tende a entrar em um dos outros estados :

- ***Estado de Emergência*** → ocorre quando a capacidade de algum equipamento está excedida ou quando alguma grandeza assumiu um valor fora dos limites aceitáveis.

- *Estado Restaurativo* → acontece quando as demandas de carga dos consumidores não estão sendo atendidas, ou seja, ocorreu uma interrupção do fornecimento de energia para parte dos consumidores.

Estas situações acontecem durante a ocorrência de um distúrbio, onde a quantidade de alarmes gerados é elevada fazendo com que o operador tenha dificuldades em reconhecer a causa do distúrbio e determinar as ações corretivas a serem executadas.

II.2.1 - Processamento de Alarmes :

Os alarmes têm por finalidade alertar o operador sobre mudanças de estado detectadas em tempo real, podendo reportar desde anormalidades no Sistema de Potência, como a ultrapassagem de limites de algum valor analógico, até situações comuns como a abertura de um disjuntor para sua manutenção.

Para cada alarme gerado, o operador normalmente deve executar os seguintes procedimentos [Wollenberg 86]:

1. Tomar conhecimento do alarme;
2. Determinar qual evento o causou;
3. Analisar as conseqüências deste evento;
4. Procurar analisar a seqüência de ocorrências que levaram a esta condição de alarme; e
5. Determinar, caso seja necessário, uma ação para normalizar a situação.

Porém, podemos verificar na realidade que tais procedimentos não são executados pelo operador para cada alarme distintamente, mas sim, fazendo uma correlação entre os alarmes existentes, os que estão ocorrendo, o estado atual do Sistema Elétrico, como também a sua própria experiência.

Mas durante a execução deste processo, o operador enfrenta vários obstáculos que dificultam a sua tarefa tais como :

- Quantidade excessiva de alarmes durante distúrbios no Sistema Elétrico;
- Alarmes muito específicos;
- Alarmes não atuados devido ao mau-funcionamento de algum equipamento;
- Alarmes múltiplos em consequência de um mesmo evento;
- Alarmes inconsistentes, ou seja, sua informação não tem valor em uma determinada situação.

Sabemos que o operador, como qualquer ser humano, possui limitações, só conseguindo absorver e processar tantas informações após um determinado período de tempo.

É nesta hora que o Sistema de Supervisão e Controle deve atuar no auxílio ao operador, realizando algum pré-processamento para, ao invés de mostrar puramente os dados, destacar o que eles significam, ajudando no processo de tomada de decisão.

O estabelecimento de prioridades para os alarmes é uma característica desejável neste pré-processamento, principalmente se esta prioridade for dinâmica (ou seja, alterável em função do estado em que se encontra o Sistema de Potência). Deste modo, só serão enfatizados para o operador os alarmes relevantes para a situação em que ele se encontra atualmente.

A capacidade do Sistema de Supervisão e Controle de realizar diagnóstico (ou seja, estabelecer os motivos que geraram os alarmes) é outro aperfeiçoamento interessante pois facilita extremamente o operador na tomada das ações corretivas apropriadas, tornando o sistema mais eficiente e seguro.

Como já foi visto na seção I.1, o processo de diagnóstico e decisão durante distúrbios é geralmente mal-estruturado e sua solução depende muito da

experiência e perícia do operador em agir acertadamente e, nem sempre, os operadores com tais capacitações estão presentes, atentos ou com a sua eficiência habitual.

A introdução de técnicas de Inteligência Artificial nos Sistemas de Supervisão e Controle representa um grande passo na solução deste e de vários outros problemas pois as áreas com maior potencial para sua aplicação são principalmente aquelas em que existam :

- Um grande número de combinações de parâmetros para sua execução;
- Tomada de decisão com informação incompleta e incerta;
- Mão-de-obra qualificada não disponível continuamente, havendo um grande risco de perdê-la, sendo então necessário armazenar o conhecimento especializado;
- Necessidade de instrução de pessoal com pouca experiência, uma vez que a Inteligência Artificial pode explicar qual o raciocínio empregado para alcançar uma determinada solução.

Notadamente, estas características estão presentes na Diagnose em Sistemas de Potência e este trabalho pretende estudar detalhadamente soluções para este problema utilizando técnicas de Inteligência Artificial.

Para tal, será apresentado a seguir, um caso exemplo de um Sistema Elétrico que será utilizado para demonstrar a viabilidade de aplicação da Inteligência Artificial no diagnóstico de falhas em um Sistema de Potência.

II.2.2 - Caso Exemplo :

Mostraremos como caso exemplo, um ***Sistema Elétrico simplificado de uma Usina*** (figura II.3). Este Sistema Elétrico é composto por ***duas unidades geradoras, dois transformadores e seus respectivos disjuntores, dois***

barramentos (barra principal e barra auxiliar) e duas linhas de transmissão (também com seus respectivos disjuntores).

Normalmente, os transformadores e as linhas de transmissão estão ligados ao barramento principal. Porém, quando um destes elementos está com seu disjuntor em manutenção, ele se liga ao barramento auxiliar e ativa o fechamento do disjuntor de by-pass para se ligar novamente ao barramento principal.

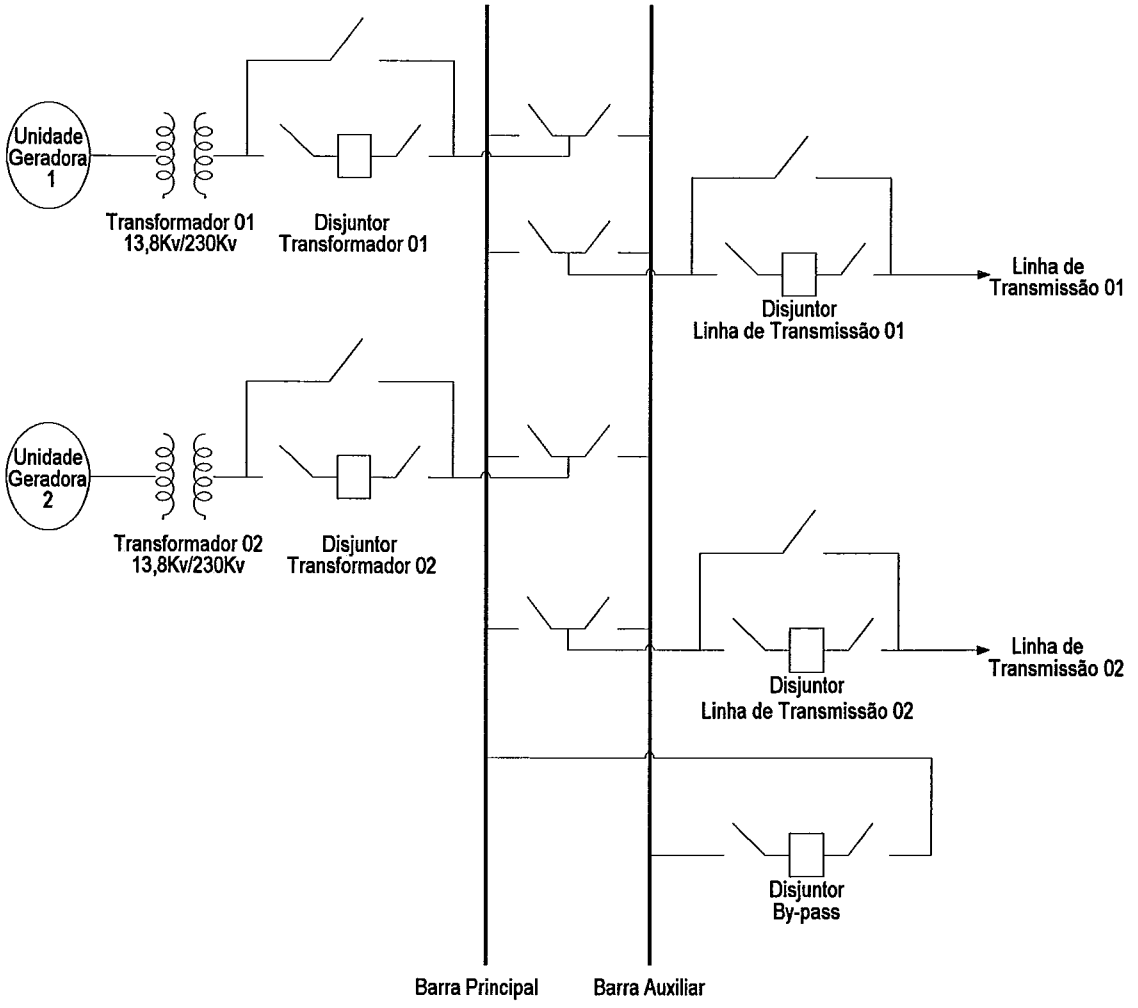


Figura II.3 : Configuração do Sistema Elétrico Simplificado de uma Usina

Cada **transformador** possui três pontos de monitoração:

- **Abertura do disjuntor** → indica se o disjuntor do transformador está aberto ou não,

- **Sobrecarga** → mostra que ocorreu sobrecarga no transformador,
- **Auxiliar** → informa se o transformador está operando com o disjuntor de by-pass, ou seja, seu próprio disjuntor está em manutenção.

Já cada **linha de transmissão** possui seis pontos de monitoração :

- **Abertura do disjuntor** → indica se o disjuntor da linha de transmissão está aberto ou não,
- **Fase** → mostra que ocorreu sobrecorrente em uma das fases **(f/f)** da linha de transmissão,
- **Terra** → informa que ocorreu sobrecorrente de terra **(f/t)** na linha de transmissão,
- **Temporizado** → significa que ocorreu uma falha distante à usina na linha de transmissão,
- **Instantâneo** → revela que ocorreu uma falha próxima à usina na linha de transmissão,
- **Auxiliar** → informa se a linha de transmissão está operando com o disjuntor de by-pass, ou seja, seu próprio disjuntor está em manutenção.

A seguir, mostraremos a lista de descrição de problemas (falhas) deste sistema (tabelas II.1 e II.2), onde cada falha está associada a um grupo de alarmes. Deste modo, a falha próxima na linha de transmissão 01 f/f é caracterizada pela atuação dos alarmes de abertura do disjuntor da linha de transmissão 01, fase da linha de transmissão 01 e instantâneo da linha de transmissão 01 e assim por diante, onde podemos notar que "X" significa que o alarme deverá estar atuado para aquela descrição de problema e " " significa que não.

Descrição do Problema / Alarmes	Falha próxima na Lt 01 F/F	Falha próxima na Lt 01 F/F (1)	Falha próxima na Lt 01 F/T	Falha próxima na Lt 01 F/T (1)	Falha distante na Lt 01 F/F	Falha distante na Lt 01 F/F (1)	Falha distante na Lt 01 F/T	Falha distante na Lt 01 F/T (1)	Falha próxima na Lt 02 F/F	Falha próxima na Lt 02 F/F (2)	Falha próxima na Lt 02 F/T	Falha próxima na Lt 02 F/T (2)	Falha distante na Lt 02 F/F	Falha distante na Lt 02 F/F (2)
1 - Abriu. Disj. Lt 01	X		X		X		X							
2 - Fase Lt 01	X				X									
3 - Terra Lt 01			X				X							
4 - Temporizado Lt 01					X		X							
5 - Instantâneo Lt 01	X		X											
6 - Auxiliar Lt 01		X		X		X		X						
7 - Abriu Disj. Lt 02									X		X		X	
8 - Fase Lt 02									X				X	
9 - Terra Lt 02											X			
10 - Temporizado Lt 02													X	
11 - Instantâneo Lt 02									X		X			
12 - Auxiliar Lt 02										X		X		X
13 - Abriu Disj. By-pass		X		X		X		X		X		X		X
14 - Fase By-pass		X				X				X				X
15 - Terra By-pass				X				X				X		
16 - Temporizado By-pass						X		X						X
17 - Instantâneo By-pass		X		X						X		X		
18 - Abriu Disj.Transf1														
19 - Auxiliar Transf1														
20 - Sobrecarga Transf1														
21 - Abriu Disj Transf2														
22 - Auxiliar Transf2														
23 - Sobrecarga Transf2														

(1) Linha de Transmissão 01 operando com By-pass (2) Linha de Transmissão 02 operando com By-pass (3) Transformador 01 operando com By-pass (4) Transformador 02 operando com By-pass

Tabela II.1 : Descrição das Falhas e seus alarmes correspondentes

Descrição do Problema / Alarmes	Falha distante na Lt 02 F/T	Falha distante na Lt 02 F/T (2)	Falha na Barra Principal	Falha na Barra Principal (1)	Falha na Barra Principal (2)	Falha na Barra Principal (3)	Falha na Barra Principal (4)	Falha na Barra Auxiliar (1)	Falha na Barra Auxiliar (2)	Falha na Barra Auxiliar (3)	Falha na Barra Auxiliar (4)	Falha no Transf. 1	Falha no Transf. 2	Falha no Transf. 1 (3)	Falha no Transf. 2 (4)
1 - Abriu. Disj. Lt 01			X		X	X	X								
2 - Fase Lt 01															
3 - Terra Lt 01															
4 - Temporizado Lt 01															
5 - Instantâneo Lt 01															
6 - Auxiliar Lt 01				X				X							
7 - Abriu Disj. Lt 02	X		X	X		X	X								
8 - Fase Lt 02															
9 - Terra Lt 02	X														
10 - Temporizado Lt 02	X														
11 - Instantâneo Lt 02															
12 - Auxiliar Lt 02		X			X				X						
13 - Abriu Disj. By-pass		X		X	X	X	X	X	X	X	X			X	X
14 - Fase By-pass															
15 - Terra By-pass		X													
16 - Temporizado By-pass		X													
17 - Instantâneo By-pass															
18 - Abriu Disj.Transf1			X	X	X		X					X			
19 - Aux.Transf1						X				X				X	
20 - Sobrecarga Transf1												X		X	
21 - Abriu Disj Transf2			X	X	X	X							X		
22 - Aux .Transf2.							X				X				X
23 - Sobrecarga Transf2													X		X

(1) Linha de Transmissão 01 operando com By-pass (2) Linha de Transmissão 02 operando com By-pass (3) Transformador 01 operando com By-pass (4) Transformador 02 operando com By-pass

Tabela II.2 : Descrição das Falhas e seus alarmes correspondentes

Podemos observar pelas tabelas II.1 e II.2, que temos 29 tipos diferentes de distúrbios, onde cada um deles está associado, na maioria das vezes, a pelo menos três alarmes diferentes.

Assim, vemos que neste exemplo bastante simples já existe uma boa quantidade de informações, mostrando como a situação de um operador é bastante crítica e delicada durante um distúrbio em um Sistema de Potência real, onde a quantidade de informações é um fator muito mais complexo.

Durante a implementação, nosso objetivo será utilizar este caso exemplo para procurar uma explicação para um conjunto de alarmes atuados.

Deste modo, quando o sistema apresentar os seguintes alarmes atuados : abriu o disjuntor da linha de transmissão 01, terra da linha de transmissão 01 e instantâneo da linha de transmissão 01, ele deve ser capaz de diagnosticar que ocorreu uma falha próxima na linha de transmissão 01 f/t (Tabela II.1).

Devemos destacar que o sistema deve ser capaz de diagnosticar não somente falhas simples mas também falhas múltiplas e ser apto a diagnosticar falhas mesmo quando o conjunto de alarmes que a caracterizam estiver incompleto. Assim, se somente os alarmes terra da linha de transmissão 01 e temporizado da linha de transmissão 01 estiverem atuados, o sistema deve ter a habilidade de diagnosticar que a falha distante na linha de transmissão 01 f/t ocorreu mesmo sem a presença do alarme de abertura do disjuntor da linha de transmissão 01. Devemos notar que esta característica aumenta consideravelmente a complexidade do problema.

A seguir, mostraremos como estão os estudos e pesquisas para aplicação de Inteligência Artificial na solução deste e de outros problemas dos Sistemas de Potência.

II.3 - Aplicações de Inteligência Artificial em Sistemas de Potência:

Como já foi citado anteriormente, a evolução do Sistema Elétrico exige um número crescente e mais complexo de funções para os Sistemas de Supervisão e Controle, tornado-o mais eficiente e seguro.

Com isto, a operação em Sistemas de Potência passou a requerer cada vez mais uma mão-de-obra com larga experiência e altamente qualificada. Entretanto, verifica-se exatamente o contrário, ou seja, um número cada vez maior de operadores com pouca experiência, bem como um grupo crescente de operadores com grande experiência, mas à beira da aposentadoria.

Deste modo, a partir do meio da década de 80, começou-se a se preocupar em criar alguma forma de preservar o conhecimento dos operadores com larga experiência, ajudando os operadores mais novos a defrontar com situações de distúrbio e tomar as ações corretivas necessárias ([Sakaguchi *et. al.* 87], [Kawada *et. al.* 89]).

Em paralelo a isto, ocorria um grande avanço dos sistemas computacionais e um crescimento do interesse em novas metodologias, tendo como conseqüência o início das primeiras pesquisas utilizando técnicas de Inteligência Artificial para solução deste e de outros problemas que ocorrem em Sistemas de Potência.

Diversos trabalhos têm sido desenvolvidos propondo ou descrevendo aplicações de Inteligência Artificial em Sistemas de Potência. A seguir, mostraremos o estágio em que se encontra atualmente estas aplicações, diferenciando-as entre os dois modelo existentes : o simbolista e o conexionista.

II.3.1 - Aplicações utilizando o Modelo Simbolista :

As primeiras aplicações de Inteligência Artificial em Sistemas de Potência foram propostas utilizando o modelo simbolista e as pesquisas ([Tamura 89], [Zhang *et. al.* 89], [CIGRE 92]) mostram que a maioria dos trabalhos está sendo realizado nas áreas de controle (como a recomposição e o controle de emergência), planejamento operacional e monitoração (diagnose de faltas, processamento de alarmes, entre outros) como pode ser visto na figura II.4.

O método de representação do conhecimento mais utilizado nestes trabalhos é a regra de produção, porém existem várias outras técnicas que merecem destaque tais como a lógica de 1ª ordem, frame, redes semânticas, roteiros ("script") e lógica fuzzy. Devemos destacar entretanto, que as regras de produção podem ser vistas como um subconjunto da lógica de 1ª ordem [Genesereth *et. al.* 87] e as redes semânticas podem ser mapeadas através da lógica não-monotônica [Etherington 87a] [Etherington 88] [Lukaszewicz 90].

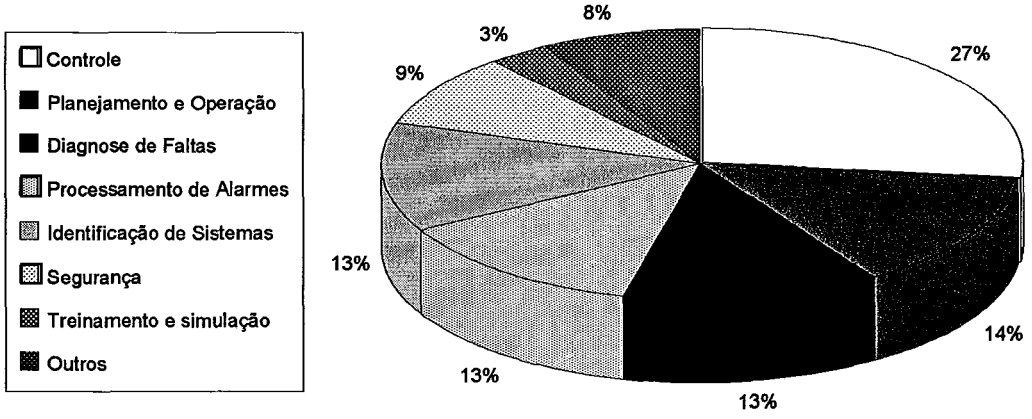


Figura II.4 : Aplicações de Modelos Simbolistas no setor elétrico

Porém, detectou-se que existem problemas a serem enfrentados durante a implementação destas aplicações [CIGRE 92], tais como a dificuldade para

adquirir o conhecimento dos especialistas (normalmente, este processo é longo e demorado) e a necessidade de muitos testes "off-line" e observações "on-line" para sua validação. Mas, quando estes obstáculos são bem superados, nota-se uma aceitação por parte dos operadores, facilitando e diminuindo o tempo para qualquer atualização e alteração que se queira fazer.

Esta mesma pesquisa verificou que os principais motivos que levaram a utilização de Inteligência Artificial no setor elétrico foram a redução das tarefas rotineiras do operador, a prevenção contra erros humanos, bem como o desejo de validar novas técnicas ou metodologias.

Como já foi mostrado anteriormente na seção II.2, a diagnose de faltas e o processamento de alarmes (áreas que fazem parte do propósito de aplicação desta tese) se encaixam perfeitamente nas características citadas acima e por isso, elas representam juntas 26% dos trabalhos realizados em Inteligência Artificial utilizando o modelo simbolista. Destes podemos destacar : [Fukui *et. al.* 86], [Cardozo *et. al.* 87], [Kirschen *et. al.* 89], [Hein *et. al.* 88], [Lambert-Torres *et. al.* 91] e [Wollenberg *et. al.* 91].

Devemos ressaltar que até hoje, nós não temos notícias da publicação de algum estudo ou aplicação da Lógica Não-monotônica (técnica que será usada na implementação do caso exemplo) no setor elétrico, apesar dela já ser utilizada principalmente para diagnose, especialmente na área médica e de circuitos elétricos, e treinamento, como nos mostram [Reggia *et. al.* 83], [Jones *et. al.* 85], [de Kleer *et. al.* 87] e [Struss *et. al.* 89] .

II.3.2 - Aplicações utilizando o Modelo Conexionista :

Conforme será apresentado na introdução da seção IV.2, as propriedades das redes neurais artificiais fazem com que estas possuam um conjunto muito grande de aplicações dentre as quais podemos citar o setor elétrico.

A partir de 1988, o setor elétrico passou a utilizar as redes neurais para solucionar os mais diversos problemas, tendo sido publicado mais de 350 artigos somente sobre aplicações nesta área.

Uma pesquisa (figura II.5) feita recentemente pelo CIGRE [CIGRE 93], envolvendo 157 projetos diferentes, mostrou que a maioria dos trabalhos (cerca de 40%) estão sendo realizados para previsão de carga e segurança do sistema porém, existem várias áreas consideradas bastante promissoras, tais como : controle, diagnose de faltas e processamento de alarmes.

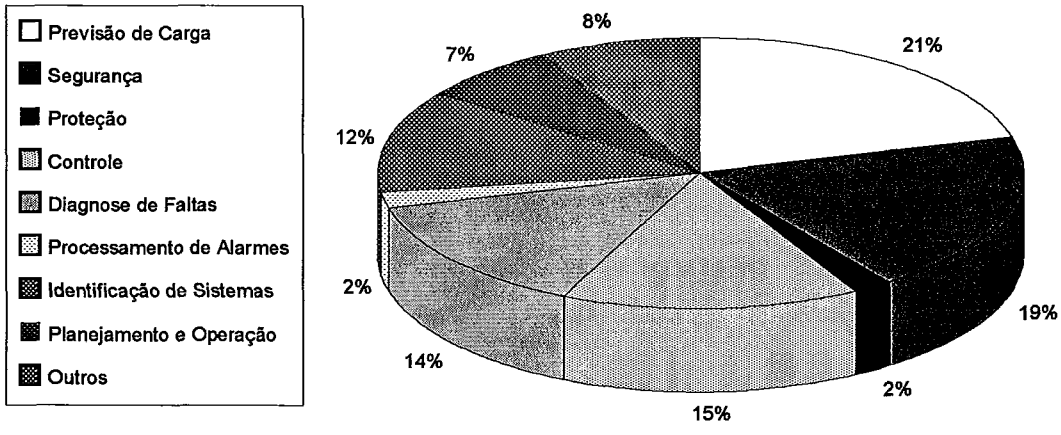


Figura II.5 : Aplicações de redes neurais no setor elétrico

Nesta mesma pesquisa, foram identificados quais os modelos de redes neurais utilizados nestes projetos (figura II.6), sendo que o grande destaque foi o modelo "Back-propagation", usado em 56% dos projetos.

Dentre as áreas consideradas promissoras, podemos destacar a diagnose de faltas e o processamento de alarmes, que fazem parte do escopo de aplicação desta tese.

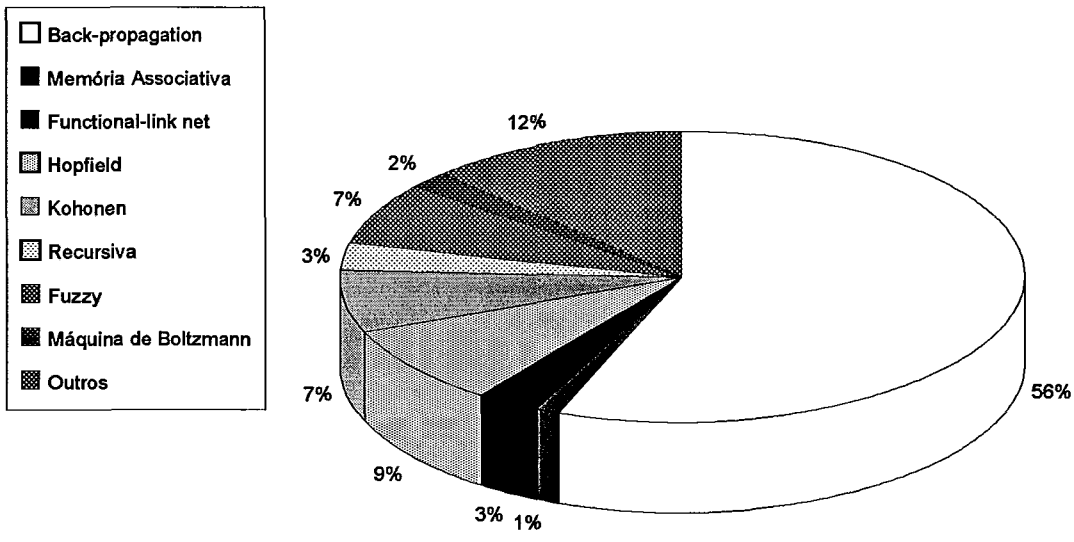


Figura II.6 : Modelos de Redes Neurais utilizados nas aplicações do setor elétrico

O grande objetivo destas duas áreas é fornecer ao operador do Sistema Elétrico, informações precisas do estado atual do sistema, auxiliando-o a identificar e corrigir qualquer problema existente.

Esta tarefa pode ser vista simplesmente como um problema de reconhecimento e interpretação de padrões pois, ao reconhecer corretamente um padrão particular de alarmes, consegue-se caracterizar qual o problema que originou estes alarmes.

A tarefa de reconhecer e interpretar padrões é realizada facilmente com sucesso pelas Redes Neurais, inclusive devido à sua capacidade de distribuição de memória, ela consegue classificar padrões de entrada incompletos ou com ruído. É claro que a precisão da classificação decai, mas ainda assim, é feita com sucesso.

Outra vantagem é a sua alta velocidade de processamento, requisito imprescindível para um rápido reconhecimento dos problemas existentes no Sistema Elétrico.

Devido a estas qualificações, vários trabalhos já foram desenvolvidos nestas duas áreas com sucesso, dentre os quais podemos destacar: [Chan 89], [Tanaka *et. al.* 89], [Kim *et. al.* 91], [Swarup *et. al.* 91], e [Rodrigues *et. al.* 92]. Destes, apenas [Chan 89] não utilizou o modelo "Back-propagation", já que a diagnose pode ser vista como uma tarefa hetero-associativa (seção I.2.2) e este tipo de tarefa é bem realizada pela redes neurais "feedforward", dentre as quais, o modelo mais difundido é o "Back-Propagation".

Iremos agora descrever mais detalhadamente as duas técnicas de Inteligência Artificial (Lógica Não-monotônica e Redes Neurais) que serão utilizadas na Diagnose em Sistemas de Potência.

Capítulo III

Lógica Aplicada à Diagnose

III.1 - Introdução :

Uma ***Diagnose*** pode ser definida como a investigação das causas do desvio de funcionamento correto em um determinado sistema. Para isto, baseia-se no modelo de comportamento deste sistema, nos possíveis mau-funcionamentos que podem ocorrer e nas manisfestações externas (sintomas) obtidas durante o seu funcionamento.

Existem, no entanto, várias maneiras de se estruturar o conhecimento através da combinação destes três principais elementos : ***o modelo, os mau-funcionamentos (anormalidades) e as observações.*** Consequentemente, pode-se realizar um diagnóstico partindo-se de abordagens bastante distintas, que relacionam estes três parâmetros em planos diferentes da relação causa *X* efeito. As abordagens mais utilizadas na literatura são:

- ***Diagnose baseada na Consistência*** → tem por objetivo descobrir um conjunto de hipóteses de anormalidade que possa explicar o conflito existente entre as observações e o comportamento normal do sistema [Reiter 87].

  Seu conhecimento é baseado na descrição do funcionamento correto do sistema [Geneserteth 84], [Reiter 87] e [de Kleer *et. al.* 87].

- ***Diagnose Abdutiva*** → tem por objetivo descobrir um conjunto de hipóteses que nos levam a explicar as observações [Poole *et. al.* 87] .
  Seu conhecimento é baseado principalmente nas informações sobre as falhas (doenças) do sistema e seus sintomas [Cox *et. al* 87], [Poole *et. al.* 87] e [Poole 89].
  Um sistema que implementa este tipo de diagnose é o Theorist [Poole *et. al.* 87] [Poole 88b].
- ***Diagnose baseada em regras*** → seu objetivo é determinar quais as anormalidades que podem ser previstas a partir dos sintomas observados ([Buchanan *et. al.* 84] e [Pearl 87]).
  Para tal, temos um conjunto de regras no formato *sintomas-causas,* onde seu conhecimento é baseado principalmente nas experiências passadas por especialistas do sistema e em informações heurísticas [Poole *et. al.* 87 pp. 336-338]. Um sistema bastante conhecido que utiliza este tipo de dignose é o MYCIN [Buchanan *et. al.* 84].

Assim, para se modelar um sistema objetivando uma diagnose do seu comportamento, é necessário um conhecimento sobre o seu domínio e todas as hipóteses que definem um comportamento normal e anormal dos componentes [Poole 88a pp. 1283]. Os elementos que precisam ser conhecidos são melhor definidos por :

- ***Modelo do Domínio*** → descreve a estrutura do sistema, como os componentes funcionam normalmente e como se comportam nos diversos tipos de falhas.
- ***Observações*** → apresenta o conjunto de observações sobre o funcionamento, necessário para diagnosticar o sistema.
- ***Hipóteses de Normalidade*** → são as suposições que descrevem que alguns componentes estão funcionando corretamente.

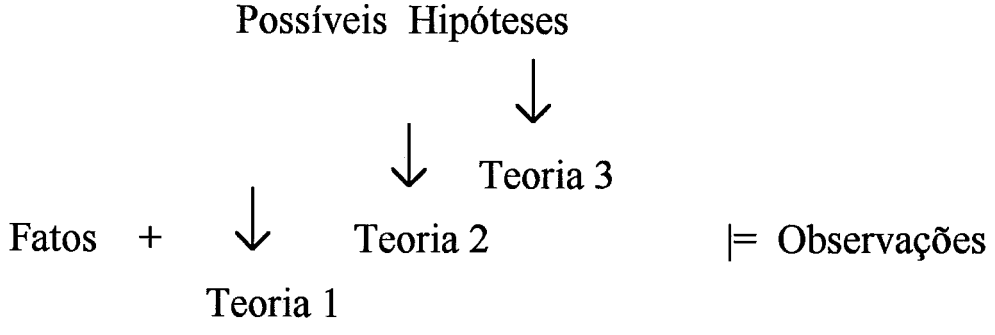
- ***Hipóteses de Anormalidade*** → são as suposições que descrevem que alguns componentes não estão funcionando corretamente.

A seguir, descreveremos mais detalhadamente um sistema para representação e inferência do conhecimento utilizando Lógica Não-monotônica [Lukaszewicz 90], denominado Theorist, mostrando como ele é aplicado em diagnose e raciocínio não-monotônico.

III.2 - Sistema Theorist :

O sistema desenvolvido por [Poole *et. al.* 87], [Poole 88a] denominado ***Theorist***, é ***uma teoria e implementação do raciocínio não-monotônico e abdutivo.***

A sua estratégia de raciocínio (figura III.1) consiste em obter justificativa para as observações a partir de um conjunto consistente de fatos e instâncias de possíveis hipóteses.



onde uma Teoria é um subconjunto das hipóteses possíveis, que são consistentes com os fatos e juntamente com eles, implicam nas observações.

Figura III.1 : Estratégia de Raciocínio do Theorist

Esta idéia de Poole [Poole *et. al.* 87] de utilizar lógica de 1ª ordem com fatos e hipóteses para se obter explicações, foi inspirada na concepção de teorias científicas [Popper 58] já que, ao conceber uma teoria, um cientista tenta a grosso modo criar um conjunto de hipóteses que aliadas ao conhecimento dado como certo (ou seja, fatos previamente comprovados) possa explicar os resultados observados.

Em [Poole *et. al.* 87], é mostrado que o ***Theorist*** é apropriado para modelar:

- o ***raciocínio default*** (neste caso, as possíveis hipóteses seriam os defaults),
- o ***aprendizado***, onde as hipóteses seriam generalizações das observações ou leis derivadas das mesmas,
- a ***diagnose***, onde as possíveis hipóteses seriam as doenças ou suposições de normalidade ou de mau-funcionamento.

O ***Sistema Theorist*** pode ser representado por uma ***tripla (F,H,C)*** onde:

- $F \rightarrow$ são os fatos do sistema, sendo sempre verdadeiros dentro do mundo modelado. São representados por um conjunto de fórmulas de 1ª ordem fechadas.
- $H \rightarrow$ são as possíveis hipóteses (ou "defaults"), que estão sempre preparadas para serem aceitas como justificativa para alguma observação. São representadas por um conjunto de fórmulas.
- $C \rightarrow$ é o conjunto de fórmulas fechadas denominadas "constraints" (ou restrições). Devemos ressaltar que os "constraints" são usados somente para rejeitar cenários (definição III.1 que será mostrada a seguir) e não podem ser usados para explicar qualquer coisa.

  Eles são mecanismos utilizados para dar prioridades explicitamente e estão relacionados com a parte não-normal da justificativa dos defaults da *Lógica Default de Reiter* [Reiter 80], como veremos posteriormente.

Eles também são usados para bloquear o uso dos defaults nas sua forma contrapositiva.

Estes três conjuntos ***(Fatos, Possíveis Hipóteses e "Constraints")*** são responsáveis pela definição semântica do ***Theorist*** através de três conceitos fundamentais: ***cenário, explanação e extensão*** [Poole 88a], [Poole 88b], [Poole 89]. Devemos mencionar, que para as definições abaixo, são necessários conhecimentos básicos sobre lógica de 1ª ordem ([Casanova *et. al.* 86], [Lukaszewicz 90]) .

DEFINIÇÃO III.1 - Um ***cenário de (F,H,C)*** é um conjunto $D \cup F$ onde D é o conjunto de instâncias básicas de elementos H tal que $D \cup F \cup C$ é consistente.

DEFINIÇÃO III.2 - Uma ***fórmula fechada O*** é explicável a partir de (F,H,C) se existir um conjunto D de instâncias básicas de elementos de H tal que:

$$D \cup F \models O \text{ e}$$

$$D \cup F \cup C \text{ seja consistente.}$$

Portanto, podemos dizer que $F \cup D$ é uma ***explicação (explanação) da fórmula O.***

DEFINIÇÃO III.3 - Uma ***extensão de (F,H,C)*** é um conjunto de consequências lógicas de um cenário maximal de (F,H,C) com respeito a inclusão.

A partir destas definições e de propriedades da lógica de 1ª ordem (dentre as quais, o teorema da compacidade), podemos destacar duas propriedades bastante interessantes que o Theorist apresenta [Poole 88b pp. 30]:

- Se existe alguma explanação para ***O***, então ***O*** é explicável por um cenário finito.

- Se D1 e D2 são subconjuntos de H onde D1 $\subseteq$ D2 e $F \cup D1 \models \boldsymbol{O}$ então, $F \cup D2 \models \boldsymbol{O}$, ou seja, o acréscimo de novas hipóteses não altera uma explicação. Esta propriedade é denominada ***semi-monotonicidade*** (monotonicidade em relação aos defaults e não com relação aos fatos e "constraints").

Com base nestas duas propriedades, Poole apresenta o seguinte teorema :

TEOREMA III.1 [Poole 88b pp. 30,31]- $\boldsymbol{O}$ é explicável sse $\boldsymbol{O}$ pertence a uma extensão.

Como ilustração, vamos mostrar como o raciocínio não monotônico pode ser representado no Theorist através do exemplo clássico da ave Tweety.

EXEMPLO III.1 - Inicialmente, vamos representar no Theorist o nosso conhecimento de que Tweety é uma ave através do fato :

$$\text{AVE(Tweety)} \tag{1}$$

e que as aves geralmente voam através do default :

$$\forall X\ \text{AVE}(X) \rightarrow \text{VOA}(X) \tag{2}$$

onde devemos ressaltar que esta informação é um default já que nem sempre as aves voam.

A partir destas duas informações, podemos concluir que Tweety voa :

$$\text{VOA(Tweety).}$$

Porém se acrescentarmos à nossa teoria dois novos fatos informando que:

- Tweety é um pinguim :

$$\text{PINGUIM(Tweety)} \tag{3}$$

- os pinguins são um espécie de ave :

$$\forall X\ \text{PINGUIM}(X) \rightarrow \text{AVE}(X) \tag{4}$$

e um novo default informando que :

- normalmente os pinguins não voam, ou seja :

$$\forall X\ \text{PINGUIM}(X) \rightarrow \neg\ \text{VOA}(X) \tag{5}$$

Podemos a partir destas novas informações, concluir também que Tweety não voa :

$$\neg \text{ VOA(Tweety)}.$$

Se todas estas assertivas fossem fatos, como ocorre na Lógica Clássica, teríamos uma contradição pois poderiamos concluir :

$$\text{VOA(Tweety) e } \neg \text{ VOA(Tweety)}$$

e uma contradição na lógica clássica trivializa toda a teoria, ou seja, podemos concluir qualquer coisa a partir desta teoria.

Neste sistema, não teremos esta contradição já que em uma extensão (definição III.3) teremos VOA(Tweety) e em uma outra extensão, $\neg$ VOA(Tweety). Isto significa que temos evidência a partir da teoria para as duas conclusões (em conjuntos de crenças distintos).

Porém, a conclusão VOA(Tweety) não é intuitiva pois sabemos que Tweety é um pinguim, que é uma sub-classe das aves, portanto mais específica. Assim, precisamos representar este conhecimento de que pinguim é uma exceção para o default $\forall X$ AVE(X) $\rightarrow$ VOA(X).

Isto é feito através do "constraint" :

$$\forall X \text{ PINGUIM(X)} \rightarrow \neg (\text{AVE(X)} \rightarrow \text{VOA(X)}) \qquad (6)$$

Desta forma, VOA(Tweety) não é mais explicável já que :

F (conjunto das fórmulas (1), (3) e (4)) $\cup$ D (fórmula (2)) $\models$ VOA(Tweety)

mas F $\cup$ D $\cup$ C (fórmula (6)) é inconsistente.

Isto é exatamente o ***Raciocínio Não-monotônico*** pois o acréscimo de novas informações ((3),(4),(5) e (6)) acarreta uma revisão da crença de informações consideradas até então verdadeiras (VOA(Tweety)). As ***Lógicas Não-monotônicas*** foram desenvolvidas para modelar este tipo de raciocínio [Lukaszewicz 90].

Na ***Lógica Clássica,*** infelizmente, isto não acontece pois uma de suas propriedades características é a ***monotonicidade*** :

Teorema III.2 - Sejam DB e DB` dois conjuntos de fórmulas e g uma fórmula . Logo, se DB $\subseteq$ DB` então { g | DB |= g} $\subseteq$ { g | DB' |= g}.

Concluindo na Lógica Clássica, novas declarações podem ser acrescentadas e com isso, novos teoremas podem ser provados, mas nenhuma delas fará com que uma declaração anteriormente conhecida ou provada torne-se inválida.

Devemos ressaltar que os "constraints" também são usados para bloquear explicações indesejáveis dos defaults.

Esta característica pode ser exemplificada pelo default $\forall X$ AVE(X) $\rightarrow$ VOA(X) pois ao descrevermos este default no Theorist, podemos gerar uma conclusão indesejável já que pela Lógica de 1ª ordem, as fórmulas :

$$\forall X \text{ AVE}(X) \rightarrow \text{VOA}(X)$$

$$\forall X \neg \text{VOA}(X) \rightarrow \neg \text{AVE}(X)$$

são equivalentes, ou seja, a contrapositiva de uma fórmula é logicamente equivalente a própria fórmula e o provador usado no Theorist, como será mostrado na seção III.2.3, utiliza não só os fatos e hipóteses mas também as suas contrapositivas para provar uma observação.

Deste modo, podemos explicar que Tweety não é um ave ($\neg$ AVE(Tweety)) a partir de uma teoria, que contenha somente a assertiva de que ele não voa ($\neg$ VOA(Tweety)) e do default $\forall X$ AVE(X) $\rightarrow$ VOA(X). O que neste caso, é uma conclusão indesejável. Para bloqueá-la, devemos criar o "constraint" que bloqueia o uso do default:

$$\forall X \neg \text{VOA}(X) \rightarrow \neg (\text{AVE}(X) \rightarrow \text{VOA}(X)) \qquad (7)$$

De modo análogo, devemos criar um "constraint" para o default $\forall X$ PINGUIM(X)$\rightarrow$ $\neg$ VOA(X), ou seja :

$$\forall X \text{ VOA}(X) \rightarrow \neg (\text{PINGUIM}(X) \rightarrow \neg \text{VOA}(X)) \qquad (8)$$

Assim, com a inserção dos "constraints" o sistema passa a explicar corretamente as observações evitando conclusões indesejáveis.

III.2.1 - Relação com a Lógica Default de Reiter :

Um dos sistemas de Raciocínio Não-monotônico mais conhecidos e investigados na literatura é a *Lógica Default de Reiter* [Reiter 80], onde uma teoria default é um par (W,D) onde W é o conjunto de fatos e D é um conjunto de defaults na forma $\frac{\alpha : \beta}{\gamma}$ (que pode ser interpretado intuitivamente como : "Se acreditamos em α e β é consistente então, acreditamos em γ").

Teorema III.3 [Reiter 80] - Seja (W,D), uma teoria default fechada. E é uma extensão default de (W,D) sse $E = \bigcup_{i=0}^{\infty} E_i$ onde :

$$E_0 = W \text{ e para } i \geq 0$$

$$E_{i+1} = Th(E_i) \cup \{ \gamma \mid \frac{\alpha:\beta}{\gamma} \in D,\ \alpha \in E_i ,\neg\beta \notin E\}$$

Dix [Dix 92] caracterizou o Theorist através da Lógica Default de Reiter, usando o seguinte mapeamento definido por :

$(F,H,C) \Rightarrow (W,D)$ com $W = F$ e $D = \{ \frac{(:(C' \wedge d))}{d} \mid d \in H \}$ onde C` é a conjunção de todos os elementos de C

A partir deste mapeamento, Dix mostrou o seguinte teorema :

TEOREMA III.4 [Dix 92 pp. 291] - Através de $(F,H,C) \Rightarrow (W,D)$, as extensões do Theorist de (F,H,C) são exatamente as extensões da teoria default de Reiter associada a (W,D).

III.2.2 - Linguagem de Programação:

A sintaxe do Theorist [Poole 91] é semelhante ao Prolog. Assim, temos dois tipos de expressões :

- ***Objetivo*** - é uma conjunção de Literais,
- ***Cláusula*** - é um Literal L ou L ← G, onde G é um Objetivo.

A Linguagem de Programação do Theorist [Poole 90], [Poole 88] permite as seguintes declarações:

- ***Fact f*** - onde f é uma cláusula.

  Este comando informa que f representa um fato para o Theorist e é nele que se deve representar as verdades incontestes do mundo modelado.

  Podemos exemplificá-lo pela afirmação de que a tensão de uma bateria B quando está em curto é igual a zero volts, ou seja :

  fact tensão(B,0) ← curto(B).

- ***default d*** - onde d é um nome (predicado com somente variáveis livres como argumento [Poole 91 pp. 5]).

  Ele indica que d é uma possível hipótese ou default para o Theorist. Como exemplo, podemos descrever o possível estado de curto de uma bateria B pelo :

  default curto(B).

- ***default d:w*** - onde d é um nome (predicado com somente variáveis livres como argumento) e w é uma cláusula. Este comando significa que w é uma possível hipótese denominada d.

  Como exemplo, podemos descrever a hipótese de que as aves normalmente voam da seguinte forma :

  default ave_voa(X) : voa(X) ← ave(X).

- ***constraint c*** - onde c é uma cláusula na forma n ← G onde G (Objetivo) é uma conjunção de literais e n é um nome de default. Ele indica que c é um "constraint" para o Theorist.

  Como exemplo, podemos utilizá-lo para eliminar a contrapositiva indesejável da hipótese ave_voa(X) mostrada acima, ou seja, impedir

que possamos provar ¬ave(X) a partir de ¬voa(X) através da declaração :

constraint ¬ave_voa(X) ← ¬voa(X).

- ***explain O*** - onde ***O*** é um objetivo.

  Uma explicação consistente para ***O*** é retornada a partir da execução deste comando e ele representa o conceito sobre a explicação de uma fórmula (definição III.2) vista anteriormente na seção III.2.

Devemos destacar que assim como no Prolog fatos, defaults e "constraints" estão implicitamente quantificados universalmente, enquanto que os "explains" estão implicitamente quantificados existencialmente.

Concluindo, para que o objetivo do Theorist seja concretizado (isto é, formar uma teoria que prove um determinado objetivo ***O*** a partir da tupla (F,H,C)), devemos incluir as declarações sobre o sistema no formato mostrado acima. Deste modo, ao procurarmos uma explicação para ***O*** (***explain O***), o Theorist responderá com as instâncias básicas das hipóteses necessárias para provar ***O*** a partir de um cenário (F,H,C).

Vamos então, descrever mais detalhadamente como o Theorist realiza este processo de procura de uma explicação durante a execução do comando ***explain O.***

III.2.3 - Descrição de Funcionamento :

A semântica do Theorist, descrita na seção III.2, permite implementar um sistema dedutivo, onde a construção das explicações consistentes para uma observação é feita em duas etapas [Poole *et. al.* 87].

Cada uma destas etapas [Poole *et. al.* 87], [Poole 91] utiliza um provador de teoremas de 1ª ordem, onde na primeira etapa, tentamos provar ***O*** usando os elementos de F e H como axiomas não lógicos, ou seja :

$$F \cup D \mid\!\!- O.$$

Para tal, é usado um provador de teoremas "backward" (para trás, a partir de ***O***) que procurará provar os sub-objetivos que for gerando, usando os fatos e as hipóteses relevantes. Desta maneira, geramos o subconjunto de instâncias básicas de elementos de H (conjunto ***D***) usado na prova.

A segunda etapa tem como objetivo verificar se a união deste conjunto D com os fatos e os "constraints" (conjunto C) é consistente. Para tal, basta estabelecer que :

$$F \cup D \cup C \not\mid\!\!- \neg d \text{ para cada } d \in D,$$

onde devemos assumir que o conjunto de fatos e "constraints" é consistente e o que desejamos realmente verificar é a consistência de cada uma das hipóteses de D [Poole *et. al.* 87] [Poole 91].

Devemos notar que durante a prova de ***O***, sempre que uma nova hipótese d for gerada, podemos realizar o seu teste de consistência desde que ***d*** esteja instanciado [Poole 91].

Esta característica é conseqüência da propriedade da semi-monotonicidade descrita na seção III.2 e permite melhorar a performance do Theorist durante a execução do comando **explain** ***O***.

Entretanto, quando uma hipótese ***d*** possui alguma variável livre (isto é, algum dos seus argumentos ainda não está instanciado), devemos esperar para testar a sua consistência somente após a execução da primeira etapa, ou seja, após a obtenção completa de uma prova para ***O***.

Quando isto acontecer, devemos substituir as variáveis livres de d por constantes únicas (constantes diferentes das existentes nos fatos, hipóteses e "constraints" que descrevem o sistema) e tentar provar a sua consistência [Poole 91].

EXEMPLO III.2 - Dado um sistema composto por :

- um conjunto F = { $\forall X$ (p(X) $\Rightarrow$ g), $\neg$p(a)} e
- um conjunto H = {p(X)}.

Neste sistema ao tentarmos explicar g, geraremos a possível hipótese p(X) onde X é uma variável livre. Deste modo para provar a sua consistência, devemos substituir X por uma constante única b (b ≠ a) e após esta substituição, podemos concluir que p(b) é uma explicação possível para g.

Devemos ressaltar que o provador de teoremas de 1ª ordem utilizado nestas duas etapas se baseia no procedimento MESON de provas [Loveland 78] e para ser implementado em Prolog necessita que [Poole *et. al.* 87] [Poole 91] :

- sejam geradas todas as contrapositivas de cada assertiva utilizada na descrição do sistema.

  Exemplo: Ao declararmos ***fact tensão(B,0)*** $\leftarrow$ ***curto(B)***, será criada a contrapositiva ***fact*** $\neg$***curto(B)*** $\leftarrow$ $\neg$***tensão(B,0).***
- durante o procedimento de prova de um objetivo ***O,*** possa ser utilizada a prova por absurdo, ou seja, ***O*** pode ser provada pela sua própria negação, ou pela negação de algum dos sub-objetivos gerados durante a busca da sua prova.

Estas duas exigências podem ser entendidas através de um simples exemplo em Prolog.

EXEMPLO III.3 - Suponha que um programa em Prolog é composto pelas seguintes cláusulas : $a \vee b$, $c \leftarrow a$ e $c \leftarrow b$ e queremos provar c.

Qualquer interpretador em Prolog irá dizer que isto não é possível apesar de a ou b serem verdadeiros e ambos implicarem em c.

Agora, se gerarmos as contrapositivas destes fatos :

$$a \vee b: \quad b \leftarrow \neg a \quad e \quad a \leftarrow \neg b,$$

$$c \leftarrow a: \quad \neg a \leftarrow \neg c \quad e$$

$$c \leftarrow b: \quad \neg b \leftarrow \neg c$$

podemos provar *c* por absurdo, ou seja, *c* é provado por $\neg c$ do seguinte modo: *c* é provado por *a* ($c \leftarrow a$), *a* é provado por $\neg b$ ($a \leftarrow \neg b$), $\neg b$ é provado por $\neg c$ ($\neg b \leftarrow \neg c$) provando *c* por absurdo.

A implementação deste provador permitiu que a descrição do Theorist em Prolog se tornasse viável [Poole *et. al.* 87] [Poole 91], tendo sido então desenvolvido dois protótipos deste modelo :

- O 1º protótipo apresentado em [Poole *et. al.* 87], utilizava os conceitos mostrados anteriormente do Theorist em uma versão interpretada mas não inseria as contrapositivas dos fatos, nem implementava o comando "constraint".

  Já mostramos acima, que a inserção de contrapositivas é necessária para implementar um provador completo de teoremas de 1ª ordem. Para tal, acrescentamos esta característica ao protótipo.

  Porém com esta modificação, ao representarmos sistemas um pouco mais complexos (com mais de 20 assertivas entre fatos e hipóteses possíveis), este protótipo passou a demorar muito para explicar uma observação tornando seu desempenho sofrível.

  Apesar deste fraco desempenho, esta versão é extremamente válida para representar sistemas mais simples e pode ser vista com maiores detalhes no apêndice A.

- O 2º protótipo descrito em [Poole 91] apresenta uma versão compilada do Theorist com a inserção das contrapositivas mas, novamente sem o comando "constraint".

  Esta versão conseguiu resolver os problemas citados acima, apresentando um bom desempenho (tanto na representação de sistemas simples como na de sistemas mais complexos) e é mostrada mais minuciosamente no apêndice B.

Como o principal objetivo deste trabalho não é apresentar detalhamente a forma de implementação do Theorist e sim, analisar a validade desta técnica de Inteligência Artificial na diagnose de Sistemas de Potência, vamos mostrar agora como as duas abordagens mais utilizadas na literatura (***Diagnose***

Abdutiva e Diagnose baseada na Consistência) podem ser representadas no Theorist.

III.3 - Diagnose Abdutiva :

A *Diagnose Abdutiva* [Poole *et. al.* 87] [Poole 88a] tem por objetivo descobrir um conjunto de hipóteses que nos levam a explicar as observações, podendo ser formalmente descrita no *Sistema Theorist* como :

DEFINIÇÃO III.4 [Poole 89 pp. 1305] - Uma *diagnose abdutiva* pode ser caracterizada como sendo um conjunto minimal de hipóteses (ou seja, um conjunto minimal D de instâncias básicas de elementos de H) que juntamente com o conjunto de fatos implicam nas observações, onde:

- **F** - é responsável pela descrição de funcionamento do sistema que está sendo representado,
- **H** - é o conjunto de doenças, suposições de normalidade e de anormalidade.

Resumindo, a diagnose abdutiva é uma explicação minimal das observações.

Devemos ressaltar que os modelos de diagnose normalmente utilizam o seu conhecimento na forma :

Efeitos (ou Observações) $\Rightarrow$ *Causas (ou Problemas).*

Este tipo de conhecimento nos induz a pensar que as observações acabam originando os problemas quando na realidade é exatamente ao contrário.

Por isso, a estrutura do conhecimento da *Diagnose Abdutiva* é representada na forma [Poole 88a pp.1284-1285] :

Causas (ou Problemas) $\Rightarrow$ *Efeitos (ou Observações)*

onde o que necessitamos é saber mais profundamente sobre os problemas do sistema que queremos representar, para a partir deles, podermos explicar as observações.

Deste modo, as possíveis hipóteses serão as doenças ou suposições de normalidade ou de mau-funcionamento, que serão aceitas como parte da explicação para algum sintoma ou observação.

EXEMPLO III.4 - Considere um circuito com duas baterias em série. Suponha que a bateria opera normalmente entre 12 e 16 volts e apresenta apenas dois tipos de falha : *baixa carga* que indica que a bateria apresenta uma tensão entre zero volts e a tensão mínima normal de funcionamento (12 volts) e *curto* que indica que a bateria está em curto, ou seja, com tensão igual a 0 volts.

A partir desta descrição, podemos destacar as seguintes relações :

- TENSÃO(B,V) $\rightarrow$ significa que a tensão da bateria B é V.
- SÉRIE(B1,B2) $\rightarrow$ informa que as baterias B1 e B2 estão em série.
- BATOK(B,V) $\rightarrow$ indica que a bateria B está funcionando corretamente com a tensão igual a V.
- BAIXA_CARGA(B,V) $\rightarrow$ informa que a bateria B está funcionando em baixa carga, ou seja, com tensão V entre 0 e 12 volts.
- CURTO(B) $\rightarrow$ significa que a bateria B está em curto, ou seja, com tensão V igual a 0 volts.

Assim, podemos caracterizar o funcionamento normal de uma bateria através dos seguintes fatos e hipóteses :

$$\text{fact TENSÃO(B,V)} \leftarrow \text{BATOK(B,V)}$$

$$\text{fact } (V \geq 12 \wedge V \leq 16) \leftarrow \text{BATOK(B,V)}$$

$$\text{default BATOK(B,V)}$$

De modo análogo, podemos descrever as duas falhas possíveis de uma bateria através dos seguintes fatos e defaults :

$$\text{fact TENSÃO(B,V)} \leftarrow \text{BAIXA\_CARGA(B,V)}$$

$$\text{fact } (0 < V < 12) \leftarrow \text{BAIXA\_CARGA(B,V)}$$

$$\text{default BAIXA\_CARGA(B,V)}$$

$$\text{fact TENSÃO(B,0)} \leftarrow \text{CURTO(B)}$$

default CURTO(B)

O funcionamento de duas baterias em série é representado por :

fact TENSÃO(SÉRIE(B1,B2),V) ←

[TENSÃO(B1,V1) ∧ TENSÃO(B2,V2) ∧ (V = V1 + V2)]

Suponhamos que foi observado que a tensão das duas baterias em série é de 15 volts (TENSÃO(SÉRIE(B1,B2),15) e que desconhecemos a tensão individual de cada bateria.

Neste caso, se pedirmos uma explicação para esta observação (ou seja, explain(TENSÃO(SÉRIE(B1,B2),15)), teremos três diagnoses possíveis :

- { CURTO(B1), BATOK(B2, 15) },
- { BATOK(B1, 15) , CURTO(B2) } e
- { BAIXA_CARGA(B1,V),BAIXA_CARGA(B2,15-V)} para 0< V <12.

Devemos ressaltar que na diagnose abdutiva, quando criamos uma possível hipótese devemos parametrizá-la com todos os dados que justificam a sua utilização [Poole 88a pp.1306]. Deste modo, quando queremos descrever o comportamento de componentes mais complexos como o de uma porta lógica devemos nos preocupar em utilizar como parâmetros de suas possíveis hipóteses, todos os elementos necessários para sua explicação.

EXEMPLO III.5 - Considere um circuito com dois inversores em série (figura III.2), onde sabemos que um inversor funciona normalmente gerando na sua saída, a negação da sua entrada.

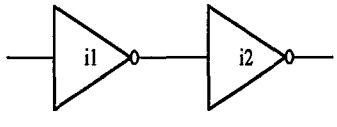


Figura III.2 : Circuito com dois inversores em série

Suponhamos para simplificar, que ele apresenta apenas um tipo de falha : *curto* que indica que a saída do inversor apresenta o mesmo valor que a sua entrada.

A partir desta descrição, podemos destacar as seguintes relações :

- INVERSOR(X) → significa que X é um inversor.
- OK(X, IN(X), OUT(X)) → informa que o inversor X está funcionando corretamente com a entrada IN(X) produzindo a saída OUT(X).
- CURTO(X, IN(X)) → indica que o inversor X está em curto com o valor de sua saída igual ao valor de sua entrada IN(X).

Assim, podemos caracterizar o funcionamento normal de um inversor através dos seguintes comandos :

fact [OUT(X) = ¬IN(X)] ← (INVERSOR(X) ∧ OK(X,IN(X),OUT(X)))

default OK(X,IN(X),OUT(X))

De modo análogo, podemos descrever a falha de curto no inversor através dos seguintes comandos :

fact [OUT(X) = IN(X)] ← (INVERSOR(X) ∧ CURTO(X,IN(X)))

default CURTO(X,IN(X))

A conexão entre os dois inversores é representada por :

fact OUT(I1) = IN(I2)

Suponha que foi observado que a entrada do circuito dos dois inversores em série é igual a 1 e a saída igual a zero, ou seja, IN(I1) = 0 e OUT(I2) = 1.

Neste caso, se pedirmos uma explicação para estas duas medidas (explain(IN(I1) = 0 e OUT(I2) = 1)) , teremos duas diagnoses possíveis :

- {CURTO(I1,0),OK(I2,0,1)} informando que apenas o inversor I1 está em curto e
- {OK(I1,0,1),CURTO(I2,1)} informando que apenas o inversor I2 está em curto.

Os exemplos III.4 e III.5 nos mostram que ao realizarmos a diagnose de um sistema, podemos fornecer explicações diferentes para um mesmo conjunto de observações.

Em geral, determinar qual é a melhor explicação para um conjunto de observações não é uma tarefa simples e depende de vários fatores. Dentre os tipos mais conhecidos de explicação, podemos destacar [Poole 87 pp. 17-18]:

- Explicação Minimal - explicação em que você utiliza o menor número de hipóteses (incluindo hipóteses de normalidade e anormalidade),
- Explicação Menos Específica - uma explicação E1 é menos específica que uma explicação E2 se E2 |= E1,
- Explicação Minimal de Anormalidades - explicação em que você utiliza o menor número possível de hipóteses que descrevem o funcionamento anormal do sistema. Caso duas ou mais explicações tenham este mesmo número de hipóteses de anormalidade, deve-se optar dentre estas, pela que possuir o menor número de hipóteses de normalidade.

Outros critérios de comparação de teoria podem ser vistos em [Popper 62] e [Quine *et. al.* 78 capítulo 6].

Concluindo podemos verificar pelos exemplos III.4 e III.5, que a representação na diagnose abdutiva é bastante simples.

Neste tipo de diagnose, o que precisamos é saber como funciona o sistema que queremos representar e a partir deste conhecimento, descrever o seu comportamento de forma bastante simples e intuitiva.

A representação do conhecimento neste tipo de diagnose é flexível e modular. Assim, podemos com relativa facilidade modificar e acrescentar novos fatos e hipóteses à sua base de conhecimento [Poole 89 pp. 308] [Poole 88a pp.1288].

Devido à estas propriedades, podemos concluir que a diagnose abdutiva é o método mais adequado para o diagnóstico de um sistema de potência, onde devemos destacar que esta escolha deveu-se principalmente :

- A maior importância que este tipo de diagnose dá ao funcionamento anormal do sistema [Poole 89] [Poole 88a] já que nosso maior objetivo neste trabalho é auxiliar o operador durante distúrbios, onde devemos dar maior destaque a descrição de falhas do que a descrição sobre o funcionamento normal do sistema.
- A facilidade de modificar e acrescentar novas informações à sua base de conhecimento já que é bastante normal em um sistema elétrico que ele sofra uma evolução com o passar do tempo (seção I.1 e II.2) e com isso, certamente surgirá a necessidade de mudanças na descrição de seu funcionamento.

III.4 - Diagnose Baseada na Consistência :

A ***Diagnose baseada na Consistência*** [Reiter 87] [de Kleer *et. al.* 87] tem por objetivo provar a anormalidade de alguns componentes a partir da detecção de um desvio do comportamento normal do sistema, notado através de uma ou mais observações. Ela pode ser formalmente descrita no Sistema Theorist como:

DEFINIÇÃO III.5 [Poole 89 pp. 1305] - A ***diagnose baseada na consistência*** pode ser caracterizada como sendo um conjunto minimal de hipóteses de anormalidade tal que as observações são consistentes com os demais componentes sendo ainda considerados normais onde :

- **F** $\rightarrow$ é responsável pela descrição de funcionamento do sistema que está sendo representado e pelo conjunto de observações
- **H** $\rightarrow$ é o conjunto de hipóteses de normalidade.

A estrutura do conhecimento na ***diagnose baseada na consistência*** é representada na forma [Poole 88a pp.1284] :

$$Observações \Rightarrow Anormalidades$$

onde a partir das observações sobre o sistema podemos concluir quais os componentes que estão funcionando anormalmente.

Assim, podemos verificar que o objetivo principal desta diagnose não é descobrir as causas que podem explicar as observações como ocorre na diagnose abdutiva mas, simplesmente identificar os componentes que devido ao seu funcionamento anormal geram um conflito entre o comportamento normal do sistema e as observações (esta forma de diagnose é denominada "excusing diagnosis") [Konolige 92].

Entretanto, podemos criar mecanismos na diagnose baseada na consistência que permitam que ela determine as causas que nos levam a explicar as observações (esta forma de diagnose é denominada diagnose explanatória).

Para tal, devemos adicionar a descrição do sistema axiomas fechados no seguinte formato :

$$E \Rightarrow C_1 \vee C_2 \vee \ldots \vee C_n$$

onde

E representa um efeito (sintoma) ou uma observação do sistema e

$C_1 \ldots C_n$ representa as possíveis causas ou problemas que podem explicar este efeito ou observação.

A partir da utilização deste mecanismo e criando certas restrições durante a descrição do sistema, podemos obter na diagnose baseada na consistência as mesmas explicacões conseguidas pela diagnose abdutiva como pode ser mostrado pelo seguinte teorema.

TEOREMA III.5 [Poole 88a pp.1286] - Se K1 é a base de conhecimento da diagnose baseada na consistência e K2 é a base correspondente de conhecimento da diagnose abdutiva então as explicações obtidas por esta diagnose baseadas em K2 são idênticas as conseguidas pela diagnose baseada na consistência a partir de K1.

Devemos ressaltar que esta conclusão também foi obtida em [Konolige 92 pp. 268-270].

A diagnose baseada na consistência é descrita com maiores detalhes no apêndice C e foi preterida pela diagnose abdutiva devido principalmente a maior importância que ela dá ao funcionamento normal do sistema ([Reiter 87] e [Poole 89]) e como já foi citado anteriormente na seção III.3, devemos dar maior destaque a descrição de falhas no diagnóstico de um sistema de potência.

Assim, iremos a seguir mostrar mais minuciosamente como devemos *representar o caso exemplo do capítulo II no sistema Theorist utilizando a diagnose abdutiva.*

III.5 - Representação e solução do caso exemplo :

Para representar o sistema elétrico do caso exemplo (seção II.2.2), devemos inicialmente estudar uma forma que descreva de modo simples mas completo, os alarmes e as falhas deste sistema.

Podemos notar, a partir da lista de descrição dos problemas deste sistema (tabelas II.1 e II.2), que um alarme pode ser caracterizado através de dois parâmetros :

- ***Componente*** → indica em qual elemento ocorreu o alarme. No caso exemplo, existem cinco componentes possíveis : ***linha_transmissão_01, linha_transmissão_02, transformador_01, transformador_02 e by_pass.***
- ***Tipo*** → fornece qual foi a forma de atuação do alarme. No nosso exemplo, existem sete tipos possíveis:

1. ***Abriu*** → associado a abertura do disjuntor. Pode ser aplicado a qualquer um dos componentes mostrados acima,
2. ***Fase*** → mostra que ocorreu sobrecorrente em uma das fases ***(f/f)*** da linha de transmissão. Pode estar associado, não só a linha de

transmissão mas também, ao disjuntor de by-pass quando este estiver operando no lugar do disjuntor da linha de transmissão.

3. ***Terra*** → informa que aconteceu sobrecorrente de terra ***(f/t)*** na linha de transmissão. Este tipo de alarme está associado aos mesmos componentes descritos para a fase.
4. ***Temporizado*** → significa que ocorreu uma falha distante à usina na linha de transmissão. Assim, como os dois anteriores, este tipo de alarme só pode ser aplicado a linha de transmissão e ao disjuntor de by-pass.
5. ***Instantâneo*** → revela que aconteceu uma falha próxima à usina na linha de transmissão, onde os componentes relacionados com este tipo de alarme são os mesmo mostrados na fase.
6. ***Sobrecarga*** → indica que ocorreu uma sobrecarga no transformador,
7. ***Auxiliar*** → informa que uma linha de transmissão ou um transformador está operando com o auxílio do disjuntor de by-pass.

Deste modo, vemos que um alarme pode ser representado pela seguinte descrição: ***alarme(Tipo,Componente)***. Como exemplo, a abertura do disjuntor da linha de transmissão 01 passa a ser representada por alarme(abriu, linha_transmissão_01).

De modo análogo, podemos definir que uma falha em um sistema de potência é a descrição de um problema originado por um conjunto de alarmes, representada a partir das seguintes características :

- ***Origem*** → fornece a origem de uma falha. No nosso caso, existem cinco origens possíveis : fase, terra, sobrecarga, barra_auxiliar e barra_principal,
- ***Local*** → indica a localidade da falha. Neste exemplo, existem dois locais disponíveis: próximo e distante (associados respectivamente aos tipos de alarmes instantâneo e temporizado).

- *Componente* → mostra qual o componente está associado à falha.
- *Componente_Aux* → informa qual o componente está operando com o auxílio do disjuntor de by-pass durante esta falha.

Assim, uma falha passa a ser caracterizada pela seguinte descrição : ***falha(Origem, Local, Componente, Componente_Aux)***. Como ilustração, a falha próxima na linha de transmissão 01 f/f(1) mostrada na Tabela II.1, passa a ser representada por falha(fase, próxima, by_pass, linha_transmissão_01).

A partir destas duas caracterizações, podemos descrever mais detalhadamente a representação do nosso caso exemplo no Theorist onde uma diagnose pode ser vista como sendo a determinação de um conjunto de falhas possíveis a partir de um dado conjunto de alarmes.

Exemplificando, ao pedirmos um diagnóstico para os alarmes de abertura do disjuntor, fase e instantâneo na linha de transmissão 01 (representados por : alarme(abriu, linha_transmissão_01), alarme(fase, linha_transmissão_01), alarme(instantâneo, linha_transmissão_01)), teremos pela tabela II.1 apenas uma explicação possível que é a falha próxima na linha de transmissão 01 f/f descrita por falha(fase, próxima, linha_transmissão_01, -).

Podemos notar pelas caracterizações mostradas acima, que ***a estrutura do conhecimento deste sistema elétrico no Theorist*** deve ser representada na forma:

$$Alarme \leftarrow Falha \text{ (ou seu equivalente } Falha \Rightarrow Alarme)$$

onde as falhas são as possíveis hipóteses que serão aceitas como explicação para os alarmes.

Deste modo, podemos representar a caracterização da falha próxima na linha de tranmissão 01 f/f através dos seguintes fatos e defaults :

$$fact\ alarme(abriu, linha\_transmissão\_01) \leftarrow falha(fase, próxima, linha\_transmissão\_01,-).$$

fact alarme(fase, linha_transmissão_01) ← falha(fase, próxima, linha_transmissão_01,-).

fact alarme(instantâneo, linha_transmissão_01) ← falha(fase, próxima, linha_transmissão_01,-).

default falha(fase, próxima, linha_transmissão_01, -)

Suponha, a partir desta descrição, que queremos um diagnóstico para os alarmes de abertura, fase e instantâneo na linha de transmisão 01, ou seja, desejamos uma explicação para estes três alarmes caracterizada pelo comando ***explain((alarme(abriu, linha_transmissão_01), alarme(fase, linha_transmissão_01), alarme(instantâneo, linha_transmissão_01))).***

Neste caso, o Theorist responderá com ***{ falha(fase, próxima, linha_transmissão_01, -) }, ou seja, a falha próxima na linha de transmissão f/f.***

Porém, durante um distúrbio em um sistema de potência, é possível que um alarme não seja atuado devido ao mau-funcionamento de algum equipamento. Para conviver com este problema, o operador só admite que uma falha esteja ocorrendo se um número mínimo de alarmes que a caracterizam, estiver atuado.

Para tal, precisamos criar um novo fato no Theorist que represente este "conhecimento" do operador (ou seja, que indique o número mínimo de alarmes que devem estar atuados para que uma falha deste sistema seja aceita como explicação). Devemos enfatizar que este novo fato representa o conhecimento heurístico, que é a base da maioria dos sistemas especialistas.

Ele foi denominado de ***mínima_caracterização*** e possui os mesmos parâmetros que representam uma falha, ou seja, origem, local, componente e componente_aux. Com isso, este novo fato passa a ser representado pela seguinte descrição: ***mínima_caracterização(Origem, Local, Componente, Componente_aux).***

Como ilustração, podemos dizer que a falha próxima na linha de transmissão 01 f/f só é considera válida para um operador se estiverem atuados pelo menos os alarmes de fase e instantâneo na linha de transmissão 01, ou seja, atuado_alarme(fase, linha_transmissão_01) e atuado_alarme(instantâneo, linha_transmissão_01). Esta característica pode ser expressa pelo fato :

fact mínima_caracterização(fase, próxima, linha_transmissão_01, -) ← (atuado_alarme(fase, linha_transmissão_01), atuado_alarme(instantâneo, linha_trasmissão_01))

Podemos notar pela descrição mostrada acima, que os alarmes passaram a ter duas formas de representação :

- ***atuado_alarme(Tipo, Componente)*** → caracteriza um alarme que está atuado no sistema elétrico e
- ***alarme(Tipo,Componente)*** → descreve um alarme para o qual o operador deseja uma explicação.

Estas duas representações foram criadas para descrever a realidade de um sistema de potência pois, como foi mostrado na seção II.2.1, muitos alarmes são gerados durante distúrbios, entretanto, dependendo do estado em que o sistema se encontre, apenas alguns destes são relevantes para o operador e devem ser explicados.

Assim, todos os alarmes gerados pelo sistema elétrico serão representados por ***atuado_alarme(Tipo, Componente),*** mas apenas aqueles que o operador deseja uma explicação, serão também descritos por ***alarme(Tipo, Componente).***

Já vimos acima que o fato ***mínima_caracterização*** deve ser usado como condição para que a falha próxima na linha de transmissão 01 f/f possa ser utilizada em uma explicação. Deste modo, devemos substituir ***o default falha(fase, próxima, linha_transmissão_ 01, -)*** utilizado na descrição anterior desta falha por uma hipótese mais genérica que permita caracterizar qualquer

falha, desde que a condição mínima de alarmes esteja satisfeita. Esta hipótese é representada pelo default :

default descrição_falha(Origem, Local, Componente, Componente_aux) : falha(Origem,Local, Componente, Componente_aux) ← mínima_caracterização(Origem, Local, Componente, Componente_aux)

Este novo default significa que podemos assumir a hipótese *descrição_falha(Origem, Local, Componente, Componente_aux)* desde que seja provado a fórmula :

falha(Origem,Local, Componente, Componente_aux) ← mínima_caracterização(Origem, Local, Componente, Componente_aux)

Esta fórmula representa justamente a restrição de só aceitar a *falha(Origem,Local, Componente, Componente_aux)* se a condição mínima de alarmes, representada por *mínima_caracterização(Origem, Local, Componente, Componente_aux)* for satisfeita.

Com isso, temos agora uma descrição mais completa para a falha próxima na linha de transmissão 01 f/f, representada pelos seguintes fatos e defaults :

fact alarme(abriu, linha_transmissão_01) ← falha(fase, próxima, linha_transmissão_01,-).

fact alarme(fase, linha_transmissão_01) ← falha(fase, próxima, linha_transmissão_01,-).

fact alarme(instantâneo, linha_transmissão_01) ← falha(fase, próxima, linha_transmissão_01,-).

fact mínima_caracterização(fase, próxima, linha_transmissão_01, -) ← (atuado_alarme(fase, linha_transmissão_01), atuado_alarme(instantâneo, linha_trasmissão_01))

default descrição_falha(Origem, Local, Componente, Componente_aux) : falha(Origem,Local, Componente, Componente_aux) ← mínima_caracterização(Origem, Local, Componente, Componente_aux)

A partir desta nova representação, se pedirmos uma justificativa para a atuação dos alarmes de abertura do disjuntor e fase na linha de trasmissão 01, ou seja, ***explain((alarme(abriu, linha_transmissão_01), alarme(fase, linha_transmissão_01)))***, o Theorist responderá que não é possível encontrar uma explicação para esses alarmes.

Entretanto, se somente os alarmes de fase e instantâneo na linha de transmissão 01 estiverem atuados e pedirmos uma explicação ao Theorist para estas atuações (isto é, ***explain((alarme(fase, linha_transmissão_01), alarme(instantâneo, linha_transmissão_01)))***, ele responderá com ***{ descrição _falha(fase, próxima, linha_transmissão_01, -) }.***

Neste exemplo, mostramos que mesmo sem a atuação do alarme de abertura do disjuntor na linha de transmissão 01, a falha próxima na linha de transmissão 01 f/f é aceita como explicação já que o seu número mínimo de alarmes atuados foi satisfeito, ou seja, ***atuado_alarme(fase, linha_transmissão_01) e atuado_alarme(instantâneo, linha_transmissão_01).***

Assim, verificamos que é bastante simples utilizar o Theorist na diagnose de um sistema de potência. Com isso, para descrever os distúrbios das tabelas II.1 e II.2, precisamos apenas representar os alarmes associados a cada falha na forma : ***Alarme*** ← ***Falha*** e estabelecer a condicão mínima de alarmes para que uma falha seja aceita como explicação através do fato ***mínima_caracterização.***

Devemos ressaltar que o uso dos "contraints" também é necessário na representação deste sistema elétrico como pode ser exemplificado pelas falhas de sobrecarga no transformador 01 com e sem by-pass (Tabela II.2) pois um operador do sistema geralmente considera válida a falha de sobrecarga no transformador 01 se pelo menos o alarme de sobrecarga no transformador 01 estiver atuado. Já em relação a falha de sobrecarga no transformador 01 (3), o operador a considera válida desde que pelo menos os alarmes de sobrecarga e auxiliar no transformador 01 estejam atuados. Estas duas caracterizações são representadas do seguinte modo :

fact((mínima_caracterização(sobre,-,transformador_01,-) ←
(atuado_alarme(sobre,transformador_01))))).
fact((mínima_caracterização(sobre,-,by_pass,transformador_01) ←
(atuado_alarme(sobre,transformador_01),
atuado_alarme(aux,transformador_01)))).

Podemos verificar por estas duas representações que se pedirmos ao Theorist uma explicação para a atuação dos alarmes de sobrecarga e auxiliar no transformador 01, ou seja, *explain((alarme(sobre, transformador_01), alarme(aux, transformador_01)))*, ele responderá que existem duas explicações possíveis:

- *{ descrição _falha(sobre, -, by_pass, transformador_01) },*
- *{ descrição _falha(sobre, -, by_pass, transformador_01) } e { descrição _falha(sobre, -, transformador_01,-) }.*

A primeira explicação está correta mas a segunda é indesejável já que um transformador não pode apresentar ao mesmo tempo uma falha de sobrecarga com e sem o auxílio de by-pass. Deste modo, devemos inibir uma explicação que inclua a falha de sobrecarga no transformador 01 sem o auxílio do by-pass quando o alarme de auxiliar no transformador estiver atuado. Para tal, devemos criar um "constraint" que represente esta restrição caracterizado por:

constraint((descrição_falha(sobre,-,transformador_01,-) ←
atuado_alarme(aux,transformador_01))).

A descrição completa deste sistema elétrico é mostrada em detalhes no apêndice D e serviu de base para o desenvolvimento de um protótipo capaz de determinar todos os diagnósticos possíveis deste nosso caso exemplo para um dado conjunto de alarmes. Vamos agora, descrever um pouco o funcionamento deste protótipo.

Sabemos que para realizar uma diagnose, devemos inicialmente fornecer o conjunto de alarmes atuados para os quais estamos interessados em procurar uma explicação. No nosso protótipo, este procedimento é realizado através da seguinte interface :

Descreva os alarmes para a diagnose :

Componente :

Tipo de Atuação :

onde o usuário digita o nome do componente envolvido e o tipo de atuação do alarme desejado.

Ao fim desta operação, procuramos as explicações possíveis para este conjunto de alarmes atuados através da execução do comando ***explain(conjunto de alarmes),*** onde teremos como resposta todos os diagnósticos possíveis deste conjunto.

Assim, se pedirmos para o protótipo justificar a atuação dos alarmes de abertura do disjuntor, terra e temporizado na linha de transmissão 02, ele nos fornecerá apenas uma explicação no seguinte formato :

Diagnóstico de um Sistema de Potência

Número total de explicações : 1

Explicação 1 :

Falha Possível :

- **Origem : terra (f/t)**
- **Local : distante**
- **Componente : linha_transmissão_02**
- **Componente_aux : -**

Devemos ressaltar que se não for possível encontrar um diagnóstico para um dado conjunto de alarmes, o protótipo informará ao operador que não existe explicação para tal conjunto de dados.

Este protótipo pode ser visto em maiores detalhes no apêndice E e foi totalmente desenvolvido em Prolog, onde a representação no Theorist das falhas do caso exemplo foi implementada utilizando a sua versão compilada.

Para verificar a validade do Theorist, nosso protótipo foi submetido aos seguintes testes de diagnóstico :

- ***Falhas simples*** → Neste teste, apresentamos ao protótipo, todos os conjuntos de alarmes característicos de falhas simples (num total de 29) e ele mostrou a explicação correta em todas as situações.
- ***Falhas simples com ruído*** → Neste caso, foi fornecido ao protótipo todos os conjuntos incompletos de alarmes característicos de falhas simples (num total de 92 conjuntos diferentes) e não foi encontrado erro em nenhumas das explicações destes conjuntos.
- ***Falhas duplas*** → Mostramos ao protótipo vários conjuntos de alarmes (aproximadamente 100), que descrevem falhas duplas e ele respondeu corretamente em todos os casos.

  Devemos notar que denominamos de falha dupla, à conjunção de duas falhas simples.
- ***Falhas duplas com ruído*** → Durante este teste, o índice de erro ao fornecermos ao protótipo 70 conjuntos incompletos de alarmes característicos de falhas duplas, também foi nulo.

Estes resultados são bastante promissores. Entretanto, para verificar a sua eficiência, devemos nos preocupar também com a sua velocidade de processamento. Para tal, geramos uma versão executável deste protótipo e o seu

desempenho em um computador IBM PC-386/SX 25 Mhz pode ser considerado bastante satisfatório como nos mostra a tabela III.1.

Devemos notar que o tempo de resposta do protótipo aumenta em função do número de alarmes apresentados. Um estudo de complexidade das Lógicas Não-monotônicas pode ser visto em [Cadoli *et. al.* 93].

Concluindo, podemos verificar a partir de todos os testes e resultados obtidos que ***a Lógica Não-monotônica, representada pela Diagnose Abdutiva e o Sistema Theorist, é uma técnica de Inteligência Artificial adequada ao diagnóstico de sistemas de potência.***

Conjunto de Alarmes	Explicações Obtidas	Desempenho do Theorist
alarme(fase,linha_transmissão_ 01) alarme(instantâneo,linha_transmissão_01)	- Falha próxima na linha de transmissão 01 f/f	t = 0,54 seg
alarme(abriu,linha_transmissão_01) alarme(instantâneo,linha_transmissão_01) alarme(terra,linha_transmissão_01) alarme(abriu,transformador_01) alarme(sobre,transformador_01)	- Falha próxima na linha de transmissão 01 f/t - Falha no transformador 01	t = 1 seg
alarme(abriu,linha_transmissão_02) alarme(temporizado,linha_transmissão_02) alarme(fase,linha_transmissão_02) alarme(abriu,by-pass) alarme(aux,transformador_01) alarme(sobre,transformador_01)	- Falha distante na linha de transmissão 02 f/f - Falha na barra auxiliar (3) - Falha no transformador 01 (3)	t = 2 seg
alarme(abriu,linha_transmissão_01) alarme(fase,linha_transmissão_01) alarme(instantâneo,linha_transmissão_01) alarme(terra,linha_transmissão_ 02) alarme(temporizado,linha_transmissão_02) alarme(sobre,transformador_01) alarme(abriu,by-pass) alarme(aux,transformador_02) alarme(sobre,transformador_02)	- Falha próxima na linha de transmissão 01 f/f - Falha distante na linha de transmissão 02 f/t - Falha no transformador 01 - Falha na barra auxiliar (4) - Falha no transformador 02 (4)	t = 4,4 seg

Tabela III.1 : Teste de desempenho do Theorist

Capítulo IV

Redes Neurais

IV.1 - Aspectos Biológicos :

O *Cérebro* humano possui dezenas de bilhões de neurônios densamente interconectados *(Redes Neuronais)*, que demonstram grande capacidade para armazenar e processar informações.

Os neurônios (figura IV.1) são constituídos de uma estrutura básica, formada por um corpo celular (soma), um axônio e diversos dendritos. O axônio liga o corpo celular de sua célula a outros neurônios e os dentritos recebem as terminações dos axônios de outros neurônios.

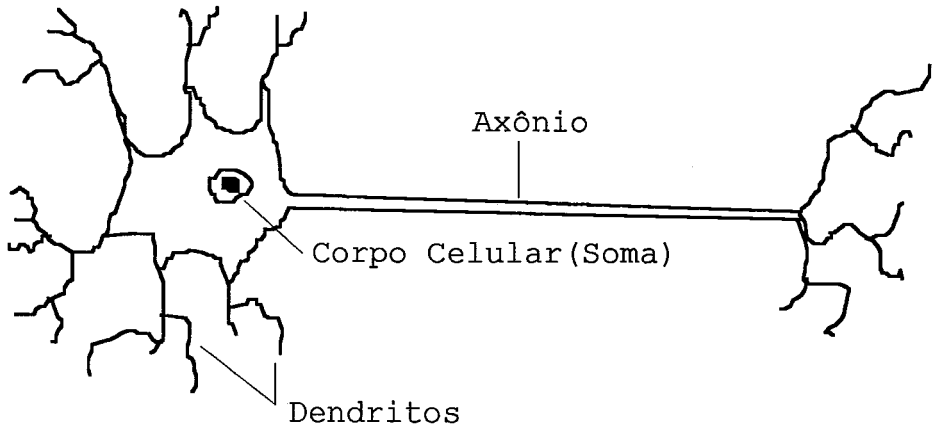


Figura IV.1 : Modelo de um Neurônio Biológico

Deste modo, a conexão entre os neurônios, denominada sinapse (figura IV.2), é formada pelo encontro da terminação axônica de um neurônio com o corpo celular ou dendrito de outro neurônio. Elas, além de permitirem a união de vários neurônios, funcionam como "válvulas" controladoras do fluxo de informação em uma rede neuronal.

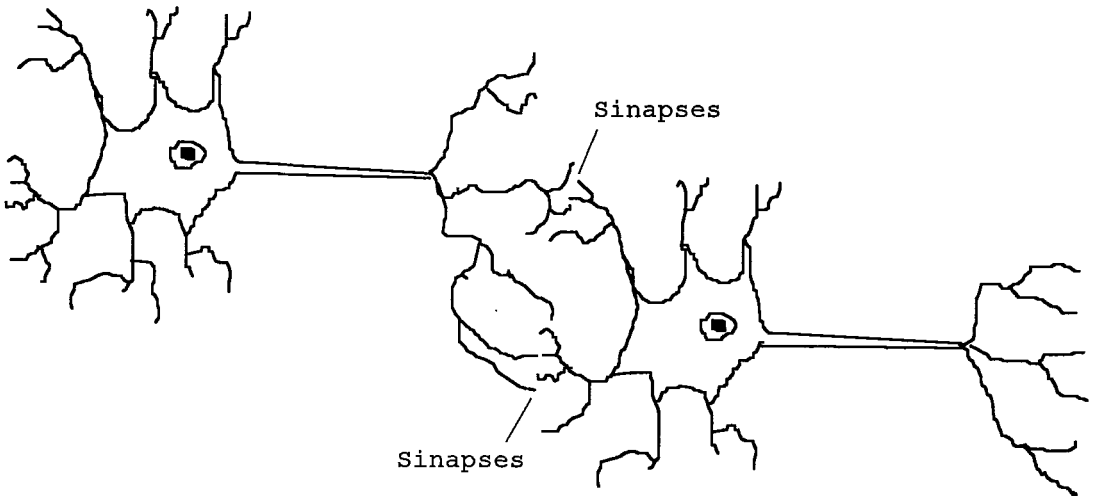


Figura IV.2 : Conexões entre neurônios - Sinapses

Existem dois tipos de sinapse : as excitatórias e as inibitórias. As sinapses excitatórias fornecem um sentido de cooperação entre os neurônios, permitindo a passagem de informações entre eles. Já as sinapses inibitórias dificultam ou coibem a passagem de informações, sugerindo uma idéia de competição entre as mesmas.

Cada sinapse tem ainda uma eficiência diferente (peso sináptico). Assim, uma única saída (axônio) de um neurônio pode sensibilizar diferentemente os outros neurônios aos quais está conectado.

A capacidade do ser humano de entender, lembrar, memorizar, generalizar, entre outras tarefas, parece estar intrinsicamente relacionada com o tipo e o peso das sinapses existentes nas redes neuronais.

O conhecimento sobre o funcionamento de um neurônio, infelizmente, ainda não é o bastante para explicar estas tarefas extremamente complexas que o cérebro é capaz de realizar.

Entretanto, o que deve ser destacado é que o cérebro realiza todas essas tarefas de forma eficaz, apesar dos neurônios serem sensivelmente mais lentos que os processadores usados nos computadores atualmente. Possivelmente, toda essa eficiência não está na capacidade individual de um neurônio e sim, no

conjunto formado pela quantidade muito grande deles (dezenas de bilhões) e pela forma como eles estão conectados.

Estes são os principais motivos do homem tentar imitar o modo de funcionamento do cérebro, procurando reproduzir artificialmente as redes neuronais biológicas e usá-las como ferramenta de computação em diversas áreas.

IV.2 - Redes Neurais Artificiais :

Ao contrário dos sistemas especialistas utilizados na Inteligência Artificial convencional, onde o conhecimento está representado na forma de regras e algoritmos, as ***Redes Neurais Artificiais*** [Rumelhart *et. al.* 86] aprendem através de exemplos, ou seja, o problema é modelado informalmente através da apresentação exaustiva de casos típicos.

A partir desta apresentação, as redes tendem a generalizar o seu conhecimento e passam a responder corretamente a casos novos, desde que parecidos com os exemplos "aprendidos". Assim, mesmo diante de entradas incompletas ou inesperadas, a rede tende a fornecer uma saída razoavelmente correta.

Esta generalização advém não só da capacidade de aprendizado das redes mas também da "memória" da rede estar distribuida, ou seja, o conhecimento está espalhado pelos diversos neurônios na forma de pesos.

A distribuição de sua memória, torna as Redes Neurais Artificiais tolerantes a falhas, ou seja, a perda de alguns neurônios não trará como conseqüência a destruição de uma quantidade considerável de informação.

Outra característica bastante interessante é a sua capacidade de processamento paralelo (originada da densa rede de conexões entre os neurônios) que a torna bastante útil para aplicações em tempo real devido à alta velocidade que o processamento paralelo proporciona.

Tais propriedades, fazem com que as Redes Neurais Artificiais possuam um escopo muito grande de aplicações, tais como compressão de dados, reconhecimento de padrões, processamento de sinais, otimização combinatória, processamento de linguagem, etc ...

A seguir, iremos descrever mais detalhadamente o modelo de um neurônio artificial e seus conceitos básicos.

IV.2.1 - Conceitos Básicos :

Uma ***rede neural artificial*** é composta por vários neurônios (também denominados elementos de processamento) organizados em grupos chamados de ***camada***. As camadas podem ser classificadas em três tipos :

- ***Camada de Entrada*** → é aquela em que as informações são apresentadas à rede,
- ***Camada de Saída*** → é a camada em que se visualiza os resultados, ou seja, mostra a resposta da rede a uma entrada apresentada,
- ***Camada Escondida ou Intermediária*** → são todas as camadas existentes entre a camada de saída e a camada de entrada.

Tipicamente, a topologia de um rede neural (figura IV.3) consiste em uma seqüência de camadas interligadas completamente ou aleatoriamente com as camadas adjacentes. Se a informação entre essas camadas é propagada num único sentido (ou seja, da camada de entrada para a camada de saída), a rede neural é denominada direta ("feedforward"). Entretanto, se a informação propagada realimenta alguma camada anterior, a rede é denominada recursiva ou realimentada.

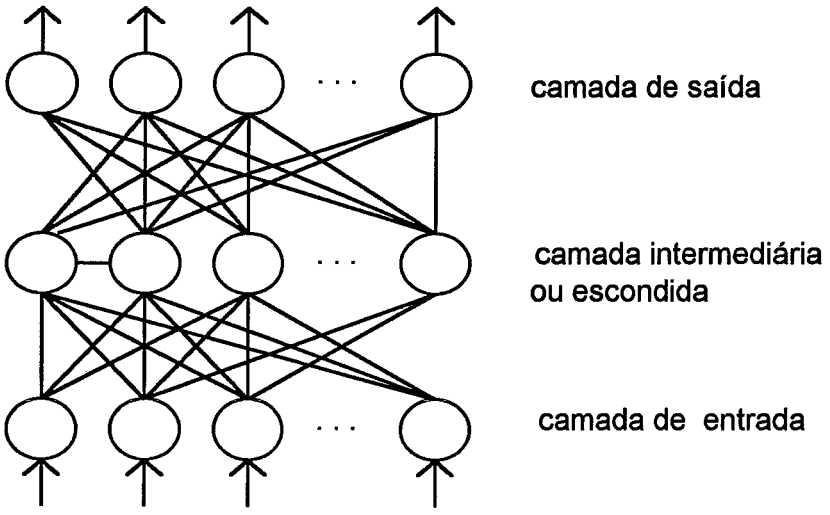


Figura IV.3 : Organização em camadas de um Rede Neural qualquer

As ligações entre neurônios (aqui, também chamadas de sinapses), são responsáveis pela comunicação entre eles e podem ser subdivididas em três classes :

- ***Inter-camadas*** → é usada para ligar dois neurônios de camadas diferentes,
- ***Lateral*** → é aquela que conecta dois elementos de uma mesma camada,
- ***Recursiva*** → tipo de conexão na qual o neurônio é ligado consigo mesmo.

Cada ligação possui ainda um peso, podendo assumir qualquer valor real. O módulo do peso indica a força da conexão, ao passo que o sinal, indica o tipo de ligação. Pesos com valores positivos correspondem a sinapses excitatórias e com valores negativos representam as sinapses inibitórias.

O ***Padrão de Conexão (Eficiência Sináptica)*** é a base do conhecimento da rede e é caracterizada pelos tipos de ligações entre os neurônios e seus respectivos pesos. Ela é usualmente representada por uma matriz de conexão **W** onde cada w_{ji} corresponde ao valor do peso da conexão que sai do neurônio i e

vai para o neurônio j. Caso não exista conexão, w_{ji} terá valor nulo. Assim, a linha j = 1,...,n representa o dendrito do neurônio j e a coluna i = 1,...,n representa o axônio do neurônio i.

Uma representação normalmente usada para o neurônio artificial está apresentada na figura IV.4, onde o neurônio j é caracterizado pelas excitações provenientes das entradas da rede ou saídas de outros neurônios, representadas por $X_i(t)$ (i = 0,..., N); pelos pesos das conexões existentes entre cada entrada e o neurônio j, representadas por w_{j1} ,..., w_{jN}; e também pela sua saída $S_j(t)$.

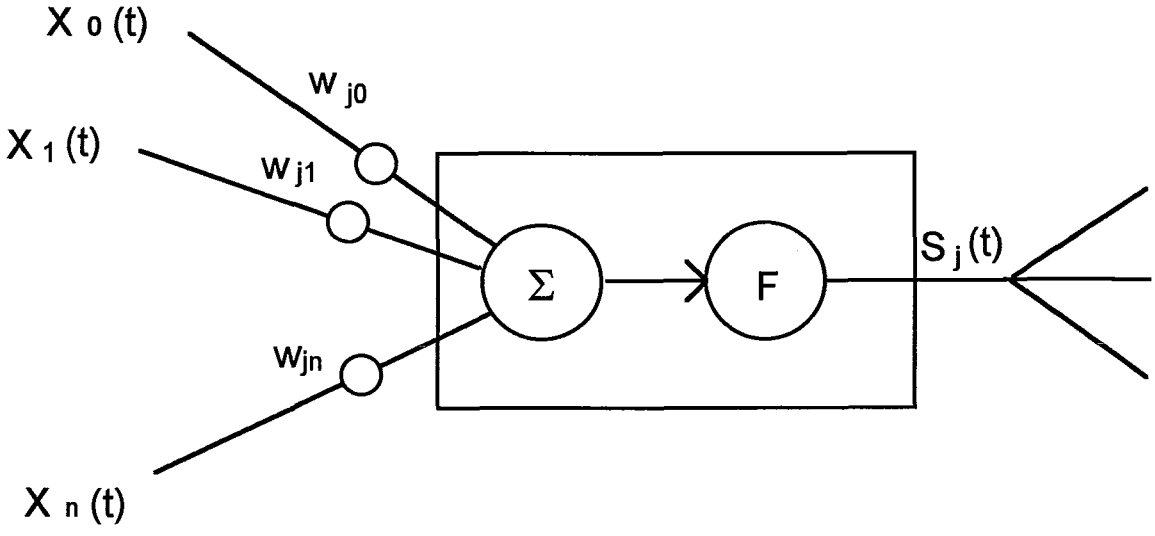


Figura IV.4 : Modelo Básico de um Neurônio

A maneira como o sinal de saída de um neurônio é transmitido para os demais é dado pela regra de propagação. Geralmente, a ***regra de propagação do neurônio j em um instante t, denominada $P_j(t)$ ou $NET_j(t)$,*** combina todos os valores no instante t das saídas conectadas ao neurônio j ($X_0(t)$ ... $X_n(t)$) com os pesos das respectivas conexões (W_{j0} ... W_{jN}) e soma este resultado a uma constante θ_j que simula o potencial limiar do neurônio biológico (ponto a partir do qual o neurônio transita de estado) :

$$NET_j(t) = \sum_{i=0}^{N} W_{ji} X_i(t) + \theta_j \qquad \text{(IV.1)}$$

Cada neurônio j também tem um ***estado de ativação $a_j(t)$***, que expressa o grau de excitação ou inibição do neurônio. A propagação dos sinais pela rede provoca uma mudança no estado de ativação dos neurônios, que é calculada através de uma regra de ativação.

Assim, o estado atual de um neurônio j é obtido a partir do estado de ativação anterior $a_j(t-1)$, do impulso total de entrada $NET_j(t)$ e de uma função F, que é denominada regra de ativação :

$$a_j(t) = F(a_j(t-1), NET_j(t)) \qquad \text{(IV.2)}$$

ou simplificando :

$$a_j(t) = F(NET_j(t)) \qquad \text{(IV.3)}$$

Existem várias funções de ativação (figura IV.5), dentre as quais podemos destacar :

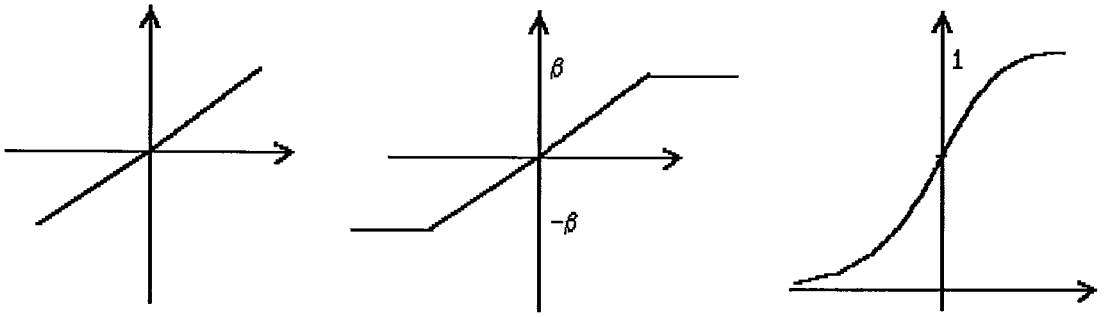


Figura IV.5 : Exemplos de funções de ativação

A esse estado de ativação do neurônio é aplicada uma função de saída f, que produz ***o sinal de saída do neurônio*** :

$$S_j(t) = f(a_j(t)) \qquad \text{(IV.4)}$$

Além desses conceitos básicos, a maioria dos modelos de redes neurais possuem ainda uma regra de aprendizado que fornece os mecanismos necessários para que a rede adquira "conhecimento", ou seja, aprenda ou se adapte a novas condições. Assim, vamos agora descrever mais detalhadamente esta característica importante, que é o aprendizado.

IV.2.2 - Aprendizado :

O ***aprendizado em uma rede neural artificial*** pode ser definido como uma mudança na matriz de conexão W (que representa o padrão de conectividade da rede mostrado na seção anterior). Este ajuste é realizado através de uma regra de aprendizado, a qual determina a forma de correção dos pesos das ligações, dado por uma matriz Δ_W, onde cada Δ_{Wji} é a mudança a ser realizada no peso da conexão que parte do neurônio i para o neurônio j.

Essas regras de aprendizado podem ser divididas em :

- ***Aprendizado Associativo*** → a rede aprende a gerar um padrão de saída particular a partir de um padrão de entrada apresentado. O aprendizado associativo pode ser subdividido em :

1. ***Auto-associador*** → neste tipo, a rede aprende a associar o padrão de entrada a ele mesmo. Assim, quando um padrão defeituoso é apresentado, a rede neural artificial é capaz de recuperar o padrão original.
2. ***Hetero-associador*** → a rede mapeia um ***padrão de entrada*** (***e***) em um ***padrão de saída*** (***s***), ajustando os pesos de tal maneira que, sempre que for apresentado o padrão de entrada ***e***, a rede mostre o padrão de saída ***s***.

- ***Detector de Regularidades*** → a rede aprende a descobrir características comuns no conjunto de entrada, separando-os em

categorias. Neste tipo de aprendizado, a rede utiliza como forma de treinamento o método não-supervisionado, onde somente o conjunto de entrada é apresentado.

Na etapa de treinamento da rede, os pesos são corrigidos de modo que as saídas convirjam para o padrão desejado. Existem duas ***formas de treinamento*** de uma rede neural artificial : ***a supervisionada e a não-supervisionada.***

No ***método supervisionado*** são informados à rede, além dos valores de entrada, os padrões de saída desejados. Assim, a rede computa os novos pesos calculando os erros relativos entre este padrão desejado e o padrão de saída gerado. Esta técnica é conhecida como correção por erro (figura IV.6).

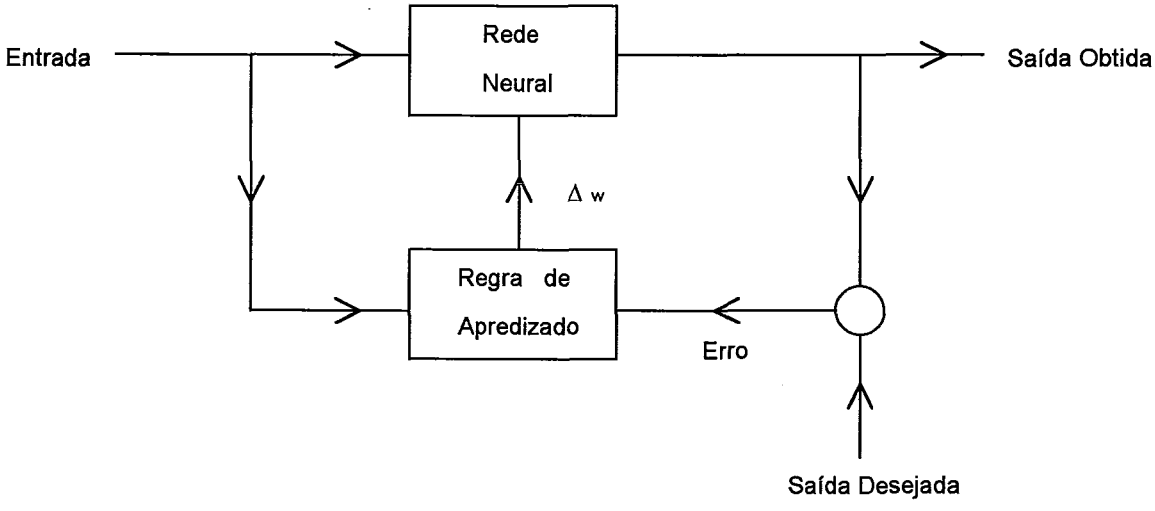


Figura IV.6 : Método supervisionado por correção por erro

Deste modo, o processo de aprendizado baseia-se na apresentação exaustiva dos padrões de entrada até que os pesos assumam valores estáveis (ou seja, o padrão de saída gerado convirja para o padrão de saída desejado). Porém, nem sempre isto acontece, e algumas das regras de aprendizado geram problemas de indefinição ou oscilação no padrão de conexão da rede (Δ_W).

Um dos ***processos de correção por erro*** mais utilizados é a ***regra delta***, na qual os pesos são modificados visando minimizar a diferença entre a saída obtida pela rede (*s*) e a saída desejada (*d*). Normalmente, a regra delta é dada por :

$$\Delta W_{ji} = \eta\ (d_j - s_j)\ e_i \qquad \text{(IV.5)}$$

onde

e_i é o estímulo de entrada fornecido pelo neurônio i,

d_j é a saída desejada para o neurônio j,

s_j é a saída efetiva do neurônio j,

η é uma constante, denominada ***taxa de aprendizado*** ($0 < \eta << 1$)

Já no ***aprendizado não-supervisionado***, somente a entrada é apresentada à rede. Deste modo, a rede aprende sem possuir conhecimento prévio da saída desejada.

A ***primeira regra de aprendizado*** foi proposta em 1949 e é conhecida como ***regra de Hebb***. Esta regra é um método não-supervisionado, onde o peso da conexão é incrementado quando os neurônios desta conexão apresentarem o mesmo estado de ativação (inibido ou excitado). Caso contrário, o peso é decrementado.

Assim, a variação da eficiência sináptica ΔW_{ji} entre os neurônios N_i e N_j é proporcional ao produto dos estados de ativação destes neurônios :

$$\Delta W_{ji} = \eta\ a_j(t)\ a_i(t) \qquad \text{(IV.6)}$$

onde

$a_i(t)$ é o estado de ativação do neurônio i,

$a_j(t)$ é o estado de ativação do neurônio j,

η é uma constante, denominada ***taxa de aprendizado*** ($0 < \eta << 1$)

Agora que já apresentamos os conceitos e as noções básicas sobre redes neurais, podemos fazer um estudo de um dos mais utilizados e interessantes modelos de rede neural, que é o modelo "Back-propagation".

IV.2.3 - Modelo "Back-propagation" :

O ***modelo "Back-propagation"*** originou-se de pesquisas independentes de vários cientistas, tais como Bryson e Ho em 1969, Werbos em 1974 e Parker em 1985 mas foi em 1986 que Rumelhart, Hinton e Willians conseguiram mostrar e descrever de maneira clara e concisa, o algoritmo "back-propagation", tornando-o *o modelo mais popular e utilizado para aplicações em computação neural.*

O "Back-propagation" é um modelo de múltiplas camadas (camada de Entrada, camada de saída e pelo menos uma camada intermediária) onde só são permitidas ligações inter-camadas de tal modo que todos os neurônios de uma camada estão totalmente conectados aos neurônios da camada imediatamente superior. Deste modo, qualquer neurônio de uma camada pode se comunicar com qualquer neurônio da camada seguinte e somente com estes. Esta estrutura não permite realimentação do fluxo de ativação dos neurônios e por isso, o modelo "Back-propagation" é uma rede neural "feedforward".

A ***regra de propagação*** adotada é a mesma mostrada anteriormente (sigma):

$$NET_j(t) = \sum_{i=0}^{N} W_{ji} X_i(t) \qquad (IV.7)$$

onde

$X_i(t)$ é o valor da saída do Neurônio i exceto se i é um neurônio da camada de entrada. Neste caso, $X_i(t)$ passa a ser o componente i do vetor de entrada da rede (e_i).

A função de saída usada é a identidade. Assim, a ***saída de um neurônio*** é igual ao seu próprio estado de ativação :

$$S_j(t) = a_j(t) \qquad \text{(IV.8)}$$

A ***função de ativação*** normalmente utilizada no modelo "Back-propagation" é a ***função sigmoidal*** (figura IV.7). Os limites geralmente utilizados nesta função são: 0 e 1 (logística) ou ainda -1 a +1 (tangente hiperbólica).

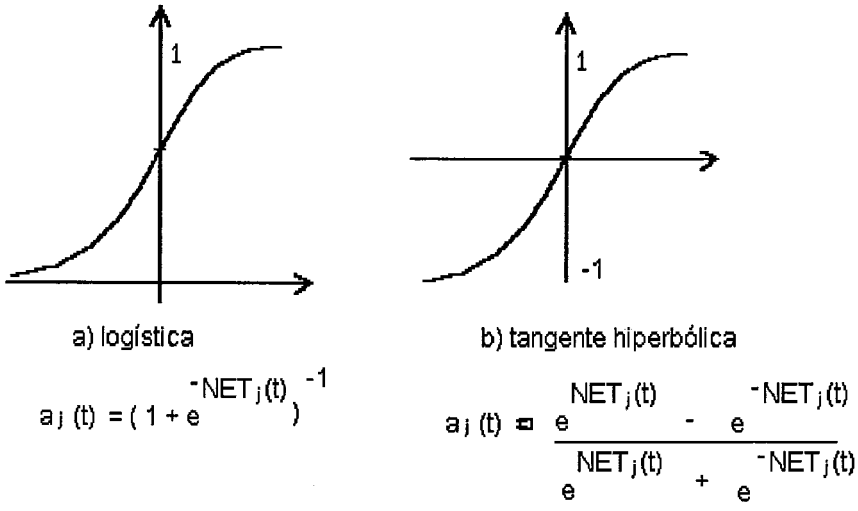


Figura IV.7 : Exemplos de funções de ativação do modelo "Back-propagation"

Neste modelo, o ***treinamento é supervisionado***. Assim, os valores de saída gerados pela rede são comparados com os padrões de saída desejados. Caso não haja diferença entre eles, não é realizado nenhum ajuste. Caso contrário, é utilizado a retropropagação do erro para o cálculo do valor dos pesos das conexões entre os neurônios visando minimizar o erro gerado na saída da rede.

Como o "Back-propagation" é um modelo de múltiplas camadas, a ***regra de aprendizado utilizada é a regra delta generalizada,*** que nada mais é do que a regra delta estendida para os neurônios das camadas intermediárias.

A partir da regra delta generalizada, podemos descrever o algoritmo do modelo "Back-propagation" do seguinte modo :

a) Escolhe-se os padrões de entrada e saída desejada que serão utilizados no treinamento.

b) Seleciona-se os parâmetros característicos da rede :

- Função de ativação (função de transferência),
- Número de camadas intermediárias,
- Número de neurônios por camada,
- Valor da taxa de aprendizado, etc...

c) Apresenta-se os padrões de entrada, propagando-os através da rede até a camada de saída, obtendo-se assim, os valores de saída para cada neurônio da rede.

d) Para cada neurônio j da camada de saída, é calculado seu erro δ_j (comparando a componente j do vetor de saídas desejadas (d_j) com a mesma componente da saída efetiva (s_j)), através da fórmula :

$$\delta_j = (d_j - s_j)\, f'(net_j) \qquad \text{(IV.9)}$$

onde f ' é a derivada da função de ativação (função de transferência).

e) Para a camada intermediária imediatamente anterior a camada de saída, o erro δ_j é propagado, obtendo-se o erro δ_i para cada neurônio desta camada definido por:

$$\delta_i = \Sigma\, \delta_j\, w_{ji}\ \ f'(net_j) \qquad \text{(IV.10)}$$

Em seguida, calcula-se o erro dos neurônios da camada imediatamente anterior a esta, retropropagando o erro δ_i como mostra a figura IV.8, utilizando novamente a equação IV.10.

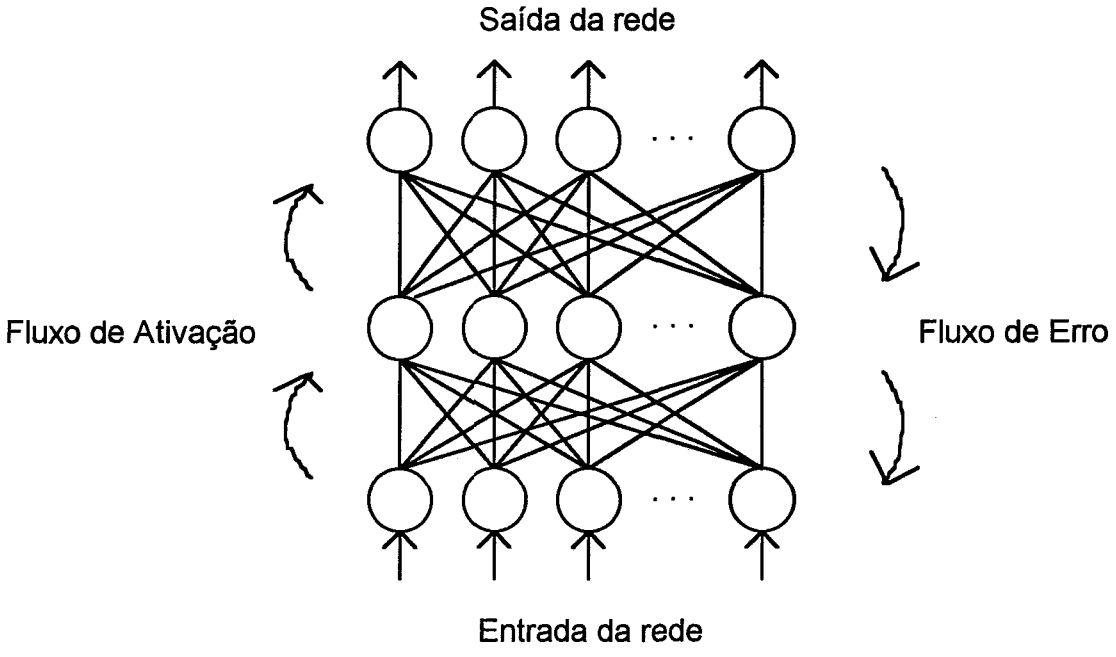


Figura IV.8 : Fluxo de ativação e erro em uma rede "Back-propagation"

f) Calcula-se o ajuste dos pesos das conexões através da equação :

$$\Delta w_{ji} = \eta \, \delta_j \, s_i \qquad \text{(IV.11)}$$

g) Repete-se os passos c, d, e e f até que o erro entre as saídas desejadas e as saídas obtidas convirja para um valor aceitável.

O treinamento da rede pela regra delta generalizada inicia-se com uma matriz de pesos escolhidos aleatoriamente e, muitas vezes, é necessário um grande número de apresentações dos padrões de entrada até que a rede convirja.

Para evitar este tipo de problema, que em alguns casos inviabilizaria a utilização do modelo "Back-propagation", surgiram algumas propostas de modificação do algoritmo com o objetivo de acelerar a convergência. A seguir, serão apresentados alguns métodos utilizados na aceleração do aprendizado :

- **Momento :**

Uma das maiores preocupações do modelo "back-propagation" é a escolha da taxa de aprendizado (ou seja, o η da fórmula IV.11 de ajuste de

pesos). Quanto maior for o seu valor, maior será a variação dos pesos das ligações, tornando mais rápido o aprendizado.

Porém, se esse valor for grande, poderá acarretar uma oscilação quando a função erro se aproximar do ponto de mínimo. Entretanto, se a constante for pequena, ocasionará uma aprendizagem muito lenta.

Assim, para acabar com esse problema (ou seja, oscilações com η grande), foi proposto uma alteração no cálculo do ajuste dos pesos das conexões na regra delta generalizada, com a inclusão de um novo termo denominado momento :

$$\Delta w_{ji}(t+1) = \eta\, \delta_j\, S_i + M\, \Delta w_{ji}(t) \qquad \text{(IV.12)}$$

onde M é o momento que define a influência do ajuste anterior ($\Delta w_{ji}(t)$) no cálculo do ajuste atual dos pesos ($\Delta w_{ji}(t+1)$).

- **Atualização acumulativa dos pesos :**

É outra técnica que também influencia na velocidade de convergência da rede. Sua função é atualizar os pesos das ligações não a cada apresentação de um padrão de entrada e sim, após um conjunto de padrões de entrada.

O tamanho deste conjunto é denominado época ("epoch") e pode compreender todo o conjunto de padrões de entrada ou apenas um subconjunto deste.

Quando a época não é muito grande, este método pode ser muito eficaz, pois ele sempre tende a reduzir o erro global, ao contrário das atualizações individuais que ao tentar reduzir o erro de um padrão particular, podem acarretar o aumento do erro para outros padrões.

Entretanto, se a época for muito grande, o cálculo pode passar a ficar muito trabalhoso e os ajustes podem assumir valores absolutos grandes, provocando oscilações se a taxa de aprendizado não for pequena o suficiente.

Uma outra questão a ser levantada é que o treinamento da rede é um processo de otimização pelo método do gradiente e deste modo, está sujeito aos problemas inerentes a estes processos, ou seja, a hipersuperfície do erro pode conter mínimos locais e assim, o algoritmo "Back-propagation" pode estabilizar-se em um mínimo local ao invés do global.

Para evitar que isto aconteça, duas medidas devem ser tomadas:

- A inicialização e a escolha do intervalo possível dos pesos devem ser realizadas com cuidado pois a inicialização dos pesos com valores muito grandes acaba por saturar os neurônios, levando a rede a se estabilizar em um mínimo local perto do ponto inicial de excitação.
- Mostrar os padrões de entrada aleatoriamente a cada apresentação de todo o conjunto de padrões. Isto impede o cancelamento de ajustes que pode acontecer quando o treinamento é realizado sempre com a mesma seqüência dos padrões de entrada.

Todas estas técnicas foram utilizadas para a implementação do caso exemplo, que será descrito detalhadamente a seguir.

IV.3 - Implementação do Caso Exemplo :

Utilizando o caso exemplo apresentado na seção II.2.2, implementamos um processador inteligente de alarmes empregando o modelo de "Back-propagation", onde cada neurônio de entrada representa um alarme e cada neurônio de saída representa uma falha do sistema.

Deste modo, pela tabela II.1 e II.2 (seção II.2.2), teremos 23 neurônios de entrada correspondentes aos 23 alarmes existentes e 29 neurônios de saída representando as 29 diferentes falhas.

A forma de representação escolhida para os vetores de entrada e saída foi a binária. Assim, quando queremos representar que um alarme está ativo ou não, basta associá-lo aos valores 1 e 0 respectivamente, onde devemos ressaltar que a ordem de caracterização dos alarmes e falhas nos vetores de entrada e saída da rede é a mesma mostrada nas tabelas II.1 e II.2.

Deste modo, se quisermos representar o diagnóstico da falha próxima na linha de transmissão 01 f/f (tabela II.1) devido a atuação dos alarmes de abertura do disjuntor, fase (f/f) e instantâneo da linha de transmissão 01, devemos fornecer à rede os seguintes padrões :

- Padrão de entrada 11001000000000000000 representando os alarmes 1 (abertura do disjuntor da linha de transmissão 01), 2 (fase da linha de transmissão 01) e 5 (instantâneo da linha de transmissão 01) atuados e os demais não.
- Padrão de saída desejado 10000000000000000000000000000 representando apenas a ocorrência da falha próxima na linha de transmissão 01 f/f .

Para ensinar a rede, foi criado um arquivo de treinamento, que continha a descrição de falhas simples e duplas (e seus respectivos alarmes associados) num total de 238 casos (deve-se notar que denominamos de falhas duplas, a conjunção de duas falhas simples). Este arquivo foi, então, apresentado à rede neural como nos mostra a figura IV.9.

O treinamento da rede foi realizado utilizando o software de simulação de rede neural NeuralWorks Professional II/Plus [Neuralware 91] em um micro-computador IBM PC/AT compatível.

A partir do arquivo de treinamento, passamos a testar neste software, quais os parâmetros que poderiam dar uma melhor performance para a rede.

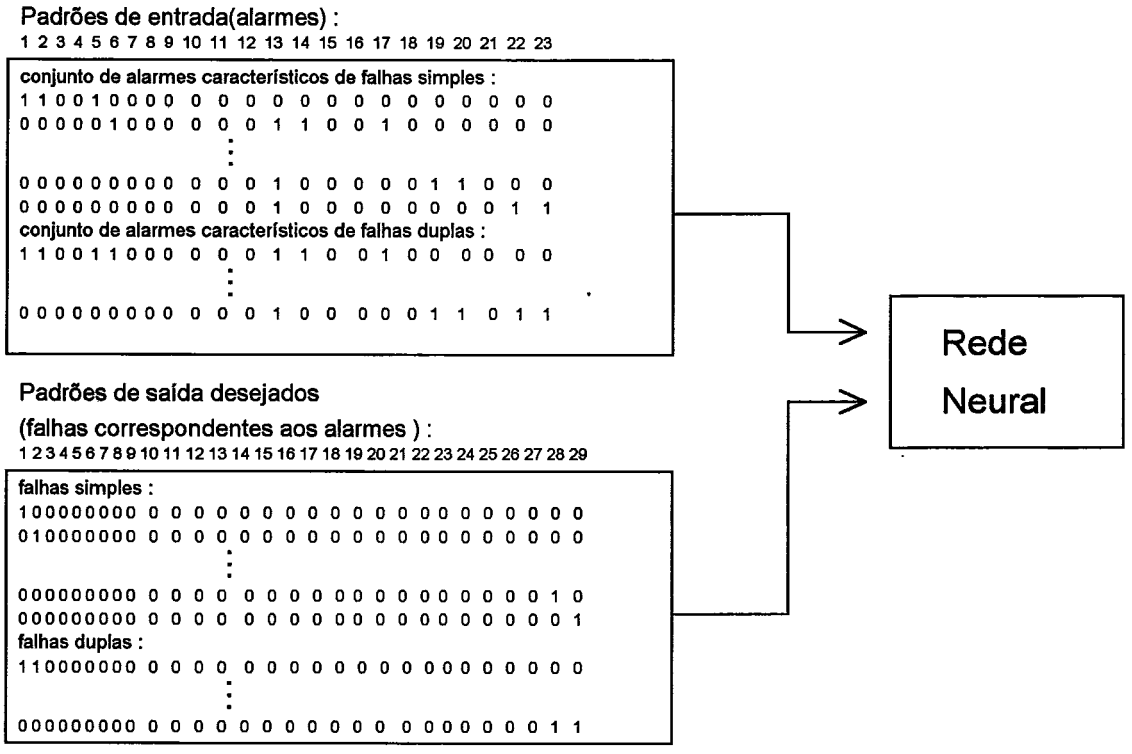


Figura IV.9 : Treinamento da rede neural para o caso exemplo

Primeiramente, procuramos descobrir o melhor dimensionamento da rede. Para isto, utilizamos os parâmetros "defaults" do software NeuralWorks Professional II/Plus (taxa de aprendizado = 0,5; momento = 0,4; época = 16; função de transferência = sigmóide; regra de aprendizado = regra delta normalizada e acumulativa, intervalos dos pesos entre -0,1 a 0,1), variando apenas o número de camadas intermediárias (uma ou duas) e o número de neurônios em cada uma dessas camadas (5, 10, 20 e 30 neurônios).

Os resultados (figuras IV.10 e IV.11) demonstraram que o erro de aprendizado utilizando duas camadas intermediárias era 10% maior que o empregando apenas uma camada e neste grupo, a melhor performace foi obtida pelo treinamento com 5 neurônios na camada intermediária (Erro rms de aprendizado = 0,075).

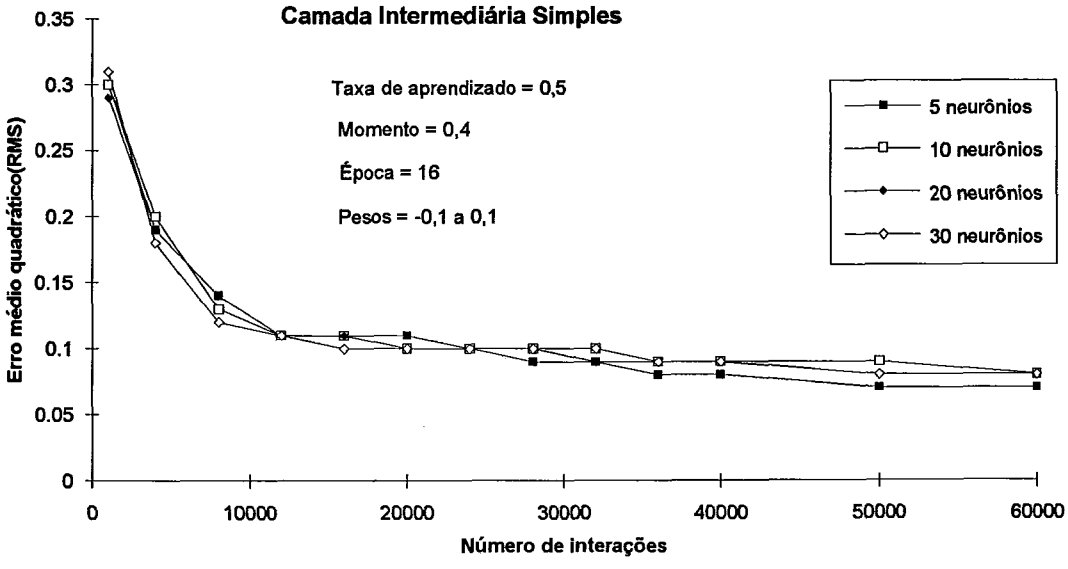


Figura IV.10 : Resultados de aprendizado com uma única camada intermediária

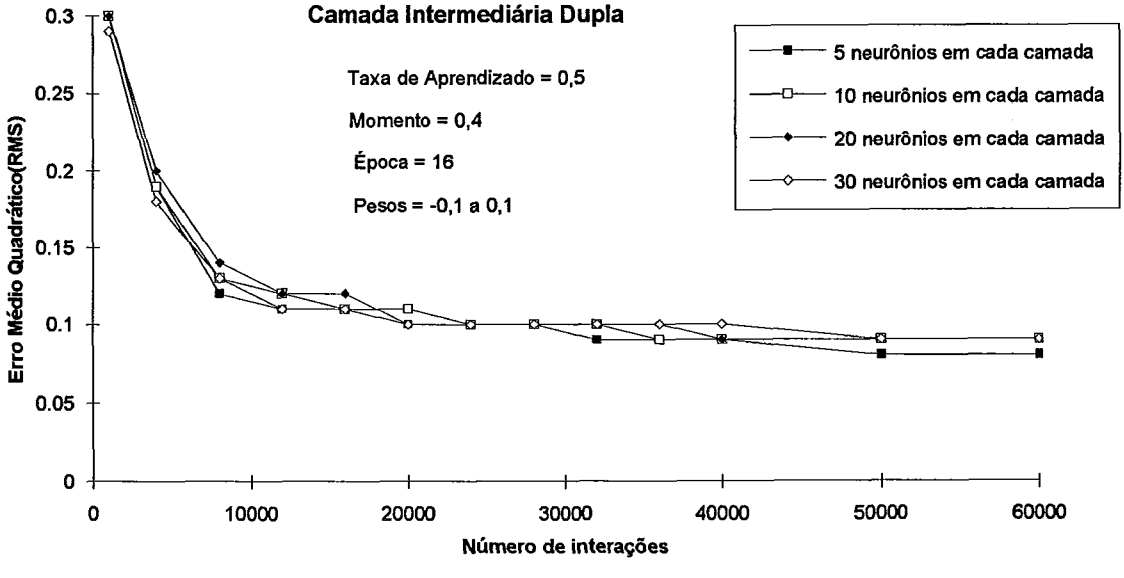


Figura IV.11 : Resultados de aprendizado com dupla camada intermediária

Passamos então, a variar outros parâmetros característicos da rede (taxa de aprendizado, momento e época) com o objetivo de diminuir o erro de aprendizado aprimorando ainda mais o resultado do treinamento.

Inicialmente, foram testadas várias taxas de aprendizado e a melhor performance foi obtida pela taxa de aprendizado igual a 0,90 com o erro rms de aprendizado caindo para 0,060 (figura IV.12).

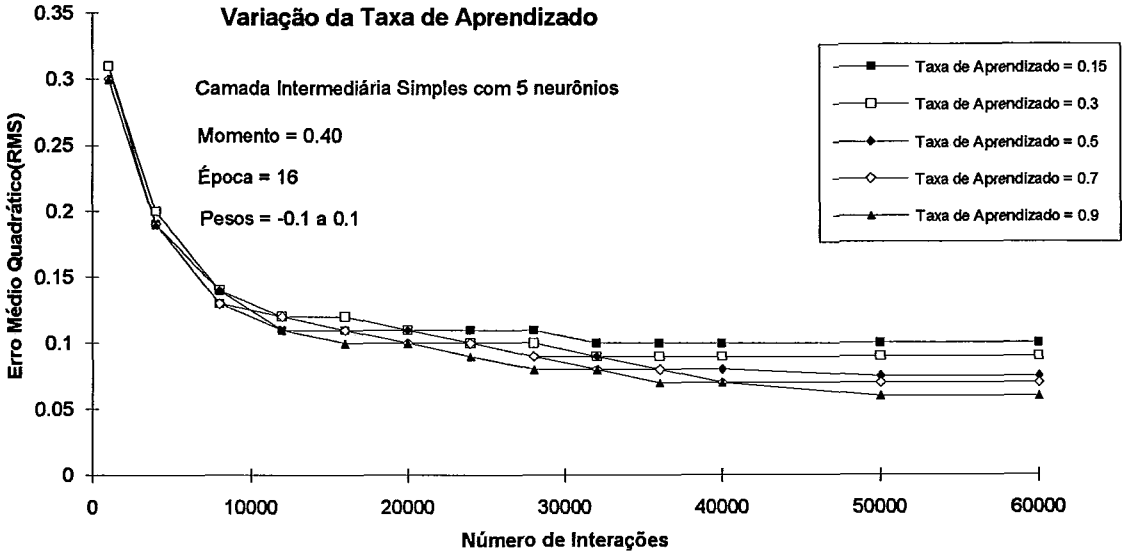


Figura IV.12 : Resultados variando a taxa de aprendizado

Em seguida, variamos o valor do momento e o momento com valor igual a 0,8 obteve o melhor resultado, fazendo cair significativamente o erro rms de aprendizado para 0,040 (figura IV.13).

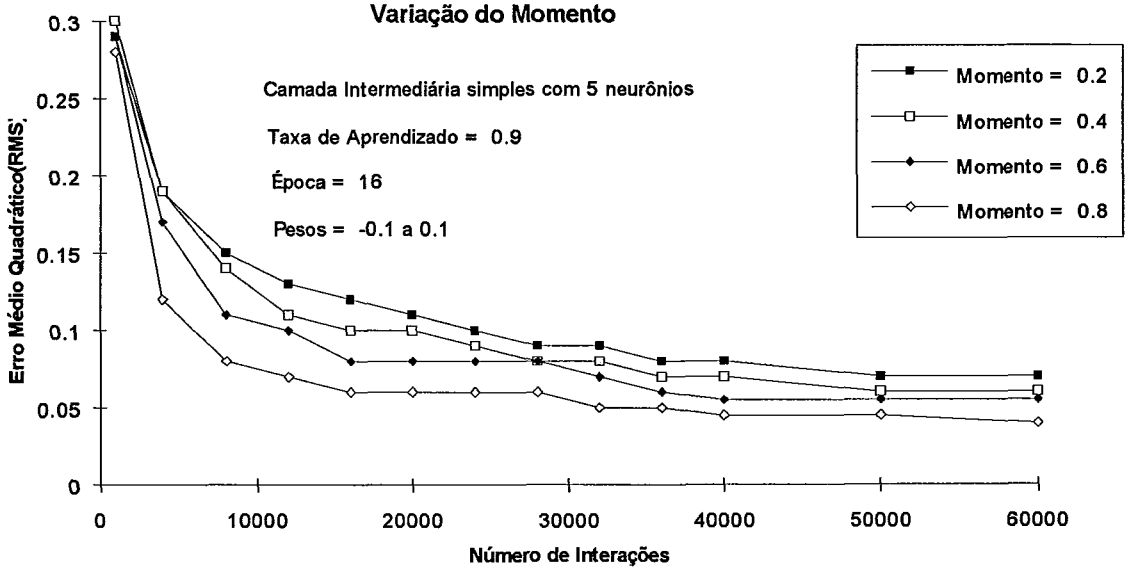


Figura IV.13 : Resultados variando o momento

Finalmente, passamos a alterar o valor da época e o menor erro rms de aprendizado (erro = 0,03) foi obtido pela época igual a 11 (figura IV.14).

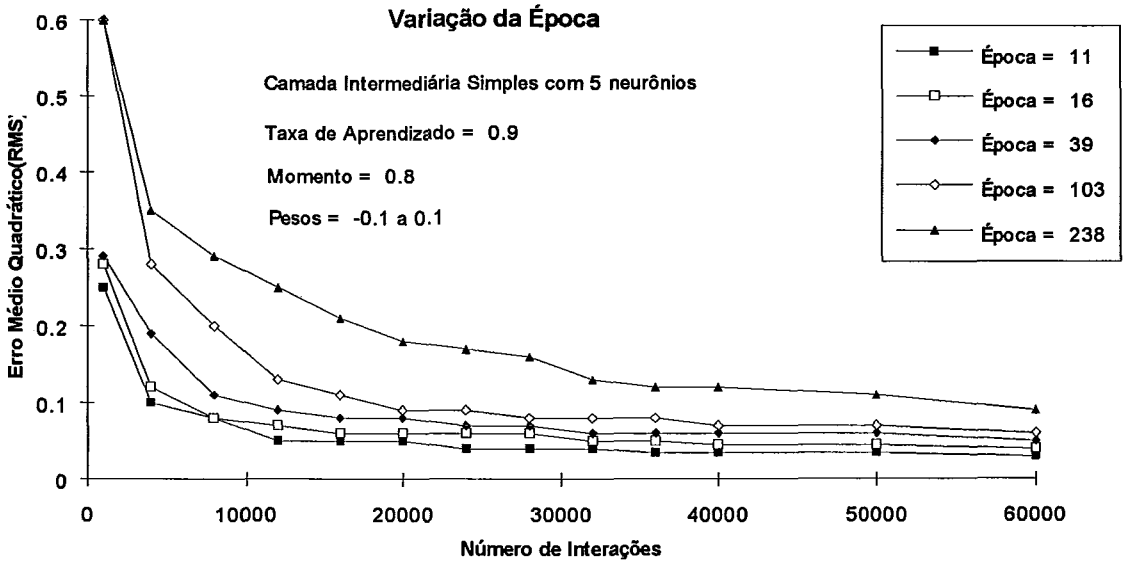


Figura IV.14 : Resultados variando o valor da época

Deste modo, os parâmetros da rede que obtiveram o melhor resultado no treinamento foram :

a) Camada intermediária simples com 5 neurônios;

b) Taxa de aprendizado = 0,9 ;

c) Momento = 0,8 ;

d) Época = 11;

e) Intervalo dos pesos entre -0,1 a 0,1 ;

f) Função de transferência = Sigmóide e

g) Regra de aprendizado = Delta normalizada e acumultativa.

Estes resultados foram obtidos durante o treinamento da rede com 238 padrões diferentes mas para verificar se a rede realmente aprendeu, precisamos testá-la não só com esses 238 padrões, mas também com outros padrões

diferentes denominados de padrões de teste para verificar a capacidade de generalização da rede.

É sabido que a capacidade de generalização de uma rede decai quando o treinamento é prolongado demais, levando a uma super-especialização (ou seja, ela passa a só responder corretamente aos padrões de treinamento, errando quando um padrão diferente destes é apresentado).

É difícil determinar qual a melhor hora de parar o treinamento, por isso, escolhemos duas configurações diferentes da rede que obteve o melhor resultado durante o aprendizado, com o objetivo de submetê-las aos padrões de teste. A primeira configuração (denominada de al20000) treinou 20000 vezes, a segunda (al30000) treinou 30000 vezes.

Os testes foram realizados com quatro tipos de padrões diferentes :

a) ***Treinamento*** → contém todos os padrões de treinamento (238 padrões).

b) ***Falhas múltiplas*** → contém todas as caracterizações de falhas triplas e quádruplas possíveis (580 padrões), onde classificamos de falhas triplas e quádruplas, a conjunção de três e quatro falhas simples diferentes.

c) ***Falhas simples com ruído*** → contém todas as descrições de falhas simples com ruído na entrada (92 padrões). Deve-se notar que denominamos de ruído, a ausência de um dos alarmes que caracterizasse as respectivas falhas.

d) ***Falhas duplas com ruído*** → contém as caracterizações de falhas duplas com ruído na entrada (745 padrões).

Sabemos infelizmente que na maioria das vezes, a rede neural (representação binária) não responde a um estímulo na camada de entrada com valores de saída iguais a 0 ou 1 mas sim com valores intermediários entre estes

dois limites. Assim, com o objetivo de facilitar a depuração, os arquivos gerados durante estes testes foram submetidos a um interpretador.

Este interpretador é bastante simples e pode ser visto mais minuciosamente no apêndice F. Ele apenas lê o arquivo de resultados, mostrando quais as entradas (alarmes) que estão ativas e informando as saídas geradas (falhas) que possuem valores superiores a 0,1 classificando-as do seguinte modo :

a) ***Falhas Previstas*** → são aquelas que se tem certeza que ocorreram, possuindo valores mais elevados, ou seja, entre 1,0 e 0,70.
b) ***Falhas Prováveis*** → são aquelas que não se tem uma convicção total, possuindo valores intermediários entre 0,40 e 0,70.
c) ***Indícios de falha*** → são aquelas com pouca certeza possuindo valores entre 0,40 e 0,10.

Vamos agora, descrever o comportamento de cada uma das duas configurações da rede (al20000 e al30000), quando submetidas aos diferentes padrões de teste :

a) ***Padrão de treinamento*** → Não foi encontrado erro em nenhuma configuração.
b) ***Padrão com falhas múltiplas*** → Também não foi encontrado erro em nenhuma das configurações.
c) ***Padrão com falhas simples com ruído*** → Foram encontrados dois erros na configuração al30000.
d) ***Padrão com falhas duplas com ruído*** → Foram encontrados 49 erros na configuração al20000 e 68 erros na configuração al30000.

Deste modo, podemos concluir que para os 1651 padrões de teste apresentados, a configuração al20000 apresentou uma taxa de erro de 2,96% e a configuração al30000 mostrou um taxa de erro um pouco maior (4,23%).

Estes resultados são bastante promissores, mostrando que a rede é capaz de diagnosticar falhas mesmo quando submetida a novos padrões (padrões diferentes dos padrões de treinamento) com uma performance superior a 97% de acerto.

Em relação à velocidade de processamento da rede, foram efetuados os mesmos testes realizados no sistema utilizando lógica não-monotônica (seção III.5) e o desempenho da rede (utilizando apenas um simulador de rede neural e não, uma versão executável) em um computador IBM PC-386/SX 25 Mhz foram bastante satisfatórios como nos mostra a tabela IV.1 . Devemos notar que o tempo de resposta é sempre o mesmo, independente do número de entradas (alarmes) ativas.

A partir de todos os resultados obtidos e descritos acima, podemos concluir que ***a rede neural, representada pelo modelo "Back-propagation", é uma técnica adequada e eficaz para a diagnose em sistemas de potência.***

Dados de Entrada	Saídas Obtidas (Explicações)	Desempenho da Rede Neural
alarme(fase,linha_transmissão_ 01) alarme(instantâneo,linha_transmissão_01)	- Falha próxima na linha de transmissão 01 f/f	t = 2,3 seg
alarme(abriu,linha_transmissão_01) alarme(instantâneo,linha_transmissão_01) alarme(terra,linha_transmissão_01) alarme(abriu,transformador_01) alarme(sobre,transformador_01)	- Falha próxima na linha de transmissão 01 f/t - Falha no transformador 01	t = 2,3 seg
alarme(abriu,linha_transmissão_02) alarme(temporizado,linha_transmissão_02) alarme(fase,linha_transmissão_02) alarme(abriu,by-pass) alarme(aux,transformador_01) alarme(sobre,transformador_01)	- Falha distante na linha de transmissão 02 f/f - Falha na barra auxiliar (3) - Falha no transformador 01 (3)	t = 2,3 seg
alarme(abriu,linha_transmissão_01) alarme(fase,linha_transmissão_01) alarme(instantâneo,linha_transmissão_01) alarme(terra,linha_transmissão_ 02) alarme(temporizado,linha_transmissão_02) alarme(sobre,transformador_01) alarme(abriu,by-pass) alarme(aux,transformador_02) alarme(sobre,transformador_02)	- Falha próxima na linha de transmissão 01 f/f - Falha distante na linha de transmissão 02 f/t - Falha no transformador 01 - Falha na barra auxiliar (4) - Falha no transformador 02 (4)	t = 2,3 seg

Tabela IV.1 : Teste de desempenho da Rede Neural

Capítulo V

Conclusões

V.1 - Comparação entre as duas técnicas :

Podemos concluir, em função das observações e resultados obtidos, que as duas técnicas de Inteligência Artificial :

- ***Lógica Não-monotônica*** → Representada pela :

  Diagnose Abdutiva e o

  Sistema Theorist

- ***Redes neurais*** → Representada pelo modelo ***"Back-propagation"***

são perfeitamente aplicáveis a Diagnose de Sistemas de Potência.

Entretanto, a representação do conhecimento destas duas técnicas é totalmente distinta. Deste modo, faremos a seguir, uma avaliação comparativa entre elas, apresentando as vantagens e desvantagens de cada método :

1) A representação do conhecimento na Lógica Não-monotônica é feita através de declarações que descrevem o comportamento do sistema que queremos representar (ou seja, representação declarativa) [Genesereth *et. al.* 87].

   Entretanto, muitas vezes a aquisição deste conhecimento é complicada (seção II.3.1) mas quando este obstáculo é superado, notamos que a sua representação em Lógica é feita de forma bastante simples e intuitiva.

   Esta simplicidade pode ser comprovada pela representação na diagnose abdutiva do nosso sistema elétrico (seção III.5) e dos exemplo das baterias e dos inversores (seção III.3).

Já as Redes Neurais possuem a capacidade de aprender (isto é, adquirir conhecimento) através de exemplos, ou seja, o funcionamento do sistema que queremos representar, é modelado informalmente através da apresentação exaustiva de casos típicos, como pode ser verificada na implementação do nosso sistema elétrico (seção IV.3).

Entretanto, esta capacidade de aprendizado da Rede Neural depende de vários parâmetros característicos (tais como, a forma de representação dos vetores de entrada e saída da rede, da taxa de aprendizado, do número de camadas intermediárias, do número de neurônios em cada camada, entre outros) e, infelizmente, ainda não existe uma formulação que indique como devemos usar cada um destes parâmetros.

Assim, estes são escolhidos de forma empírica, o que diminui consideravelmente esta facilidade das Redes Neurais de aprenderem informalmente através de exemplos.

2) A Lógica Não-monotônica normalmente pode mostrar os motivos que a levaram a obter uma determinada resposta. Esta característica só é possível porque a representação do conhecimento é feita de forma explícita através de estruturas de dados apropriadas [Genesereth *et. al.* 87] (no caso da diagnose abdutiva, estas estruturas seriam representadas pelos fatos, possíveis hipóteses e "constraints").

Esta capacidade de explicar as suas conclusões é uma caracterítica extremamente desejável no diagnóstico de falhas em um sistema de potência pois ao mostrar o seu mecanismo de raciocínio, estamos fornecendo mais ferramentas para o operador entender o que está ocorrendo no sistema elétrico.

Já a Rede Neural não é capaz de explicar porque obteve uma determinada resposta pois a representação do seu conhecimento está

distribuida através da interação entre os seus neurônios, realizada em função dos seus valores de ativação e da força das conexões existentes entre eles (seção IV.2.2).

3) A capacidade de generalização de uma Rede Neural faz com que ela seja capaz de responder corretamente em situações similares, mas não idênticas, àquelas utilizadas no seu aprendizado.

Podemos ilustrar esta capacidade através do diagnóstico de uma das falhas (falha próxima na linha de transmissão 01 f/f - tabela II.1) do nosso caso exemplo.

Durante o aprendizado, a única informação que fornecemos a Rede Neural sobre esta falha, é que ela é caracterizada pela atuação ***conjunta*** de três alarmes :

Alarme de Abertura do Disjuntor da Linha de Transmissão 01

Alarme de Fase (F/F) na Linha de Transmissão 01

Alarme Instantâneo na Linha de Transmissão 01

Porém, se após o aprendizado quisermos saber da Rede Neural, qual o diagnóstico possível para a atuação dos alarmes de fase (f/f) e instantâneo na linha de transmissão 01, ela responderá que a falha próxima na linha de transmissão 01 f/f ocorreu mesmo sem este caso ter sido representado durante o treinamento da rede e estar faltando o alarme de abertura do disjuntor na linha de transmissão 01. A Rede Neural diagnosticou esta falha por ser a que melhor justifica a atuação conjunta destes dois alarmes em relação a outras falhas existentes (Tabela II.1 e II.2)

Esta característica da Rede Neural é bastante interessante em um sistema de potência real, entretanto, devemos ressaltar que conforme a situação apresentada a rede neural pode responder erradamente. Para tal, devemos certificar esta capacidade de generalização (ou seja, verificar como a rede se comporta em situações diferentes às

apresentadas durante o aprendizado) através dos padrões de teste conforme mostrado na seção IV.3 .

Infelizmente, a Lógica Não-monotônica só é capaz de responder ao que estiver descrito na sua base de conhecimento.

Deste modo, para Lógica Não-monotônica ser capaz de diagnosticar a falha próxima na linha de transmissão 01 f/f pela atuação conjunta dos três alarmes citados acima, ou apenas pela atuação dos alarmes de fase e instantâneo na linha de transmissão 01, devemos descrever estas duas caracterizações na sua base de conhecimento.

4) A modificação e acréscimo de novos dados geralmente é realizada com facilidade pela Lógica Não-monotônica devido a representação do seu conhecimento ser feita de forma declarativa [Genesereth *et. al.* 87].

   Entretanto, devemos ressaltar que a representação de sistemas mais complexos (como é o caso de um sistema elétrico real) pode gerar uma base de conhecimento muito grande, tornando difícil a sua manutenção.

   Em relação a Rede Neural, este acréscimo de novos dados é um fator um pouco mais delicado pois se tentarmos retreinar uma rede para acrescentar um novo dado, isto pode acarretar na perda de alguma informação anteriormente aprendida.

   Deste modo, o mais seguro é recomeçar o treinamento do zero, introduzindo este novo dado entre os caso típicos utilizados anteriormente na representação do sistema.

5) Em relação aos tempos de resposta obtidos nos testes com as duas técnicas de Inteligência Artificial (tabelas III.1 e IV.1), podemos concluir que a Lógica Não-monotônica é capaz de explicar mais rapidamente a atuação de um conjunto de alarmes com até 6 elementos.

Entretanto, notamos que o seu tempo de resposta aumenta em função do número de alarmes apresentados, enquanto que o tempo da Rede Neural mantêm-se sempre estável em 2,3 segundos.

Esta característica da Rede Neural é extremamente desejável em um sistema elétrico real onde a quantidade de alarmes, que se deseja explicar, normalmente é elevada.

Devemos ressaltar que estes tempos da Rede Neural foram obtidos em um simulador e podem ser melhorados utilizando uma versão executável ou uma arquitetura de hardware específica.

6) Em relação aos resultados obtidos pelas duas técnicas (seção III.5 e IV.3) nos testes de diagnóstico, podemos verificar que a Lógica Não-monotônica identificou corretamente todas as falhas simuladas que lhe foram apresentadas (291 casos diferentes com 121 representando falhas sem ruído, ou seja, sem a ausência de algum alarme característico, e 170 descrevendo falhas com ruído).

Já a taxa de acerto da Rede Neural ao diagnosticar falhas, foi de 100% quando não ocorre ausência de alarmes (818 casos diferentes) e decaiu para 97% quando o conjunto de alarmes estava incompleto (837 padrões diferentes).

Estes resultados são extremamente satisfatórios e nos encorajam a dar prosseguimento a este trabalho.

Entretanto, devemos notar que os testes realizados com a Rede Neural, foram muito mais rigorosos que os da Lógica Não-monotônica. Este fato foi conseqüência das limitações impostas pelo compilador Prolog, utilizado na implementação do Theorist, que não permitiu uma automação completa dos seus testes de diagnóstico.

V.2 - Conclusões e Trabalhos Futuros :

Esta comparação nos permite concluir que cada uma das técnicas de Inteligência Artificial estudadas apresenta características desejáveis para um Sistema de Potência que embora distintas são complementares.

Assim, a integração da Lógica Não-monotônica com a Rede Neural traria grandes benefícios pois poderíamos aproveitar ao máximo a potencialidade de cada uma destas técnicas e evitar as suas deficiências.

Este ***Sistema Híbrido*** utilizaria :

- A ***Rede Neural*** para informar quais as falhas que estão ocorrendo em um sistema de potência,
- A ***Lógica Não-monotônica*** para interpretar estes resultados, ou seja, explicar os motivos que levaram ao diagnóstico de cada uma destas falhas.

Podemos destacar como contribuições desta tese :

- O estudo dos dois tipos de diagnose utilizando lógica mais abordados na literatura : *Diagnose abdutiva e Diagnose baseada na consistência,*
- A implementação de um *interpretador e de um compilador* em Prolog para o *Sistema Theorist,*
- A implementação da *Diagnose Abdutiva no Sistema Theorist,*
- O estudo do modelo mais utilizado para aplicações em computação neural : *modelo "Back-propagation",*
- A implementação das duas técnicas de Inteligência Artificial *(Lógica Não-monotônica e Rede Neural)* em um Sistema Elétrico Simplificado de uma Usina,
- A avaliação e desenvolvimento de um *Sistema Híbrido* utilizando estas duas técnicas de Inteligência Artificial (Inclusive, devemos ressaltar que já desenvolvemos o módulo do Sistema Híbrido que utiliza Lógica

Não-monotônica - representado novamente pela Diagnose Abdutiva e o Theorist - faltando apenas a sua integração com a Rede Neural)

Estes estudos e resultados apresentados servem de base para a implementação da ***Diagnose em um Sistema de Potência Real utilizando técnicas de Inteligência Artificial.***

Finalizando, podemos afirmar que as dificuldades e desafios que surgiram durante este trabalho foram recompensadas com o progressivo domínio destas técnicas de Inteligência Artificial. Esperamos agora, que a sua continuação seja premiada com a implementação em um sistema de potência real.

Referências Bibliográficas

[Buchanan *et. al.* 84] B. G. Buchanan, E. H. Shortlife, "Ruled-based Expert Systems: The MYCIN Experiments of the Stanford Heurist Programming Project", Addison-Wesley, 1984.

[Cadoli *et. al.* 93] M. Cadoli, M. Schaerf, "Complexity Results for Non-monotonic Logics", Journal of Logic Programming, Vol. 17, No. 2,3,4, 1993, pp. 127-160.

[Cardozo *et. al.* 87] E. Cardozo, S. N. Talukdar, "A Distributed Expert System for Fault Diagnosis", Proceedings of the IEEE Power Industry Computer Application Conference, Montreal, Canada, May 1987.

[Carvalho 88] L. A. V. de Carvalho , "Redes Neuronais e a Tradição Conexionista da Inteligência Artificial", Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1988.

[Carvalho 89] L. A. V. de Carvalho, "Síntese de Redes Neuronais com Aplicação à Representação do Conhecimento e a Otimização", Tese de Doutorado, COPPE- Universidade Federal do Rio de Janeiro, março de 1989.

[Carvalho *et. al.* 91] L. A. V. de Carvalho, F. M. G. França, L. M. R. Eizirik, S. B. T. Mendes, V. C. Barbosa, "Redes Neuronais Artificiais : a volta do cerébro eletrônico", Ciência Hoje, Vol..12, No.12, Janeiro 1991, pp. 12-21.

[Casanova *et. al.* 86] M. A. Casanova, F. A. C. Giorno, A. L. Furtado, "Programação em Lógica", V Escola de Computação, Belo Horizonte, 1986.

[Chan 89] E. H. P. Chan , "Application of Neural-Network Computing in Intelligent Alarm Processing", Power Industry Computer Application Conference, Seattle, Washington, May 1989, pp. 246-251.

[Chan 90] E. H. P. Chan , "Using Neural Network to Interpret Multiple Alarms", IEEE Computer Applications in Power, Vol. 3, No. 2, April 1990, pp. 33-37.

[Chan *et. al.* 89] E. H. P. Chan , N. S. Markushevich, R. Adapa, "Interpretive Power System Alarm Processing Test Results", Proceedings of First International Forum on Applications of Neural Networks to Power Systems, Seattle, Washington, May 1991.

[CIGRE 92] CIGRE, "A Progress Report on Pratical Use of Expert Systems in Plannnig and Operation", CIGRE - 1992 Session, August 1992, Ref. Task Force 38.03.10.

[CIGRE 93] CIGRE, "Artificial Neural Networks for Power Systems - A Literature Survey", CIGRE - 1993 Session, July 1993, Ref. Task Force 38.06.06.

[Cox *et. al.* 87] P. T. Cox, T. Pietrzykowski, "General Diagnosis by Abdutive Inference", Techinical Report CS8701, School of Computer Science, Techinical University of Nova Scotia, April 1987.

[de Kleer 76] J. de Kleer, "Local Methods for localizing faults in eletronics circuits", MIT AI Memo 394, Cambridge, 1976.

[de Kleer *et. al.* 87] J. de Kleer, e B. C. Williams, "Diagnosing Multiple Faults", Artificial Intelligence, Vol. 32, pp. 97-130.

[de Kleer *et. al.* 90] J. de Kleer, A. Mackworth e R. Reiter, "Characterizing Diagnoses" , Proceedings AAAI-90, Boston, pp. 324-330.

[de Kleer *et. al.* 92] J. de Kleer, A. K. Mackworth e R. Reiter, "Characterizing Diagnoses and Systems", Artificial Intelligence, Vol. 56, No. 2-3, pp. 197-222.

[Dix 92] J. Dix, "Default Theories of Poole-Type and a Method for Constructing Cumulative Versions of Default Logic" , Proceedings ECAI-92, Vienna, Austria.

[Dy Liacco 67] T. E. Dy Liacco, "The Adaptative Reliability Control System" IEEE Transactions on Power Apparatus and Systems, Vol. PAS-86, No. 5, May 1967, pp. 517-531.

[Dy Liacco 78] T. E. Dy Liacco, "System Security : The Computer's Role" IEEE Spectrum, Vol. 15, June 1978, pp. 43-50.

[Enderton 72] H. B. Enderton, "A Mathematical Introduction to Logic", Academic Press, Orlando Florida, 1972.

[Etherington 87a] D. W. Etherington, "Formalizing Nonmonotonic Reasoning", Artificial Intelligence, Vol. 31, pp. 41-85, 1987.

[Etherington 88] D. W. Etherington, "Reasoning with Incomplete Information", Reserch Notes in Artificial Intelligence, Pitman/Morgan Kaufmann, London, 1988.

[Fukui *et. al.* 86] C. Fukui, J. Kawakami, "An Expert System for Fault Section Estimation using Information from Protective Relays and Circuit Breakers", IEEE Transactions on Power Delivery, Vol. 1, No. 4, October 1986.

[Genesereth 84] M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis", Artificial Intelligence, Vol. 32, 1984, pp.411-436.

[Genesereth *et. al.* 87] M. R. Genesereth, N. J. Nilsson, "Logical Foundations of Artificial Intelligence", Morgan Kauffmann Publishers Inc. , 1987

[Hecht-Nielsen 90] R. Hecht-Nielsen, "Neurocomputing", Addison-Wesley Publishing Company, 1990.

[Hein *et. al.* 88] F. Hein, G. Schellstede, "Use of Expert Systems in Energy Control Centres", CIGRE - International Conference on Large High Voltage Eletric Systems, September 1988, Ref. 39-15.

[Jones *et. al.* 85] M. Jones e D. L. Poole, "An Expert System for Educational Diagnosis based on Default Logic", Proceedings Fifth International WorkShop on Expert Systems and their Aplications, Avignon, France, pp. 673-683.

[Kawada *et. al.* 89] H. Kawada, T. Hasegawa, K. Ose, T. Fukuda, Y. Mukaiyama, S. Moirguchi, H. Noguchi,, N. Ito, "Present Situation and Future Plans of Operation and Maintenance Supporting System at Substation", CIGRE - Symposium Bournemouth 1989, Ref. 1A-06.

[Kim *et. al.* 91] K. Kim, J. Park, "Application of Hierarchical Neural Network to Fault Diagnosis of Power Systems", Third Symposium on Expert Systems Applications to Power Systems, Tokyo, April 1991, pp. 323-327.

[Kirschen *et. al.* 89] D. S. Kirschen, B. F. Wollenberg, G. D. Irisarri, J. J. Bann, B. N. Miller, "Controlling Power Systems during Emergencies : The Role of Expert Systems", IEEE Computer Applications in Power, Vol. 2, No. 2, April 1989, pp. 41-45.

[Konolige 92] K. Konolige, "Abduction versus Closure in Causal Theories", Artificial Intelligence,Vol. 53, pp. 255-272.

[Kowalski 74] R. Kowalski, "Predicate Logic as a Programming Language", Proceedings IFIP 74 World Congress, North- Holland Publishing Company, Amsterdam.

[Lambert-Torres *et. al.* 91] G. Lambert-Torres, B. Valiquette, D. Mukhldkar, "An Expert System based Diagnosis and Advisor Tool for Teaching Power Systems Operation Emergency Control Strategies", IEEE Transactions on Power Systems, Vol. 6, No. 3, August 1991, pp. 1315-1322.

[Loveland 78] D. W. Loveland, "Automated Theorem Proving : A Logical Basis", North-Holland, Amsterdam, 1978.

[Lukaszewicz 90] W. Lukaszewicz, "Non-monotonic Reasoning - Formalization of Commonsense Reasoning", Ellis Horwood Limited, England, 1990.

[NeuralWare 91] NeuralWare Inc., "NeuralWorks Professional II/Plus and Neural Works Explorer - Reference Guide", 1991.

[Newell *et. al.* 91] W. R. Becraft, P. L. Lee, R. B. Newell, "Integration of Neural Networks and Expert Systems for Process Fault Diagnosis", Proceedings IJCAI 91, pp. 832-837.

[Nilsson 80], N. Nilsson, "Principals of Artificial Intelligence", Springer Verlag, Berlin, 1980.

[Poole *et. al.* 87] D. L. Poole, R. G. Goebel e R. Aleliunas, "Theorist: a Logical Reasoning System for Defaults and Diagnosis", The Knowledge Frontier: Essays in the Representation of Knowledge, Springer Varlag, New York, pp. 331-352.

[Poole 87] D. L. Poole, "Defaults and Conjectures : Hypothetical Reasoning for Explanation and Prediction", Research Report CS-87-54, Department of Computer Science, University of Waterloo, October 1987.

[Poole 88a] D. L. Poole, "Representing Knowledge for Logic-Based Diagnosis", Proceedings International Conference on Fifth Generation Computing Systems, Tokyo, pp. 1282-1290.

[Poole 88b] D. L. Poole, "A Logical Framework for Default Reasoning", Artificial Intelligence, Vol. 36, No.1, pp. 27-47.

[Poole 89] D. L. Poole, "Normality and Faults in Logic-Based Diagnosis" , Proceedings IJCAI 89, pp. 1304-1310.

[Poole 91] D. L. Poole, "Compiling a Default Reasoning System into Prolog", New Generation Computing, pp. 3-38.

[Popper 58] K. R. Popper, "The Logic of Scientific Discovery", Basic Books, New York, 1958.

[Popper 62] K.. R. Popper, "Conjectures and Refutations : The Growth of Scientific Knowledge", Basic Books, New York, 1962.

[Quine *et. al.* 78] W.. V. Quine, J. S. Ullian, "The WEB of Belief", Random House, New York, Second Edition, 1978.

[Reggia *et. al.* 83] J. A. Reggia, D. S. Nau and P. Y. Wang, "Diagnostic Expert Systems based on a Set Covering Model", International Joint Man-Machine Studies 19, pp. 437-460.

[Reiter 80] R. Reiter, "A Logic for Default Reasoning", Artificial Intelligence, Vol. 13, No. 1, pp. 81-132, 1980.

[Reiter 87] R. Reiter, "A Theory of Diagnosis from First Principles", Artificial Intelligence, Vol. 32, No. 1, pp. 57-96.

[Rodriguez *et al.* 92] C. Rodriguez, S. Rementeria, C. Ruiz, A. Lafuente, J. I. Martin, J. Murgueza, "A Modular Approach to the Design of Neural Networks for Fault Diagnosis in Power Systems", International Joint Conference on Neural Networks, Baltimore, June 1992, Vol.III, pp. 16-23.

[Rumelhart *et. al.* 86] D. Rumelhart, J. McClelland and the PDP Research Group, "Parallel Distributed Processing : exploration in the microstruture of cognition", MIT Press, 1986.

[Sakaguchi *et. al.* 87] T. Sakaguchi, H. Tanaka, K. Uenishi, T. Gotoh, Y. Sekine, "Prospects of Expert Systems in Power System Operation", 9th Power Systems Computation Conference, Cascais, Portugal, September 1987.

[Sattar *et. al.* 91] A. Sattar, R. Goebel, "Using Crucial Literals to Select Better Theories", Computer Intelligence, Vol. 7, pp. 11 -22.

[Souza *et. al.* 93] G. N. F. de Souza, N. de M. Roehl, M. Moszkowicz, "Experiência de Aplicação de Redes Neuronais ao Setor Elétrico", V ERLAC - Encontro Regional Latino-Americano da CIGRE, junho de1993.

[Struss *et. al.* 89], P. Struss, O. Dresller, "Physical Negation - Integrating Fault Models into the General Diagnostic Engine", Proceedings IJCAI-89, Detroit, Michigan, 1989, pp. 1318-1323.

[Swarup *et. al.* 91], K. S. Swarup, H. S. Chandrasekharaiah, "Fault Detection and Diagnosis of Power Systems using Artificial Neural Networks", Third Symposium on Expert Systems Applications to Power Systems", Tokyo/Kobe, April 1991, pp. 102-106.

[Tamura 89] Y. Tamura, "An International Survey of the Present Status and the Perspective of Expert Systems on Power System Analysis and Technique", CIGRE - Symposium Bournemouth 1989, Ref. 1B-08.

[Tanaka *et. al.* 89] H. Tanaka, S. Matsuda, H. Ogi, Y. Izui, H. Taoka, T. Sakaguchi, "Design and Evaluation of Neural Network for Fault Diagnosis", Second Symposium on Expert System Applications to Power Systems, Seattle, Washington, July 1989, pp. 378-384.

[Wollenberg 86] B. F. Wollenberg, "Feasibility for an Energy Management System - Intelligent Alarm Processor", IEEE Transactions on Power Systems, Vol. 1, No. 2, May 1986, pp. 241-247.

[Wollenberg *et. al.* 89] B. F. Wollenberg, W. R. Prince, D. B. Bertagnolli, "Survey on Excessive Alarms ", IEEE Transactions on Power Systems, Vol. 4, No. 3, August 1989, pp. 950-956.

[Wollenberg *et. al.* 91] B. F. Wollenberg, R. W. Bijoch, S. H. Harris, T. L. Volkmann, J. J. Bann, "Development and Implementation of the NSP Intelligent Alarm Processor", IEEE Transactions on Power Systems, Vol. 6, No. 2, May 1991, pp. 806-812.

[Zhang *et al.* 89] Z. Z. Zhang, G. S. Hope, O. P. Mailk, "Expert Systems in Eletric Power Systems - A Bibliographical Survey", IEEE Transactions on Power Systems, Vol. 4, No. 4, October 1989, pp. 1355-1361.

Apêndice A - Versão Interpretada do Theorist

Neste apêndice, é apresentada a listagem da versão interpretada do Theorist em Prolog, no qual começamos a representar os primeiros exemplos utilizando diagnose abdutiva.

```
/*
    Módulo Theorist :
    Neste módulo, estão declaradas todas as rotinas necessárias para utilizarmos o Theorist em uma
    versão interpretada.
*/

:- module theorist.

:-visible explain / 2.
:-visible prall / 4.
:-visible pr / 4.
:-visible prusedef / 6.
:-visible neg / 2.
:-visible member / 2.

/*
    explain(G,D) :
    Procura  provar G utilizando as possíveis hipóteses existentes no conjunto D.
*/
explain(G,D) :-
 prall(G,[],[],D).

/*
    prall(B,A,D,D) :
    Tenta provar que todo o elemento de B pode ser explicado a partir dos antecessores A, começando
    da teoria D1 e resultando na teoria D2.
*/
prall([],A,D,D).

prall([G|B],A,D1,D3) :-
 pr(G,A,D1,D2),
 prall(B,A,D2,D3).


/*
    pr(G,A,D1,D2) :
    Ela é responsável por provar G, usando os elementos de F (conjunto de Fatos) e H (conjunto de
    Hipóteses Possíveis), ou seja, o predicado pr implementa a primeira etapa da construção das
    explicações consistentes para uma observação.
```

```
    Para tal, tenta provar que G pode ser explicado a partir dos antecessores A, começando da teoria
    D1 e resultando na teoria D2.
*/

/*
    Procura uma contradição nos ancestrais da árvore de prova.
*/
pr(G,A,D,D) :-
 neg(G,GN),
 member(GN,A).

/*
    Procura provar os fatos nos quais G aparece na cabeça de alguma regra.
*/
pr(G,A,D1,D2) :-
 fact(G,Body),
 prall(Body,[G|A],D1,D2).

/*
    Tenta provar se G pode ser obtido pela aplicação de alguma possível hipótese existente em H.
*/
pr(G,A,D1,D2) :-
 default(N,G,B),
 prusedef(G,A,N,B,D1,D2).

/*
    Permite utilizar na prova, os predicados do próprio Prolog ("bult-in predicates").
*/
pr(G,A,D,D) :-
 meta(G),
 call(G).



/*
    prusedef(G,A,N,B,D1,D2) :
    Ela é responsável por provar a consistência do default G, ou seja, o predicado prusedef
    implementa a segunda etapa da construção das explicações consistentes para uma observação.
    Tenta provar que G pode ser explicado usando o default D com o corpo B e com o default inicial
    D1, resultando no default D2 e os antecessores A.
*/

/*
    Verifica se o default já foi utilizado anteriormente.
*/
prusedef(G,A,N,B,D1,D2) :-
 member(N,D1),
 prall(B,[G|A],D1,D2).

/*
    Procura provar a negação da possível hipótese G.
*/
prusedef(G,A,N,B,D1,D2) :-
 not(member(N,D1)),
 prall(B,[G|A],[N|D1],D2),
 neg(G,GN),
 not(pr(GN,[],D1,D1)).
```

```
/*
     neg(A,X) :
     Retorna em X, o literal correspondente a negação do literal A.
*/

neg(n(A),A) :-
 A \= n(_).

neg(A, n(A)) :-
 A \= n(_).

/*
     member(A, Lista) :
     Verifica se o literal A pertence a Lista.
*/
member(A,[A|_]).

member(A,[X|Resto]) :-
 member(A,Resto).


/*
     declara_fact(Cabeca,Corpo) :
     É responsável por declarar um fato e todas as suas contrapositivas.
*/
declara_fact(Cabeca,Corpo) :-
 !,
 insere_contrapositivas(Corpo,Cabeca,[]).


/*
     insere_contrapositivas(Corpo,Cabeca,[]) :
     Insere as contrapositivas de qualquer fato declarado no Theorist.
*/
insere_contrapositivas([],Cabeca,Corpo):-
  asserta(fact(Cabeca,Corpo)).


insere_contrapositivas([A|Resto],Cabeca,Corpo):-
  append([A|Resto],Corpo,NovoCorpo),
  asserta(fact(Cabeca,NovoCorpo)),
  neg(Cabeca,Ncabeca),
  neg(A,Na),
  append(Corpo,[Ncabeca],L),
  insere_contrapositivas(Resto,Na,L).
```

Apêndice B - Versão Compilada do Theorist

Neste apêndice, apresentamos a versão compilada do Theorist. Devemos ressaltar que nesta versão, o predicado nnf permite que os fatos, defaults e "constraints" utilizem fórmulas bem formadas e não somente cláusulas conforme mostrado na seção III.2.2. Este programa foi desenvolvido em Prolog e apresenta seguinte estrutura :

```
/*
    Módulo Theorist :
    Neste módulo, estão declaradas todas as rotinas necessárias para utilizarmos o Theorist em uma
    versão compilada.
*/


:- module theorist.


:- op(1150,fx,fact).
:- op(1150,fx,default).
:- op(1150,fx,predict).
:- op(1150,fx,explain).
:- op(1130,xfx,:).
:- op(1110,xfx,<-).
:- op(1110,xfx,=>).
:- op(1100,xfy,or).
:- op(1000,xfy,and).
:- op(1000,xfy,&).
:- op(950,fy,not).


:-visible recorded/3.
:-visible retract/1.
:-visible erase/1.
:- visible call/1.
:- visible delete_fact/1.
:- visible predicado_instanciado/1.
:- visible member /2.
:- visible prolog_cl/1.


:-public main/0.
```

```
/*
    member(A, Lista) :
    Verifica se o literal A pertence a Lista.
*/

member(A,[A|_]).

member(A,[X|Resto]) :-
 member(A,Resto).

/*
    append(L1, L2,L3) :
    Cria em L3, uma lista gerada pela união de L1 com L2.
*/
append([],L,L).

append([H|T],L,[H|T1]):-
 append(T,L,T1).

/*
    new_lit(Prefix, Reln, NewArgs, NewReln) :
    Constroe um novo átomo NewReln utilizando os argumentos de Prefix, Reln e NewArgs.
    Exemplo : Ao executarmos new_lit("ex_", father(a,b),[T,A,B], N), teremos N =
    ex_father(a,b,T,A,B).
*/
new_lit(Prefix,Reln,NewArgs,NewReln):-
 Reln =..[Pred|Args],
 name(Pred,PredName),
 append(Prefix,PredName,NewPredName),
 name(NewPred,NewPredName),
 append(Args,NewArgs,AllArgs),
 NewReln =..[NewPred|AllArgs].

/*
    predicado instanciado(Pred) :
Verifica todos os argumentos do predicado Pred estão instanciados.
*/
predicado_instanciado(Pred):-
 functor((Pred),NomePred,NumVar),
 verifica_variaveis(NumVar,Pred).

/*
    verifica variáveis(NumVar, Pred) :
    Verifica se a variável de índice NumVar do predicado denominado Pred está instanciada.
*/
verifica_variaveis(0,_).

verifica_variaveis(NumVar,Pred):-
 arg(NumVar,Pred,Var),
 nonvar(Var),
 dec(NumVar,NVar),
 verifica_variaveis(NVar,Pred).

/*
    make bodies(B, T, [Ths, Anc, Ans], ProveB, ExB) :
    Cria o corpo das estruturas de prova e explicação para a fórmula B, onde T é o cenário usado na
    prova de B, Ths é a teoria usada na sua explanação e Anc são os ancestrais usados durante a sua
    prova e explicação.
```

```
*/
make_bodies((H,B),T,[ths(T1,T3,D1,D3),Anc,ans(A1,A3)],
        (ProveH,ProveB),(ExH,ExB)):-
 !,
 make_bodies(H,T,[ths(T1,T2,D1,D2),Anc,ans(A1,A2)],ProveH,ExH),
 make_bodies(B,T,[ths(T2,T3,D2,D3),Anc,ans(A2,A3)],ProveB,ExB).

make_bodies((H;B),T,Ths,(ProveH;ProveB),(ExH;ExB)):-
 !,
 make_bodies(H,T,Ths,ProveH,ExH),
 make_bodies(B,T,Ths,ProveB,ExB).

make_bodies(n(A),T,[Ths,Anc,Ans],ProveA,ExA):-
 !,
 new_lit("prove_not_",A,[T,Anc],ProveA),
 new_lit("ex_not_",A,[Ths,Anc,Ans],ExA).

make_bodies(A,T,[Ths,Anc,Ans],ProveA,ExA):-
 new_lit("prove_",A,[T,Anc],ProveA),
 new_lit("ex_",A,[Ths,Anc,Ans],ExA).


/*
     rule(R) :
     Responsável por preparar as estruturas apropriadas para a fórmula R, onde R pode ser um literal
     ou ter a forma if(H,B) onde H é a cabeça da regra e B é o seu corpo.
*/

rule(if(H,B)):-
 !,
 make_anc(H),
 make_bodies(H,T,[Ths,Anc,Ans],ProveH,ExH),
 form_anc(H,Anc,Newanc),
 make_bodies(B,T,[Ths,Newanc,Ans],ProveB,ExB),
 call(prolog_cl((ProveH:-ProveB))),
 call(prolog_cl((ExH:-ExB))).

rule(H):-
 !,
 make_anc(H),
 make_bodies(H,T,[ths(T,T,D,D),_,ans(A,A)],ProveH,ExH),
 call(prolog_cl(ProveH)),
 call(prolog_cl(ExH)).

/*
     form_anc(L, A1, A2) :
     Cria a nova estrutura de ancestrais A2 a partir do sub-objetivo atual L e da estrutura de ancestrais
     anterior denominada A1.
*/
form_anc(n(G),anc(P,N),anc(P,[G|N])):-!.
form_anc(G,anc(P,N),anc([G|P],N)).

/*
     prolog_cl(C) :
     Responsável por inserir a cláusula C no Prolog.
*/
prolog_cl(X):-
 asserta(X).
```

```
/*
    declare fact(F) :
    Declara que F é um fato do Theorist. Para tal, ele primeiro converte F para a forma negativa
    normal (NNF).
*/
declare_fact(F):-
 nnf(F,even,N),
 rulify(N).

delete_fact(X):-
 recorded(fatos_ins,X,Ref),
 retract(X),
 erase(Ref),fail.

delete_fact(_).

opposite_parity(even,odd).
opposite_parity(odd,even).

/*
    nnf(Wff, Parity, Nnf) :
    Significa que Nnf é a forma negativa normal de :
    • Wff se a Parity for odd
    • Wff se a Parity for even.
*/
nnf((X => Y),P,B):-
 !,
 nnf((Y or not X),P,B).

nnf((Y <- X),P,B):-
 !,
 nnf((Y or not X),P,B).

nnf((X & Y),P,B):-
 !,
 nnf((Y and X),P,B).

nnf((X,Y),P,B):-
 !,
 nnf((X and Y),P,B).

nnf((X;Y),P,B):-
 !,
 nnf((X or Y),P,B).

nnf((X and Y),P,B):-
 !,
 opposite_parity(P,OP),
 nnf((not X or not Y),OP,B).

nnf((X or Y),even,(XB,YB)):-
 !,
 nnf(X ,even,XB),
 nnf(Y ,even,YB).

nnf((X or Y),odd,(XB,YB)):-
```

```
 !,
 nnf(X,odd,XB),
 nnf(Y,odd,YB).
/*
 nnf((~X),P,B):-
 !,
 nnf((not X),P,B).
*/
nnf((not X),P,B):-
 !,
 opposite_parity(P,OP),
 nnf(X ,OP,B).

nnf(F,odd,F).

nnf(n(F),even,F):-!.

nnf(F,even,n(F)).

/*
     rulify(N) :
     Ele é responsável pela criação das estruturas  de explicação e prova da fórmula N no Theorist
*/
rulify((A,B)):-
 !,
 contrapos(B,A),
 contrapos(A,B).

rulify((A;B)):-
 !,
 rulify(A),
 rulify(B).

rulify(n(A)):-
 !,
 rule(A).

rulify(A):-
 rule(n(A)).

/*
     contrapos(D,T) :
     Gera as contrapositivas da fórmula D ← T.
*/
contrapos(D,(L,R)):-
  !,
  contrapos((R,D),L),
  contrapos((L,D),R).

contrapos(D,(L;R)):-
  !,
  contrapos(D,L),
  contrapos(D,R).

contrapos(D,n(A)):-
  !,
  rule(if(A,D)).
```

```
contrapos(D,A):-
  rule(if(n(A),D)).

/*
     declare default (D) :
     Declara que D é um default para o Theorist.
*/
declare_default(D):-
 make_anc(D),
 new_lit("prove_",D,[T,_],Pr_D),
 call(prolog_cl((Pr_D:-member(D,T)))),
 new_lit("ex_",D,[ths(T,T,Defer,Defer),_,ans(A,A)],ExD),
 call(prolog_cl((ExD:-member(D,T)))),
 new_lit("ex_",D,[ths(T,[D|T],Defer,Defer),_,ans(A,A)],ExDass),
 new_lit("prove_not_",D,[[D|T],anc([],[])],Pr_not_D),
 call(prolog_cl((ExDass:-predicado_instanciado(D),\+ member(D,T),
        \+ Pr_not_D))),
 new_lit("ex_",D,[ths(T,T,Defer,[D|Defer]),_,ans(A,A)],ExDefer),
 call(prolog_cl((ExDefer:- \+ predicado_instanciado(D)))).

same_length([],[]).

same_length([H|T],Nargs):-
  same_length(T,Narg),
  Nargs = [ X | Narg].

/*
     make anc(Nome) :
     Cria as assertivas para verificar se o fato Nome já foi usado anteriormente na busca de uma prova
     ou explicação. Este predicado cria os mecanismos necessários para a prova por absurdo
*/
make_anc(Name):-
 call(ancestor_recorded(Name)),
 !.

make_anc(n(Goal)):-
 !,
 make_anc(Goal).

make_anc(Goal):-
 Goal =.. [Pred|Args],
 same_length(Args,Nargs),
 NG =..[Pred|Nargs],
 make_bodies(NG,_,[ths(T,T,D,D),anc(P,N),ans(A,A)],ProveG,ExG),
 make_bodies(n(NG),_,[ths(T,T,D,D),anc(P,N),ans(A,A)],ProvenG,ExnG),
 call(prolog_cl((ProveG:-member(NG,N)))),
 call(prolog_cl((ProvenG:- member(NG,P)))),
 call(prolog_cl((ExG:-member(NG,N)))),
 call(prolog_cl((ExnG:-member(NG,P)))),
 call(prolog_cl(ancestor_recorded(NG))).

numbervars('$VAR'(N),N,N1):-
 inc(N,N1).

numbervars(Term,N1,N2):-
 nonvar(Term),
 functor(Term,Name,N),
 numbervars(0,N,Term,N1,N2).
```

```
numbervars(N,N,Term,N1,N1).

numbervars(I,N,Term,N1,N3):-
 I < N,
 inc(I,I1),
 arg(I1,Term,Arg),
 numbervars(Arg,N1,N2),
 numbervars(I1,N,Term,N2,N3).

/*
     ground(Term) :
     Instancia as variáveis livres de Term com constantes únicas.
*/
ground(Term):-
 numbervars(Term,0,N).


/*
     expl(G, T0, T1, Ans) :
     Implementa a função explanação do Theorist, ou seja, ele tenta explicar a fórmula G a paritr da
     Teoria inical T0, gerando a nova teoria T1
*/
expl(G,T0,T1,Ans):-
 ground(N),
 declare_fact( ('<-'(newans(N,G),G))),
 call(ex_newans(N,G,ths(T0,T,[],D),anc([],[]),ans(G,Ans))),
 ground(D),
 check_consist(D,T,T1),
 delete_fact(X).

/*
     check consist(D, T, T1) :
     Checa a consistência do conjunto de defaults D utilizados na explicação da fórmula G.
*/
check_consist([],T,T).

check_consist([H|D],T1,T):-
 new_lit("prove_not_",H,[T1,anc([],[])],Pr_n_H),
 \+ Pr_n_H,
 check_consist(D,[H|T1],T).

/*
     fact(Fato) :
     Implementa a declaração Fato do Theorist onde Fato é uma fórmula bem formada.
*/
fact(F):-
 declare_fact(F),
 !.

/*
     default(N:H) :
     Implementa a declaração default do Theorist onde H é uma possível hipótese com nome N.
*/
default((N:H)):-
 !,
 declare_default(N),
 declare_fact((H '<-' N)),
```

```
!.

/*
    default(N) :
    Implementa a declaração default do Theorist onde N é o nome de uma possível hipótese.
*/
default(N):-
 declare_default(N),
 !.

/*
    explainall(G) :
    Representa o conceito explanação do Theorist onde G é a fómula que deve ser provada.
*/
explainall(Explicacao):-
 expl(Explicacao,[],D,Ans),
 write(D),nl,
 write(Ans),nl,
fail.

explainall(_).

escreve_titulo_2:-
 cls,
 tmove(2,10),
 write($ Explicacoes Logicas utilizando o Theorist $),
 tmove(3,10),
 write($ ========== ======= ========== = ======== $).

/*
    pede explicação(Observações) :
    Este predicado pergunta quais as Observações que você deseja explicação.
*/
pede_explicacao(Explicacao):-
 tmove(6,0),
 write($ Informe a explicacao desejada : $),
 nl,
 write($ ------- - ---------- -------- $),
 /* Limpa parte da tela */
 tscroll(7,(8,0),(15,79)),
 tmove(9,0),
 write($ - $),
 read_string(60,In),
 string_term(In,Explicacao),
 nl.

escreve_titulo_1:-
 cls,
 tmove(2,10),
 write($ Compilacao da Descricao para o Theorist $),
 tmove(3,10),
 write($ ========== == ========= ==== = ======== $).

/*
    pede nome do arquivo(ArquivoIn) :
    Pede nome do arquivo onde estão descritos os fatos e defaults sobre o sistema que se quer
    representar.
*/
```

```
pede_nome_do_arquivo(ArqIn):-
 tmove(6,0),
 write($ Informe o nome do arquivo : $),
 nl,
 write($ ------- - ---- --- -------- $),
 /* Limpa parte da tela */
 tscroll(7,(8,0),(15,79)),
 tmove(9,0),
 write($  Arquivo Inicial : $),
 read_string(13,In),
 atom_string(ArqIn,In),
 nl.

/*
     le escreve(ArquivoIn) :
     Le os fatos e defaults descritos no arquivo ArquivoIn e os transcreve para a versão compilada do
     Theorist.
*/
le_escreve(ArqIn):-
 open(In,ArqIn,r),
 repeat,
 read(In,Formula),
 escreve_Theorist(Formula),
 Formula = end_of_file,
 close(In).

/*
     escreve Theorist(F) :
     Transcreve para a versão compilada do Theorist a fómula F.
*/
escreve_Theorist(end_of_file):-!.

escreve_Theorist(fact(X)):-
 !,
 fact(X).

escreve_Theorist(default(X)):-
 !,
 default(X).

/*
     prepara dados :
     Predicado responsável por ler o arquivo onde estão descritos os fatos e defaults que queremos
     representar e transcrevê-los para a versão complilada do Theorist.
*/
prepara_dados:-
 escreve_titulo_1,
 pede_nome_do_arquivo(ArqIn),
 le_escreve(ArqIn).

/*****************************************************************************

                          PROGRAMA PRINCIPAL

*****************************************************************************/

main :-
/* Le arquivo e prepara os fatos e defaults para a versão compilada do Theorist */
```

```
prepara_dados,
abolish(prolog_cl/1),
asserta((prolog_cl(X):- asserta(X), recordz(fatos_ins,X,_))),
/*
Procura explicação para as observações do sistema descrito até que o usuário deseje finalizar.
*/
repeat,
escreve_titulo_2,
pede_explicacao(Explicacao),
explainall(Explicacao),
nao_quer_mais(Resp),
Resp = nao.


/*
    nao_quer_mais(Resp) :
    Verifica se o usuário deseja continuar pedindo explicações ou não.
*/
nao_quer_mais(Resp) :-
 write($ Digite N para finalizar $),
 nl,
 write($ ou qualquer outra tecla para continuar no programa.  $),
 nl,
 keyb(X,Y),
 (
  /* Teclou N */
  X=78,Y=49,Resp = nao;
  /* Teclou n */
  X=110,Y=49, Resp =nao;
 /* Qualquer outra tecla -> continua */
 Resp = sim).
```

Apêndice C - Diagnose Baseada na Consistência

Neste apêndice, vamos descrever mais detalhadamente sobre este método de diagnose baseada na lógica bastante utilizado na literatura [Reiter 87], [de Kleer *et. al.* 90], [de Kleer *et. al.* 92], [Poole 88a], [Poole 89].

1 - Introdução :

O modelo, desenvolvido por [Reiter 87], é um conjunto **(SD,COMPS,OBS)** onde :

- ***SD*** é a descrição do sistema representado por um conjunto de sentenças de 1ª ordem, onde deve ser caracterizado o seu funcionamento.
- ***COMPS*** são os componentes representados por um conjunto finito de constantes.
- ***OBS*** é o conjunto de observações representado também por um conjunto de sentenças de 1ª ordem.

Para fins de representação, ele definiu que quando um componente C não estiver funcionando corretamente, o ***comportamento de C*** deve ser caracterizado como ***anormal,*** ou seja, ***AN(C)***.

Exemplo: Considere um circuito com duas baterias em série, onde uma bateria opera normalmente com a tensão entre 12 volts e 16 volts. A partir desta descrição, podemos destacar as seguintes relações :

- BATERIA(B) → informa que B é uma bateria
- AN(B) → indica que a bateria B está trabalhando corretamente
- TENSÃO(B,V) → significa que a tensão da bateria B é V

- SÉRIE(B1,B2) → indica que as baterias B1 e B2 estão em série

Assim, podemos caracterizar o funcionamento normal da bateria da seguinte forma :

$\forall B\ BATERIA(B) \wedge \neg AN(B) \rightarrow \exists V\ (V \geq 12 \wedge V \leq 16 \wedge TENSÃO(B,V))$

O funcionamento das duas baterias em série pode ser representado por :

$\forall B1 \forall B2 \forall V \exists V1 \exists V2\ [TENSÃO(SÉRIE(B1,B2),V)] \rightarrow (V=V1+V2) \wedge TENSÃO(B1,V1) \wedge TENSÃO(B2,V2)$

Neste exemplo :

- o SD (ou seja, a descrição de funcionamento do sistema) está representado pelas duas fórmulas acima,
- o COMPS são as duas baterias : B1 e B2 e
- OBS são as tensões das baterias (TENSÃO(B1,V1), TENSÃO(B2,V2), TENSÃO(SÉRIE(B1,B2),V3), ou seja, as possíveis observações sobre o funcionamento.

Suponha, que foi observado que a soma da tensão das duas baterias (B1 e B2) é de 15 volts, ou seja, TENSÃO(SÉRIE(B1,B2), 15).

Por esta observação, podemos verificar que o circuito está apresentando problemas já que a soma de duas baterias que estão funcionando corretamente é no mínimo de 24 volts. Devemos então, tentar diagnosticar o defeito para tal, primeiramente vamos mostrar como caracterizar uma ***diagnose baseada na consistência***.

Quando um sistema está funcionado corretamente, podemos dizer que todos os seus componentes estão normais, ou seja, $\{\neg AN(C_1), ..., \neg AN(C_N)\}$.

Entretanto, ao observamos um desvio do seu funcionamento correto como o ocorrido no exemplo acima, devemos concluir que houve uma mudança no comportamento de um ou mais componentes deste sistema, ou seja, o estado

normal de funcionamento do sistema (SD $\cup$ $\{\neg AN(C_1), ..., \neg AN(C_N)\}$) passou a ser inconsistente com as observações obtidas.

Durante estes distúrbios, nosso objetivo é diagnosticar o motivo do mau-funcionamento, ou seja, é descobrir quais os componentes que mudaram de comportamento passando a não funcionar corretamente [Reiter 87] [de Kleer *et. al.* 90].

Este conjunto de componentes defeituosos deve ser capaz de explicar as observações obtidas (OBS) com os demais componentes sendo ainda considerados normais.

Assim, a diagnose pode ser caracterizada mais amplamente como sendo o estado de comportamento do sistema representado por $D(\Delta,COMPS-\Delta)$ onde $\{AN(C) \mid C \in \Delta\} \cup \{\neg AN(C) \mid C \in COMPS - \Delta\}$, ou seja, Δ é o conjunto de componentes anormais e $COMPS - \Delta$, o conjunto de componentes normais [de Kleer *et. al.* 90].

Deste modo, no exemplo das baterias, podemos dizer que existem três diagnoses possíveis para a observação TENSÃO(SÉRIE(B1,B2), 15):

- Apenas a bateria B1 está ruim representada por D(B1,B2),
- Apenas a bateria B2 está ruim representada por D(B2,B1) e
- Ambas as baterias estão com problemas representada por $D(\{B1,B2)\}, \varnothing)$.

Formalmente, a diagnose pode ser definida como :

DEFINIÇÃO 1 - Uma ***diagnose para (SD,COMPS,OBS)*** é um conjunto $D(\Delta,COMPS-\Delta)$ tal que SD $\cup$ OBS $\cup$ $D(\Delta,COMPS-\Delta)$ seja consistente e $\Delta \subseteq COMPS$ [de Kleer *et. al.* 90].

Porém, o objetivo inicial de [Reiter 87] não era determinar uma diagnose mas sim, determinar todas as minimas diagnoses para (SD,COMPS,OBS) como veremos a seguir.

2 - Diagnose Minimal :

Uma ***diagnose mínimal*** é o menor conjunto Δ de suposições de anormalidade, que ainda possa explicar as observações obtidas (OBS) durante o funcionamento do sistema (SD).

Assim, no exemplo das baterias, vemos que as duas primeiras diagnoses são consideradas minimais ({AN(B1)}, {AN(B2)}) e a terceira é um superconjunto das outras duas ({AN(B1), AN(B2)}).

Formalmente, podemos caracterizá-la como:

DEFINIÇÃO 2 - Uma ***diagnose D(Δ,COMPS-Δ)*** é uma diagnose minimal sse nenhum subconjunto próprio Δ' de Δ for também uma diagnose D(Δ',COMPS-Δ') [de Kleer *et. al.* 90].

[Reiter 87] dava tanta importância as diagnoses minimais porque ele achava que a partir delas, poderia gerar qualquer diagnose, ou seja, ele pensava que todas as diagnoses eram superconjuntos de alguma diagnose minimal.

Para atingir este objetivo de Reiter, ou seja, determinar todas as diagnoses minimais para (SD,COMPS,OBS), precisamos conhecer dois novos conceitos: ***Conflito e "Hitting Set".***

Sabemos que um ***conflito*** é um passo intermediário na determinação de um diagnóstico. O principal objetivo de um diagnóstico é determinar defeitos e a fonte de um diagnóstico é exatamente a discrepância (ou seja, o conflito) entre o que se espera e o que foi observado em um sistema.

Mais formalmente, [de Kleer *et al.* 90] caracterizou *um conflito para (SD,COMPS,OBS)* a partir das seguintes definições :

DEFINIÇÃO 3 - Uma *AN-Clausal* é uma disjunção de literais, onde cada literal representa o estado de funcionamento de um determinado componente.

Exemplo : $AN(C_1) \vee \neg AN(C_2) \vee ... \vee AN(C_N)$.

Devemos destacar que uma AN-Clausal não pode conter nenhum par complementar de um mesmo literal.

Ela é caracterizada como **positiva**, se todos os seu literais forem positivos, ou seja, ***AN(C) para todo*** $C \in$ ***AN-Clausal.***

DEFINIÇÃO 4 - Um ***conflito para (SD,COMPS,OBS)*** é qualquer ***AN-Clausal X*** tal que ***X*** seja conseqüência lógica da descrição do sistema (SD) com as observações (OBS), ou seja, $SD \cup OBS \models X$.

Pela lógica de 1ª ordem, sabemos que $SD \cup OBS \models X$ ***sse*** $SD \cup OBS \cup \neg X$ ***seja insatisfatível.***

No exemplo das baterias, a união da descrição do sistema:

$\forall B\ BATERIA(B) \wedge \neg AN(B) \rightarrow \exists V\ (V \geq 12 \wedge V \leq 16 \wedge TENSÃO(B,V))$

$\forall B1 \forall B2 \forall V \exists V1 \exists V2\ [TENSÃO(SÉRIE(B1,B2),V)] \rightarrow (V=V1+V2) \wedge$
$TENSÃO(B1,V1) \wedge TENSÃO(B2,V2)$

com a observação TENSÃO(SÉRIE(B1,B2), 15)) é insatisfatível com o fato das duas baterias estarem funcionando corretamente ($\neg AN(B1) \wedge \neg AN(B2)$).

Assim, $AN(B1) \vee AN(B2)$ é um conflito para este (SD,COMPS,OBS).

DEFINIÇÃO 5 - Um ***conflito minimal para (SD,COMPS,OBS)*** é um conflito tal que nenhum subconjunto dele próprio seja também um conflito para (SD,COMPS,OBS).

A partir destas definições, podemos caracterizar uma diagnose em função de um conflito :

TEOREMA 1 [de Kleer *et. al.* 90 pp. 326] - Suponha que π é o conjunto dos conflitos para (SD,COMPS,OBS) e $\Delta \subseteq$ COMPS, então $D(\Delta, COMPS-\Delta)$ é uma diagnose sse $\pi \cup D(\Delta, COMPS-\Delta)$ é satisfatível.

Vamos agora, mostrar como podemos obter uma diagnose a partir do conjunto de conflitos. Para tal, iremos caracterizar o que é um "hitting set" [Reiter 87] :

DEFINIÇÃO 6 - Suponha C como uma coleção de conjuntos. A ***"Hitting Set"*** para C é um conjunto $H \subseteq \cup_{S \in C} S$ tal que $H \cap S \neq \{\}$ para cada $S \in C$, ou seja, um ***"Hitting Set"*** para a coleção de conjuntos C é um conjunto que possui pelo menos um elemento de cada conjunto de C.

DEFINIÇÃO 7 - Um ***"Hitting Set" para C*** é minimal sse nenhum subconjunto dele próprio é um "Hitting Set".

A partir destas definições de "Hitting Set", vamos caracterizar uma diagnose minimal em função de um conflito :

TEOREMA 2 [Reiter 87 pp. 67] - $D(\Delta, COMPS-\Delta)$ é uma diagnose minimal para (SD,COMPS,OBS) sse Δ é o mínimo "Hitting Set" para a coleção de conflitos minimais positivos de (SD,COMPS,OBS) e $\Delta \subseteq$ COMPS.

Assim, podemos agora no exemplo das baterias, obter as diagnoses minimais em função dos conceitos descritos acima, onde os dois mínimos "hitting sets" para o único conflito positivo existente $(AN(B1) \vee AN(B2))$ são AN(B1) e AN(B2).

Podemos então pelo teorema 2, afirmar que existem apenas duas diagnoses minimais :

- D(B1,B2), ou seja, apenas B1 está com defeito e
- D(B2,B1), ou seja, apenas B2 está com defeito.

Até o momento nos preocupamos apenas em descrever o funcionamento normal do sistema. Vamos a partir de agora, mostrar também como descrever as falhas em um sistema.

Para tal, acrescentaremos dois tipos de falha às baterias:

- BAIXA_CARGA(B) que significa que a bateria B apresenta uma tensão abaixo da tensão mínima normal de 12 volts e
- CURTO(B) que indica que a bateria B está em curto. Estes novos conhecimentos são representados por :

$$\forall B \forall V\ [TENSÃO(B,V) \wedge (0 < V < 12)] \rightarrow BAIXA\_CARGA(B)$$

$$\forall B\ TENSÃO(B,0) \rightarrow CURTO(B)$$

A partir dos fatos que descreviam o seu funcionamento normal e da observação (TENSÃO(SÉRIE(B1,B2),15)) mostrados anteriormente neste mesmo exemplo, se assumirmos que B1 está funcionando corretamente ($\neg$ AN(B1)), podemos provar que a tensão de B2 está entre : a tensão observada das duas baterias em série menos a tensão máxima normal de B1 (15 - 16) e a tensão observada das duas baterias em série menos a tensão mínima normal de B1 (15 - 12), ou seja :

$$\exists V\ TENSÃO(B2,V) \wedge (-1 < V < 3).$$

Esta dedução é inconsistente com o fato de B2 estar funcionado corretamente ($\neg$AN(B2)), porém nós ainda não podemos descobrir o tipo de falta que a caracteriza. Para tal, precisamos explicitar todos os estados que uma bateria pode assumir, descritos por :

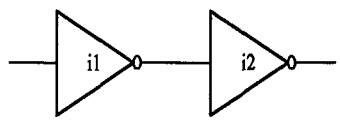
$$\forall B\ BATERIA(B) \rightarrow \neg AN(B) \vee CURTO(B) \vee BAIXA\_CARGA(B)$$

Com isso, para a observação TENSÃO(SÉRIE(B1,B2),15)) passamos a poder provar que se B1 está ok então B2 tem que estar em curto ou com baixa carga :

$$\neg AN(B1) \rightarrow CURTO(B2) \vee BAIXA\_CARGA(B2).$$

Mas ao acrescentar a descrição de falhas do sistema no SD, podemos encontrar problemas na obtenção de diagnoses a partir da diagnose minimal como mostraremos no exemplo a seguir.

Considere um circuito com dois inversores :



onde o funcionamento normal do inversor está representado por:

$$\forall X\ INVERSOR(X) \wedge \neg AN(X) \rightarrow [OUT(X) = \neg\ IN(X)]$$

e a conexão entre eles descrita pela fórmula :

$$OUT(I1) = IN(I2).$$

Para simplificação, este circuito apresenta apenas um único tipo de falha:

- CURTO(X) → que significa que a saída do inversor está sempre com o mesmo valor que a entrada. Esta falha pode ser descrita pelo seguinte fato:

$$\forall X\ [OUT(X) = IN(X)\ ] \rightarrow CURTO(X)$$

Para obtermos uma descrição mais completa da diagnose, devemos explicitar todos os estados que um inversor pode assumir. Neste exemplo, este estados são caracterizado por :

$$\forall X\ INVERSOR(X) \rightarrow \neg AN(X) \vee CURTO(X).$$

Se observarmos neste circuito, as seguintes medidas: IN(I1) = 0 e OUT(I2) = 1, teremos então, dois conflitos minimais possíveis :

$$\{AN(I1) \vee AN(I2)\} \text{ e } \{\neg AN(I1) \vee \neg AN(I2)\}.$$

Estes dois conflitos são obtidos a partir da definição 4 de conflito (SD ∪ OBS ∪ ¬CONFLITO é insatisfatível) pois, a união da descrição do sistema (SD) :

$$\forall X\ INVERSOR(X) \wedge \neg AN(X) \rightarrow [OUT(X) = \neg\ IN(X)],$$

$$OUT(I1) = IN(I2),$$

$$\forall X\ \ [OUT(X) = IN(X)] \rightarrow CURTO(X) \text{ e}$$

$$\forall X\ INVERSOR(X) \rightarrow \neg AN(X) \vee CURTO(X))$$

com as observações : IN(I1) = 0 e OUT(I2) = 1 são incompatíveis com as suposições dos dois componentes estarem funcionado corretamente (¬AN (I1) ∧ ¬AN(I2)) bem como de ambos estarem com defeito (AN(I1) ∧ AN(I2)).

A partir do conflito minimal positivo (AN(I1) ∨ AN(I2)) e do teorema 2 obtemos duas diagnoses minimais : {AN(I1)} e {AN(I2)}.

Porém, não é possível que ambos componentes estejam com defeito, ou seja, o superconjunto destas duas diagnoses minimais não pode ser também uma diagnose, contrariando o pensamento de [Reiter 87] sobre a diagnose minimal.

Isto ocorre porque um dos conflitos minimais não é positivo: $\neg AN(I1) \vee \neg AN(I2)$. Desta maneira, não se pode afirmar nada a partir da diagnose minimal, ou seja, você só pode afirmar que $D(\Delta,COMPS-\Delta)$ é uma diagnose obtida a partir da diagnose minimal $D(\Delta',COMPS-\Delta')$ tal que $COMPS \supseteq \Delta \supseteq \Delta'$ se todos os conflitos minimais forem positivos [de Kleer *et. al.* 90].

Assim, podemos concluir que nem sempre um superconjunto de uma diagnose minimal também é uma diagnose.

Este tipo de problema foi originado porque [Reiter 87] só se preocupava em descrever o funcionamento normal do sistema. Deste modo, os conflitos seriam sempre positivos [de Kleer *et. al.* 90 pp. 324].

Porém, a partir do momento que se deseja descrever também as falhas para tornar a descrição do sistema mais completa, conflitos não positivos podem ser gerados e com eles, a diagnose minimal passa a ser inadequada para caracterizar todas as diagnoses.

Para resolver este problema, [de Kleer *et al.* 90] apresentou uma nova alternativa de diagnose : a ***"kernel diagnose"***.

3 - Kernel Diagnose :

Para entendermos melhor o que é uma *Kernel Diagnose*, precisamos primeiramente caracterizar dois novos conceitos : ***Cobertura e "Implicant"*** [de Kleer *et. al.* 90].

DEFINIÇÃO 8 - Uma conjunção C de literais ***"cobre"*** uma conjunção D de literais sse todo o literal de C ocorre em D. Exemplo: $C = A \wedge B$ e $D = A \wedge B \wedge E$ logo, C "cobre" D.

DEFINIÇÃO 9 - A ***"implicant"*** de Σ é uma conjunção de literais π que não contenha nenhum par complementar de literais entre eles, tal que $\pi \models \Sigma$, ou seja, Σ é uma consequência lógica de π, onde Σ é verdadeira para todos os modelos de π.

DEFINIÇÃO 10 - π é um ***"prime implicant"*** se ao retirarmos qualquer um dos seus literais, ele passa a não ser mais um "implicant" de Σ, ou seja, o único "implicant" de Σ que "cobre" π é o próprio π.

A partir das três definições apresentadas acima, podemos caracterizar o que é uma diagnose parcial e em conseqüência, definirmos uma kernel diagnose.

Uma ***diagnose parcial*** pode ser facilmente caracterizada a partir de um simples exemplo : Suponha que existam apenas duas diagnoses possíveis para um sistema :

$$AN(C1) \wedge AN(C2) \wedge AN(C3) \text{ e}$$

$$AN(C1) \wedge AN(C2) \wedge \neg AN(C3)),$$

podemos interpretá-las dizendo que C1 e C2 estão ruins e C3 pode ou não estar com defeito.

Deste modo, podemos compactar as duas diagnoses em uma só: $AN(C1) \wedge AN(C2)$, vista como uma diagnose parcial onde o estado de C3 passa a não importar tanto, já que qualquer que seja o seu estado ele levara a uma diagnose.

Assim, podemos dizer que P é uma diagnose parcial se todos os superconjuntos de P forem também diagnoses.

Mais formalmente, [de Kleer *et. al.* 90] a caracterizou como:

DEFINIÇÃO 11 - Uma ***diagnose parcial*** para (SD,COMPS,OBS) é uma conjunção satisfatível P de AN-literais em que qualquer conjunção satisfatível ϕ de AN-literais, que é coberta por P (ou seja, contenha todos os literais de P além de outros literais) seja tal que $SD \cup OBS \cup \phi$ é satisfatível.

Desta maneira, uma diagnose parcial representa o conjunto de todas as diagnoses que a contenham como subconjunto.

DEFINIÇÃO 12 - Uma diagnose parcial mínima é denominada de ***Kernel diagnose,*** ou seja, a única diagnose que pode cobrir uma kernel diagnose é ela própria.

Podemos então, caracterizar uma diagnose em função destes novos conceitos :

TEOREMA 3 [de Kleer *et. al.* 90 pp. 327] - D(Δ,COMPS-Δ) é uma diagnose sse existe uma kernel diagnose que é um subconjunto dela.

TEOREMA 4 [de Kleer *et. al.* 90 pp. 328] - As ***diagnoses parciais de (SD,COMPS,OBS)*** são os "implicants" dos conflitos minimais de (SD,COMPS,OBS).

TEOREMA 5 [de Kleer *et. al.* 90 pp. 328] - Os ***kernel diagnoses*** são os "prime implicants" dos conflitos minimais de SD ∪ OBS.

Vamos ilustrar estes conceitos, voltando ao exemplo dos inversores, onde sabemos que existem dois conflitos minimais possíveis :

$$AN(I1) \vee AN(I2) \text{ e } \neg AN(I1) \vee \neg AN(I2).$$

Teremos então duas kernel diagnoses :

$$AN(I1) \wedge \neg AN(I2) \text{ e } \neg AN(I1) \wedge AN(I2).$$

Devemos notar, que a diferença entre a kernel diagnose e a diagnose minimal é o conflito não positivo, ou seja, se todos os conflitos minimais forem positivos então não existirá diferença entre elas [de Kleer *et. al.* 90 pp.328].

Porém apesar de neste exemplo, existir o mesmo número de kernel diagnoses ($\{AN(I1) \wedge \neg AN(I2)\}$ e $\{\neg AN(I1) \wedge AN(I2)\}$) e de diagnoses minimais($\{AN(I1)\}$ e $\{AN(I2)\}$), o normal é que o número de kernel diagnoses cresça exponencialmente em relação ao de diagnoses minimais [de Kleer *et. al.* 90 pp.328].

Para evitar esta complexidade, que tornaria a computabilidade do problema intratável, devemos criar restrições no OBS e no SD de tal modo que todos os conflitos minimais sejam positivos. Dentre as técnicas que descrevem tais restrições, devemos destacar as apresentadas em [de Kleer *et. al.* 92].

Finalizando, podemos dizer, a partir dos conceitos e propriedades mostradas acima, que a diagnose minimal não é adequada para caracterizar qualquer tipo de diagnose e que a kernel diagnose demonstrou ser um método mais apropriado para realizar um diagnóstico baseado na consistência.

Apêndice D - Descrição das Falhas do Caso Exemplo utilizando a Diagnose Abdutiva e o Theorist

Neste apêndice, são apresentadas a caracterização de todas as falhas do sistema elétrico simplificado de uma usina (tabelas II.1 e II.2). Devemos ressaltar, que as características individuais de cada uma das falhas são descritas por fatos e apenas uma possível hipótese genérica permite representar qualquer falha desde que a condição mínima de alarmes seja satisfeita.

1) Descrição das Falhas utilizando o Theorist :

Para realizar uma descrição completa vamos inicialmente, descrever os componentes deste sistema , onde sabemos que existem três tipos de componentes possíveis : Linha de transmissão, Transformador e Disjuntor de by-pass.

No nosso caso exemplo, temos duas linhas de transmissões (linha_transmissão_01 e linha_transmissão_02), dois transformadores (transformador_01 e transformador_02) e um único disjuntor de by-pass (by_pass). Estes componentes serão representados pelos seguintes fatos :

fact(é_linha_transmissão(linha_transmissão_01)).

fact(é_linha_transmissão(linha_transmissão_02)).

fact(é_transformador(transformador_01)).

fact(é_transformador(transformador_02)).

fact(é_by_pass(by_pass)).

Sabemos que os componentes (linha de transmissão e transformador) podem ser auxiliados pelo disjuntor de by-pass. Este fato é representado do seguinte modo:

fact((comp_auxiliado_pelo_by_pass(X) ← é_linha_transmissão(X))).

fact((comp_auxiliado_pelo_by_pass(X) ← é_transformador(X))).

A partir desta descrição dos componentes, podemos mostrar mais detalhadamente a caracterização das falhas do sistema. Para tal, vamos descrever primeiramente a possível hipótese utilizada nas explicacões dos alarmes :

default((descrição_falha(Origem,Local,Comp,Comp_aux) : (falha(Origem, Local, Comp, Comp_aux) ← mínima_caracterização(Origem ,Local, Comp, Comp_aux))))).

Agora, iremos representar cada uma das falhas descritas nas tabelas II.1 e II.2 :

1) Falha próxima na linha de transmissão 01 f/f :

fact((alr(abriu,linha_transmissão_01) ← falha(fase,próximo,linha_transmissão_01,-))).

fact((alr(fase,linha_transmissão_01) ← falha(fase,próximo,linha_transmissão_01,-))).

fact((alr(instantâneo,linha_transmissão_01) ← falha(fase,próximo,linha_transmissão_01,-))).

fact((mínima_caracterização(fase,próximo,linha_transmissão_01,-) ← (atuado_alarme(fase,linha_transmissão_01),atuado_alarme(instantâneo,linha_transmissão_01)))).

2) Falha próxima na linha de transmissão 01 f/f (1) :

fact((alr(abriu,by_pass) ← falha(fase,próximo,by_pass,linha_transmissão_01))).

fact((alr(fase,by_pass) ←

falha(fase,próximo,by_pass,linha_transmissão_01))).

fact((alr(instantâneo,by_pass) ←

falha(fase,próximo,by_pass,linha_transmissão_01))).

fact((alr(aux,linha_transmissão_01) ←

falha(fase,próximo,by_pass,linha_transmissão_01))).

fact((mínima_caracterização(fase,próximo,by_pass,Comp) ←

(atuado_alarme(aux,Comp),atuado_alarme(fase,by_pass),

atuado_alarme(instantâneo,by_pass)))).

3) Falha próxima na linha de transmissão 01 f/t :

fact((alr(abriu,linha_transmissão_01) ←

falha(terra,próximo,linha_transmissão_01,-))).

fact((alr(terra,linha_transmissão_01) ←

falha(terra,próximo,linha_transmissão_01,-))).

fact((alr(instantâneo,linha_transmissão_01) ←

falha(terra,próximo,linha_transmissão_01,-))).

fact((mínima_caracterização(terra,próximo,linha_transmissão_01,-) ←

(atuado_alarme(terra,linha_transmissão_01),atuado_alarme(instantâneo,lin

ha_transmissão_01)))).

4) Falha próxima na linha de transmissão 01 f/t (1) :

fact((alr(abriu,by_pass) ←

falha(terra,próximo,by_pass,linha_transmissão_01))).

fact((alr(terra,by_pass) ←

falha(terra,próximo,by_pass,linha_transmissão_01))).

fact((alr(instantâneo,by_pass) ←

falha(terra,próximo,by_pass,linha_transmissão_01))).

fact((alr(aux,linha_transmissão_01) ←

falha(terra,próximo,by_pass,linha_transmissão_01))).

fact((mínima_caracterização(terra,próximo,by_pass,Comp) ←

(atuado_alarme(aux,Comp),atuado_alarme(terra,by_pass),

atuado_alarme(instantâneo,by_pass)))).

5) Falha distante na linha de transmissão 01 f/f :

fact((alr(abriu,linha_transmissão_01) ←

falha(fase,distante,linha_transmissão_01,-))).

fact((alr(fase,linha_transmissão_01) ←

falha(fase,distante,linha_transmissão_01,-))).

fact((alr(temporizado,linha_transmissão_01) ←

falha(fase,distante,linha_transmissão_01,-))).

fact((mínima_caracterização(fase,distante,linha_transmissão_01,-) ←

(atuado_alarme(fase,linha_transmissão_01),atuado_alarme(temporizado,linh

a_transmissão_01)))).

6) Falha distante na linha de transmissão 01 f/f (1) :

fact((alr(abriu,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_01))).

fact((alr(fase,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_01))).

fact((alr(temporizado,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_01))).

fact((alr(aux,linha_transmissão_01) ←

falha(fase,distante,by_pass,linha_transmissão_01))).

fact((mínima_caracterização(fase,distante,by_pass,Comp) ←

(atuado_alarme(aux,Comp),atuado_alarme(fase,by_pass),

atuado_alarme(temporizado,by_pass)))).

7) Falha distante na linha de transmissão 01 f/t :

fact((alr(abriu,linha_transmissão_01) ←

falha(terra,distante,linha_transmissão_01,-))).

fact((alr(terra,linha_transmissão_01) ←

falha(terra,distante,linha_transmissão_01,-))).

fact((alr(temporizado,linha_transmissão_01) ←

falha(terra,distante,linha_transmissão_01,-))).

fact((mínima_caracterização(terra,distante,linha_transmissão_01,-) ←

(atuado_alarme(terra,linha_transmissão_01),atuado_alarme(temporizado,lin

ha_transmissão_01))))).

8) Falha distante na linha de transmissão 01 f/t (1) :

fact((alr(abriu,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_01))).

fact((alr(terra,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_01))).

fact((alr(temporizado,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_01))).

fact((alr(aux,linha_transmissão_01) ←

falha(terra,distante,by_pass,linha_transmissão_01))).

fact((mínima_caracterização(terra,distante,by_pass,Comp) ←

(atuado_alarme(aux,Comp), atuado_alarme(terra,by_pass),

atuado_alarme(temporizado,by_pass)))).

9) Falha próxima na linha de transmissão 02 f/f :

fact((alr(abriu,linha_transmissão_02) ←

falha(fase,próximo,linha_transmissão_02,-))).

fact((alr(fase,linha_transmissão_02) ←

falha(fase,próximo,linha_transmissão_02,-))).

fact((alr(instantâneo,linha_transmissão_02) ←

falha(fase,próximo,linha_transmissão_02,-))).

fact((mínima_caracterização(fase,próximo,linha_transmissão_02,-) ←

(atuado_alarme(fase,linha_transmissão_02),atuado_alarme(instantâneo,linh

a_transmissão_02))))).

10) Falha próxima na linha de transmissão 02 f/f (2) :

fact((alr(abriu,by_pass) ←

falha(fase,próximo,by_pass,linha_transmissão_02))).

fact((alr(fase,by_pass) ←

falha(fase,próximo,by_pass,linha_transmissão_02))).

fact((alr(instantâneo,by_pass) ←

falha(fase,próximo,by_pass,linha_transmissão_02))).

fact((alr(aux,linha_transmissão_02) ←

falha(fase,próximo,by_pass,linha_transmissão_02))).

11) Falha próxima na linha de transmissão 02 f/t :

fact((alr(abriu,linha_transmissão_02) ←

falha(terra,próximo,linha_transmissão_02,-))).

fact((alr(terra,linha_transmissão_02) ←

falha(terra,próximo,linha_transmissão_02,-))).

fact((alr(instantâneo,linha_transmissão_02) ←

falha(terra,próximo,linha_transmissão_02,-))).

fact((mínima_caracterização(terra,próximo,linha_transmissão_02,-) ←

(atuado_alarme(terra,linha_transmissão_02),atuado_alarme(instantâneo,lin

ha_transmissão_02))))).

12) Falha próxima na linha de transmissão 02 f/t (2) :

fact((alr(abriu,by_pass) ←

falha(terra,próximo,by_pass,linha_transmissão_02))).

fact((alr(terra,by_pass) <-

falha(terra,próximo,by_pass,linha_transmissão_02))).

fact((alr(instantâneo,by_pass) ←

falha(terra,próximo,by_pass,linha_transmissão_02))).

fact((alr(aux,linha_transmissão_02) ←

falha(terra,próximo,by_pass,linha_transmissão_02))).

13) Falha distante na linha de transmissão 02 f/f :

fact((alr(abriu,linha_transmissão_02) ←

falha(fase,distante,linha_transmissão_02,-))).

fact((alr(fase,linha_transmissão_02) ←

falha(fase,distante,linha_transmissão_02,-))).

fact((alr(temporizado,linha_transmissão_02) ←

falha(fase,distante,linha_transmissão_02,-))).

fact((mínima_caracterização(fase,distante,linha_transmissão_02,-) ←

(atuado_alarme(fase,linha_transmissão_02),atuado_alarme(temporizado,linh

a_transmissão_02)))).

14) Falha distante na linha de transmissão 02 f/f (2) :

fact((alr(abriu,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_02))).

fact((alr(fase,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_02))).

fact((alr(temporizado,by_pass) ←

falha(fase,distante,by_pass,linha_transmissão_02))).

fact((alr(aux,linha_transmissão_02) ←

falha(fase,distante,by_pass,linha_transmissão_02))).

15) Falha distante na linha de transmissão 02 f/t :

fact((alr(abriu,linha_transmissão_02) ←

falha(terra,distante,linha_transmissão_02,-))).

fact((alr(terra,linha_transmissão_02) ←

falha(terra,distante,linha_transmissão_02,-))).

fact((alr(temporizado,linha_transmissão_02) ←

falha(terra,distante,linha_transmissão_02,-))).

fact((mínima_caracterização(terra,distante,linha_transmissão_02,-) ←

(atuado_alarme(terra,linha_transmissão_02),atuado_alarme(temporizado,lin

ha_transmissão_02))))).

16) Falha distante na linha de transmissão 02 f/t (2) :

fact((alr(abriu,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_02))).

fact((alr(terra,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_02))).

fact((alr(temporizado,by_pass) ←

falha(terra,distante,by_pass,linha_transmissão_02))).

fact((alr(aux,linha_transmissão_02) ←

falha(terra,distante,by_pass,linha_transmissão_02))).

17) Falhas relacionadas com a Barra Principal :

fact((alr(abriu,linha_transmissão_01) ←falha(barra_princ,-,todos,_))).

fact((alr(abriu,linha_transmissão_02) ←falha(barra_princ,-,todos,_))).

fact((alr(abriu,transformador_01) ←falha(barra_princ,-,todos,_))).

fact((alr(abriu,transformador_02) ←falha(barra_princ,-,todos,_))).

fact((alr(abriu,by_pass) ← comp_auxiliado_pelo_by_pass(Comp)
falha(barra_princ,-,todos,Comp))).
fact((alr(aux,Comp) ← falha(barra_princ,-,todos,Comp))).
fact((not(falha(barra_princ,-,todos,-)) ← atuado_alarme(aux,_))).
fact((not(falha(barra_princ,-,todos,-)) ← atuado_alarme(_,by_pass))).

Devemos ressaltar que estes dois últimos fatos deveriam ser substituidos pelos "constraints" :

constraint((descrição_falha(barra_princ,-,todos,-) ←
atuado_alarme(aux,_))).
constraint((descrição_falha(barra_princ,-,todos,-)) ←
atuado_alarme(_,by_pass))).

mas como este comando não foi implementado pelas versões interpretada e compilada do Theorist, tivemos que descrevê-los através dos dois fatos mostrados acima. Entretanto, devemos destacar que estes dois fatos podem ocasionar efeitos colaterais indesejáveis devido a inserção das suas contrapositivas.

fact((mínima_caracterização(barra_princ,-,todos,-) ←
(atuado_alarme(abriu,linha_transmissão_01),atuado_alarme(abriu,linha_tra
nsmissão_02),atuado_alarme(abriu,transformador_01)))).
fact((mínima_caracterização(barra_princ,-,todos,-) ←
(atuado_alarme(abriu,linha_transmissão_01),atuado_alarme(abriu,linha_tra
nsmissão_02),atuado_alarme(abriu,transformador_02)))).
fact((mínima_caracterização(barra_princ,-,todos,-) ←
(atuado_alarme(abriu,linha_transmissão_01),atuado_alarme(abriu,transfor
mador_01),atuado_alarme(abriu,transformador_02)))).
fact((mínima_caracterização(barra_princ,-,todos,-) ←
(atuado_alarme(abriu,linha_transmissão_02),atuado_alarme(abriu,transfor
mador_01),atuado_alarme(abriu,transformador_02)))).

fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,linha_transmissão_01),
atuado_alarme(abriu,linha_transmissão_02))))).
fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,linha_transmissão_01),
atuado_alarme(abriu,transformador_01))))).
fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,linha_transmissão_01),
atuado_alarme(abriu,transformador_02))))).
fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,linha_transmissão_02),
atuado_alarme(abriu,transformador_01))))).
fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,linha_transmissão_02),
atuado_alarme(abriu,transformador_02))))).
fact((mínima_caracterização(barra_princ,-,todos,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,transformador_01),
atuado_alarme(abriu,transformador_02))))).

18) Falhas relacionadas com a Barra Auxiliar :

fact((alr(aux,Comp) ←falha(barra_aux,-,by_pass,Comp))).
fact((alr(abriu,by_pass)← comp_auxiliado_pelo_by_pass(Comp),
falha(barra_aux,-,by_pass,Comp))).
fact((mínima_caracterização(barra_aux,-,by_pass,Comp) ←
(atuado_alarme(aux,Comp),atuado_alarme(abriu,by_pass)))).

19) Falha de Sobrecarga no Transformador 01 :

fact((alr(abriu,transformador_01)←falha(sobre,-,transformador_01,-))).

fact((alr(sobre,transformador_01) ← falha(sobre,-,transformador_01,-))).

fact((not(falha(sobre,-,transformador_01,-)) ←

atuado_alarme(aux,transformador_01))).

Devemos ressaltar que este último fato deveria ser substituido pelo "constraint":

constraint((descrição_falha(sobre,-,transformador_01,-) ←

atuado_alarme(aux,transformador_01))).

mas como este comando não foi implementado pelas versões interpretada e compilada do Theorist, tivemos que descrevê-lo através do fato mostrado acima. Entretanto, devemos destacar que este fato pode ocasionar efeitos colaterais indesejáveis devido a inserção das suas contrapositivas.

fact((mínima_caracterização(sobre,-,Comp,-) ←

(atuado_alarme(sobre,Comp)))).

20) Falha de Sobrecarga no Transformador 02:

fact((alr(abriu,transformador_02) ← falha(sobre,-,transformador_02,-))).

fact((alr(sobre,transformador_02) ← falha(sobre,-,transformador_02,-))).

fact((not(falha(sobre,-,transformador_02,-)) ←

atuado_alarme(aux,transformador_02))).

Devemos ressaltar que este último fato deveria ser substituido pelo "constraint":

constraint((descrição_falha(sobre,-,transformador_02,-) ←

atuado_alarme(aux,transformador_02))).

mas como este comando não foi implementado pelas versões interpretada e compilada do Theorist, tivemos que descrevê-lo através do fato mostrado acima. Entretanto, devemos destacar que este fato pode ocasionar efeitos colaterais indesejáveis devido a inserção das suas contrapositivas.

21) Falha de Sobrecarga no Transformador 01 (3) :

fact((alr(sobre,transformador_01) ← falha(sobre,-

,by_pass,transformador_01))).

fact((alr(aux,transformador_01) ← falha(sobre,-

,by_pass,transformador_01))).

fact((alr(abriu,by_pass) ← falha(sobre,-,by_pass,transformador_01))).

fact((mínima_caracterização(sobre,-,by_pass,Comp) ←

(atuado_alarme(sobre,Comp),atuado_alarme(aux,Comp)))).

22) Falha de Sobrecarga no Transformador 02 (4) :

fact((alr(sobre,transformador_02) ← falha(sobre,-

,by_pass,transformador_02))).

fact((alr(aux,transformador_02) ← falha(sobre,-

,by_pass,transformador_02))).

fact((alr(abriu,by_pass) ← falha(sobre,-,by_pass,transformador_02))).

2) Modificações na Descrição do Sistema com o objetivo de melhorar a sua eficiência :

A seguir, mostraremos duas formas de modificação na descrição que permitiriam melhorar a eficiência do Sistema de Diagnóstico de Falhas utilizando o Theorist :

a) Uma maneira de melhorar a eficiência do sistema seria retirar a condição mínima_caracterização do default descrição_falha e colocá-lo diretamente nos fatos que descrevem os alarmes e suas respectivas falhas. Deste modo, melhorariamos a performance dos testes de consistência dos defaults.

Este mudança pode ser exemplificada pela descrição da falha próxima na linha de transmissão 01 f/f.:

default((falha(Origem, Local, Comp, Comp_aux)).

fact((alr(abriu,linha_transmissão_01) ←
(mínima_caracterização(fase,próximo,linha_transmissão_01,-),
falha(fase,próximo,linha_transmissão_01,-))).

fact((alr(fase,linha_transmissão_01) ←
(mínima_caracterização(fase,próximo,linha_transmissão_01,-),
falha(fase,próximo,linha_transmissão_01,-))).

fact((alr(instantâneo,linha_transmissão_01) ←
(mínima_caracterização(fase,próximo,linha_transmissão_01,-),
falha(fase,próximo,linha_transmissão_01,-))).

fact((mínima_caracterização(fase,próximo,linha_transmissão_01,-) ←
(atuado_alarme(fase,linha_transmissão_01),
atuado_alarme(instantâneo,linha_transmissão_01))).

b) Outra mudança interessante seria utilizar a Lógica Proposicional ao invês da Lógica de 1ª ordem. Deste modo, substituiremos os predicados com argumentos por proposições (constantes) tornando o sistema mais rápido durante as unificações.

Esta mudança pode ser exemplificada pela descrição da falha próxima na linha de transmissão 01 f/f :

default((descrição_falha_próxima_linha_de_transmissão_01_ff :
(falha_próxima_linha_de_transmissão_01_ff ←
mínima_caracterização_falha_próxima_linha_de_transmissão_01_ff))).

fact((alr_abriu_linha_transmissão_01 ←
falha_próxima_linha_de_transmissão_01_ff)).

fact((alr_fase_linha_transmissão_01 ←
falha_próxima_linha_de_transmissão_01_ff)).

fact((alr_instantâneo_linha_transmissão_01 ←

falha_próxima_linha_de_transmissão_01_ff)).

fact((mínima_caracterização_falha_próxima_linha_de_transmissão_01_ff

← (atuado_alarme_fase_linha_transmissão_01,

atuado_alarme_instantâneo_linha_transmissão_01))).

Apêndice E - Implementação do Caso Exemplo utilizando o Theorist

Neste apêndice, apresentaremos o sistema completo para diagnóstico de falhas do caso exemplo utilizando o Sistema Theorist. Devido a sua complexidade, este sistema foi dividido em quatro módulos :

- Achafal → é o módulo principal do sistema. Ele é responsável pela interface com o usuário e pela procura de um explicação para um conjunto de alarmes atuados.
- Descri00, Descri01 e Descri02 → estes módulos apresentam a versão compilada do Theorist para a descrição das falhas apresentadas no apêndice C.

  Devemos destacar que o código gerado por esta transcrição é bastante extenso, não cabendo nos 64k disponíveis para o código do programa. Assim, tivemos de criar três novas áreas de código e com isso, conseguimos gerar uma versão executável.

A seguir, mostraremos a listagem em Prolog de cada um destes programas :

A) ACHAFAL.ARI :

```
/*
    Módulo Achafal :
    Neste módulo, estão declaradas todas as rotinas necessárias para utilizarmos o Theorist em uma
    versão compilada. Além das interfaces com o usuário.
*/


:- module achafal.

/*
    Diretivas para o compilador Prolog :
*/
:- op(1150,fx,fact).
:- op(1150,fx,default).
:- op(1150,fx,predict).
:- op(1150,fx,explain).
:- op(1130,xfx,:).
```

```
:- op(1110,xfx,<-).
:- op(1110,xfx,=>).
:- op(1100,xfy,or).
:- op(1000,xfy,and).
:- op(1000,xfy,&).
:- op(950,fy,not).


:-visible predicado_instanciado/1.
:-visible member /2.

:- public main/0.
:- public member /2.
:- public predicado_instanciado/1.

/*
     member(A, Lista) :
     Verifica se o literal A pertence a Lista.
*/

member(A,[A|_]).

member(A,[X|Resto]) :-
 member(A,Resto).

/*
     append(L1, L2,L3) :
     Cria em L3, uma lista gerada pela união de L1 com L2.
*/
append([],L,L).

append([H|T],L,[H|T1]):-
 append(T,L,T1).

/*
     new_lit(Prefix, Reln, NewArgs, NewReln) :
     Constroe um novo átomo NewReln utilizando os argumentos de Prefix, Reln e NewArgs.
     Exemplo : Ao executarmos new_lit("ex_", father(a,b),[T,A,B], N), teremos N =
     ex_father(a,b,T,A,B).
*/
new_lit(Prefix,Reln,NewArgs,NewReln):-
 Reln =..[Pred|Args],
 name(Pred,PredName),
 append(Prefix,PredName,NewPredName),
 name(NewPred,NewPredName),
 append(Args,NewArgs,AllArgs),
 NewReln =..[NewPred|AllArgs].

/*
     predicado instanciado(Pred) :
     Verifica todos os argumentos do predicado Pred estão instanciados.
*/
predicado_instanciado(Pred):-
 functor((Pred),NomePred,NumVar),
 verifica_variaveis(NumVar,Pred).

/*
     verifica variáveis(NumVar, Pred) :
```

```
    Verifica se a variável de índice NumVar do predicado denominado Pred está instanciada.
*/
verifica_variaveis(0,_).

verifica_variaveis(NumVar,Pred):-
 arg(NumVar,Pred,Var),
 nonvar(Var),
 dec(NumVar,NVar),
 verifica_variaveis(NVar,Pred).

/*
    make bodies(B, T, [Ths, Anc, Ans], ProveB, ExB) :
    Cria o corpo das estruturas de prova e explicação para a fórmula B, onde T é o cenário usado na
    prova de B, Ths é a teoria usada na sua explanação e Anc são os ancestrais usados durante a sua
    prova e explicação.
*/
make_bodies((H,B),T,[ths(T1,T3,D1,D3),Anc,ans(A1,A3)],
        (ProveH,ProveB),(ExH,ExB)):-
 !,
 make_bodies(H,T,[ths(T1,T2,D1,D2),Anc,ans(A1,A2)],ProveH,ExH),
 make_bodies(B,T,[ths(T2,T3,D2,D3),Anc,ans(A2,A3)],ProveB,ExB).

make_bodies((H;B),T,Ths,(ProveH;ProveB),(ExH;ExB)):-
 !,
 make_bodies(H,T,Ths,ProveH,ExH),
 make_bodies(B,T,Ths,ProveB,ExB).

make_bodies(n(A),T,[Ths,Anc,Ans],ProveA,ExA):-
 !,
 new_lit("prove_not_",A,[T,Anc],ProveA),
 new_lit("ex_not_",A,[Ths,Anc,Ans],ExA).

make_bodies(A,T,[Ths,Anc,Ans],ProveA,ExA):-
 new_lit("prove_",A,[T,Anc],ProveA),
 new_lit("ex_",A,[Ths,Anc,Ans],ExA).


/*
    rule(R) :
    Responsável por preparar as estruturas apropriadas para a fórmula R, onde R pode ser um literal
    ou ter a forma if(H,B) onde H é a cabeça da regra e B é o seu corpo.
*/

rule(if(H,B)):-
 !,
 make_anc(H),
 make_bodies(H,T,[Ths,Anc,Ans],ProveH,ExH),
 form_anc(H,Anc,Newanc),
 make_bodies(B,T,[Ths,Newanc,Ans],ProveB,ExB),
 call(prolog_cl((ProveH:-ProveB))),
 call(prolog_cl((ExH:-ExB))).

rule(H):-
 !,
 make_anc(H),
 make_bodies(H,T,[ths(T,T,D,D),_,ans(A,A)],ProveH,ExH),
 call(prolog_cl(ProveH)),
 call(prolog_cl(ExH)).
```

```
/*
    form_anc(L, A1, A2) :
    Cria a nova estrutura de ancestrais A2 a partir do sub-objetivo atual L e da estrutura de ancestrais
    anterior denominada A1.
*/
form_anc(n(G),anc(P,N),anc(P,[G|N])):-!.
form_anc(G,anc(P,N),anc([G|P],N)).

/*
    prolog_cl(C) :
    Responsável por inserir a cláusula C no Prolog.
*/
prolog_cl(X):-
 asserta(X).



/*
    declare_fact(F) :
    Declara que F é um fato do Theorist. Para tal, ele primeiro converte F para a forma negativa
    normal (NNF).
*/
declare_fact(F):-
 nnf(F,even,N),
 rulify(N).

delete_fact(X):-
 recorded(fatos_ins,X,Ref),
 retract(X),
 erase(Ref),fail.

delete_fact(_).

opposite_parity(even,odd).
opposite_parity(odd,even).

/*
    nnf(Wff, Parity, Nnf) :
    Significa que Nnf é a forma negativa normal de :
    • Wff se a Parity for odd
    • Wff se a Parity for even.
*/
nnf((X => Y),P,B):-
 !,
 nnf((Y or not X),P,B).

nnf((Y <- X),P,B):-
 !,
 nnf((Y or not X),P,B).

nnf((X & Y),P,B):-
 !,
 nnf((Y and X),P,B).

nnf((X,Y),P,B):-
 !,
```

```
 nnf((X and Y),P,B).

nnf((X;Y),P,B):-
 !,
 nnf((X or Y),P,B).

nnf((X and Y),P,B):-
 !,
 opposite_parity(P,OP),
 nnf((not X or not Y),OP,B).

nnf((X or Y),even,(XB,YB)):-
 !,
 nnf(X ,even,XB),
 nnf(Y ,even,YB).

nnf((X or Y),odd,(XB,YB)):-
 !,
 nnf(X,odd,XB),
 nnf(Y,odd,YB).
/*
 nnf((~X),P,B):-
 !,
 nnf((not X),P,B).
*/
nnf((not X),P,B):-
 !,
 opposite_parity(P,OP),
 nnf(X ,OP,B).

nnf(F,odd,F).

nnf(n(F),even,F):-!.

nnf(F,even,n(F)).

/*
     rulify(N) :
     Ele é responsável pela criação das estruturas  de explicação e prova da fórmula N no Theorist
*/
rulify((A,B)):-
 !,
 contrapos(B,A),
 contrapos(A,B).

rulify((A;B)):-
 !,
 rulify(A),
 rulify(B).

rulify(n(A)):-
 !,
 rule(A).

rulify(A):-
 rule(n(A)).

/*
```

```
    contrapos(D,T) :
    Gera as contrapositivas da fórmula D ← T.
*/
contrapos(D,(L,R)):-
 !,
 contrapos((R,D),L),
 contrapos((L,D),R).

contrapos(D,(L;R)):-
 !,
 contrapos(D,L),
 contrapos(D,R).

contrapos(D,n(A)):-
 !,
 rule(if(A,D)).

contrapos(D,A):-
 rule(if(n(A),D)).

/*
    declare default (D) :
    Declara que D é um default para o Theorist.
*/
declare_default(D):-
 make_anc(D),
 new_lit("prove_",D,[T,_],Pr_D),
 call(prolog_cl((Pr_D:-member(D,T)))),
 new_lit("ex_",D,[ths(T,T,Defer,Defer),_,ans(A,A)],ExD),
 call(prolog_cl((ExD:-member(D,T)))),
 new_lit("ex_",D,[ths(T,[D|T],Defer,Defer),_,ans(A,A)],ExDass),
 new_lit("prove_not_",D,[[D|T],anc([],[])],Pr_not_D),
 call(prolog_cl((ExDass:-predicado_instanciado(D),\+ member(D,T),
        \+ Pr_not_D))),
 new_lit("ex_",D,[ths(T,T,Defer,[D|Defer]),_,ans(A,A)],ExDefer),
 call(prolog_cl((ExDefer:- \+ predicado_instanciado(D)))).

same_length([],[]).

same_length([H|T],Nargs):-
 same_length(T,Narg),
 Nargs = [ X | Narg].

/*
    make anc(Nome) :
    Cria as assertivas para verificar se o fato Nome já foi usado anteriormente na busca de uma prova
    ou explicação. Este predicado cria os mecanismos necessários para a prova por absurdo
*/
make_anc(Name):-
 call(ancestor_recorded(Name)),
 !.

make_anc(n(Goal)):-
 !,
 make_anc(Goal).

make_anc(Goal):-
 Goal =.. [Pred|Args],
```

```
 same_length(Args,Nargs),
 NG =..[Pred|Nargs],
 make_bodies(NG,_,[ths(T,T,D,D),anc(P,N),ans(A,A)],ProveG,ExG),
 make_bodies(n(NG),_,[ths(T,T,D,D),anc(P,N),ans(A,A)],ProvenG,ExnG),
 call(prolog_cl((ProveG:-member(NG,N)))),
 call(prolog_cl((ProvenG:- member(NG,P)))),
 call(prolog_cl((ExG:-member(NG,N)))),
 call(prolog_cl((ExnG:-member(NG,P)))),
 call(prolog_cl(ancestor_recorded(NG))).

numbervars('$VAR'(N),N,N1):-
 inc(N,N1).

numbervars(Term,N1,N2):-
 nonvar(Term),
 functor(Term,Name,N),
 numbervars(0,N,Term,N1,N2).

numbervars(N,N,Term,N1,N1).

numbervars(I,N,Term,N1,N3):-
 I < N,
 inc(I,I1),
 arg(I1,Term,Arg),
 numbervars(Arg,N1,N2),
 numbervars(I1,N,Term,N2,N3).

/*
     ground(Term) :
     Instancia as variáveis livres de Term com constantes únicas.
*/
ground(Term):-
 numbervars(Term,0,N).


/*
     expl(G, T0, T1, Ans) :
     Implementa a função explanação do Theorist, ou seja, ele tenta explicar a fórmula G a paritr da
     Teoria inical T0, gerando a nova teoria T1
*/
expl(G,T0,T1,Ans):-
 ground(N),
 declare_fact( ('<-'(newans(N,G),G))),
 call(ex_newans(N,G,ths(T0,T,[],D),anc([],[]),ans(G,Ans))),
 ground(D),
 check_consist(D,T,T1),
 delete_fact(X).

/*
     check consist(D, T, T1) :
     Checa a consistência do conjunto de defaults D utilizados na explicação da fórmula G.
*/
check_consist([],T,T).

check_consist([H|D],T1,T):-
 new_lit("prove_not_",H,[T1,anc([],[])],Pr_n_H),
 \+ Pr_n_H,
 check_consist(D,[H|T1],T).
```

```
/*
    fact(Fato) :
    Implementa a declaração Fato do Theorist onde Fato é uma fórmula bem formada.
*/
fact(F):-
 declare_fact(F),
 !.

/*
    default(N:H) :
    Implementa a declaração default do Theorist onde H é uma possível hipótese com nome N.
*/
default((N:H)):-
 !,
 declare_default(N),
 declare_fact((H '<-' N)),
 !.

/*
    default(N) :
    Implementa a declaração default do Theorist onde N é o nome de uma possível hipótese.
*/
default(N):-
 declare_default(N),
 !.

/*
    explainall(G) :
    Representa o conceito explanação do Theorist onde G é a fómula que deve ser provada.
*/
explainall(Explicacao):-
 expl(Explicacao,[],D,Ans),
 write(D),nl,
 write(Ans),nl,
fail.

explainall(_).

/*
    diag(Lista de Alarmes, Explicacao) :
    Procura uma explicação para um conjunto de alarmes atuados.
*/
diag([],[]).

diag([A|B],Exp):-
/* Pede explicação para o alarme A */
 expl(A,[],D,Ans),
 diag(B,Expli),
/* Acescenta novas falhas a explicação */
 append(D,Expli,Exp).

/*
    explique alarmes(Lista de Alarmes) :
    Procura todas as explicações possíveis para os alarmes.
*/
explique_alarmes(Observacao):-
 diag(Observacao,Explicacao),
```

```
 /* Verifica se esta pode ser aceita */
 insere_explicacao(Explicacao),
 /* Incrementa o numero de explicacoes validas */
 ctr_inc(1,_),
 fail.

explique_alarmes(_).

/*
     insere explicacao(Explicacao) :
     Este predicado insere uma Explicacao na base de dados desde que esta não esteja inconsistente ou
     esta já  não exista na base de dados.
*/
insere_explicacao(Explicacao):-
  /* Ordena Lista */
  sort(Explicacao,Explic),
  /* Insere a explicacao na base de dados */
  insere_na_lista(Explic).

/*
     insere na lista(Lista) :
     Insere Lista na base de dados desde que esta já  não exista.
*/
insere_na_lista(Lista):-
  /* Verifica se elemento nao esta na lista */
  not(recorded(exp,Lista,_)),
  /* Salva conteudo na base de dados */
  recordz(exp,Lista,_).

/*
     insere observacoes(Lista) :
     Insere todos os alarmes observados como fact(obs_arl(Tipo,Componente)).
*/

insere_observacoes([]).

insere_observacoes([alr(X,Y)|B]):-
 fact(obs_alr(X,Y)),
 insere_observacoes(B).

/*
     origem componente(Origem,Componente) :
     O predicado origem_componente fornece os tipos de alarme que caracterizam a Origem de uma
     falha de acordo com o Componente especificado.
*/
origem_componente(fase,Componente)  :- e_linha_transmissao(Componente).
origem_componente(fase,Componente)  :- e_by_pass(Componente).
origem_componente(terra,Componente) :- e_linha_transmissao(Componente).
origem_componente(terra,Componente) :- e_by_pass(Componente).
origem_componente(sobre,Componente) :- e_transformador(Componente).


/*
     local componente(Tipo,Local,Componente) :
     O predicado local_componente fornece o Tipo de alarme que caracteriza o local de uma falha de
     acordo com o Componente especificado.
*/
local_componente(inst,proximo,Componente)  :- e_linha_transmissao (Compo nente).
```

```
local_componente(inst,proximo,Componente)  :- e_by_pass(Componente).
local_componente(temp,distante,Componente) :- e_linha_transmissao(Compo nente).
local_componente(temp,distante,Componente) :- e_by_pass(Componente).

/*
    tipo componente(Tipo,Componente) :
    O predicado tipo_componente fornece todas as comnbinacções possíveis de descrição de alarme.
 */
tipo_componente(Tipo,Componente)  :- local_componente(Tipo,_,Componente).
tipo_componente(Origem,Componente)  :- origem_componente(Origem,Componente).
tipo_componente(abriu,Componente) :- e_componente(Componente).
tipo_componente(aux,Componente)   :- nao_e_by_pass(Componente).

/*
Descrição dos Componentes :

Tipos de componentes possiveis : Linha de tranmissao, Transformador e disjuntor de  by-pass.
No  caso exemplo, existem  dois linha de tranmissaoes (lt_01 e lt_02 ), dois transformadores
(transformador_01 e transformador_02) e um disjuntor de by-pass(by_pass).
*/

e_linha_transmissao(lt_01).
e_linha_transmissao(lt_02).
e_transformador(transf_01).
e_transformador(transf_02).
e_by_pass(by_pass).


e_componente(X):- e_linha_transmissao(X).
e_componente(X):- e_transformador(X).
e_componente(X):- e_by_pass(X).

nao_e_by_pass(X):- e_linha_transmissao(X).
nao_e_by_pass(X):- e_transformador(X).


/*
    pede alarmes observados(Lista) :
    Este predicado pergunta quais os alarmes que foram observados e os coloca em Lista.
*/
pede_alarmes_observados(Lista):-
 tmove(6,0),
 write($ Descreva os alarmes atuados : $),
 nl,
 write($ -------- -- ------- ------- $),
 /* Limpa parte da tela */
 tscroll(9,(8,0),(17,79)),
 tmove(9,0),
 /* Pede componente */
 write($     Componente : $),
 read_string(99,B),
 nl,
 nl,
 /* Pede tipo de atuacao */
 write($     Tipo de Atuacao : $),
 read_string(99,A),
 nl,
 (
```

```
 /* Se algum do dois estiver vazio, fim do pedido de alarmes */
 (A = $$ ; B = $$), Lista =[], !;
 /* Caso contrario, verifica se o alarme e' consistente  e pede novo alarme */
 string_term(A,Tipo),
 string_term(B,Componente),
 (tipo_componente(Tipo,Componente),
  pede_alarmes_observados(Lista1),
  /* Acrescenta alarme na lista */
  Lista = [alr(Tipo,Componente)|Lista1];
  put(7),
  tmove(14,0),
  write($ Este alarme nao existe, digite qualquer tecla para continuar.$),
  keyb(F,G),
  pede_alarmes_observados(Lista))).

/*
     apresenta tela comandos :
     Este predicado é responsável por apresentar todos os comandos disponíveis  para acessar as
     explicações na tela.
*/
apresenta_tela_comandos:-
 tmove(23,0),
 write($ Comandos :  PgUp - Proxima diagnose   PgDn -  Diagnose Anterior $),
 nl,
 write($ Digite qualquer outra tecla para sair do programa.  $).

/*
     escreve explicacao :
     Este predicado apresenta na tela todo o conteúdo de uma explicação.
*/
escreve_explicacao(Lista_Alarmes):-
 /* Procura explicacao na base de dados */
 ctr_is(0,Ind),
 nth_ref(exp,Ind,Ref),
 instance(Ref,Lista_Explicacao),
 /* Limpa parte da tela */
 tscroll(16,(6,0),(22,79)),
 tmove(7,0),
 write($ Explicacao $),
 /* Mostra o indice da explicacao */
 write(Ind),
 write($ : $),
 nl,
 write($ ----------$),
 nl,
 nl,
 write($ Falhas Possiveis : $),
 nl,
 imprime_conteudo_explicacao(Lista_Explicacao).

/*
     imprime lista de alarmes(Lista) :
     Este predicado imprime a Lista de alarmes utilizados em uma explicação.
*/
imprime_lista_de_alarmes([]).

imprime_lista_de_alarmes([A|B]):-
 write(A),
```

```
write($ $),
imprime_lista_de_alarmes(B).

/*
    imprime_explicacoes(Lista_Alarmes,Num_explicacoes) :
    Este predicado tem como função controlar a apresentação na tela das explicações possíveis para a
    Lista_Alarmes.
    Caso Num_explicacoes seja nulo, ele indicara que nao foi possível encontrar uma explicação.
*/
imprime_explicacoes(Lista_alarmes,0):-
 /* Nao existe nenhum explicacao */
 !,
 escreve_titulo,
 tmove(6,0),
 write($ Nao existe nenhuma diagnose para configuracao abaixo :$),
 nl,
 nl,
 write($ Alarmes Atuados : $),
 nl,
 write($  $),
 imprime_lista_de_alarmes(Lista_alarmes),
 /* Espera que alguma tecla seja digitada */
 keyb(X,Y).

imprime_explicacoes(Lista_alarmes,Num_explicacoes):-
  escreve_titulo,
  tmove(5,0),
  write($ Numero total de explicacoes : $),
  write(Num_explicacoes),
  nl,
  apresenta_tela_comandos,
  ctr_set(0,1),
  escreve_explicacao(Lista_alarmes),
  le_lista_de_explicacoes(Lista_alarmes,Num_explicacoes).


/*
    acerta_indice(Indice) :
    Este predicado tem como acertar o índice da explicação apresentada quando o limite for
    ultrapassado.
*/
acerta_indice(Indice):-
 put(7),
 ctr_set(0,Indice).

/*
    procura_nova_explicacao(Tam_lista,Flag) :
    Este predicado tem como funcao procurar um novo explicacao para ser apresentada conforme a
    tecla digitada pelo usuario.
    • Tecla pgup → procura proxima explicacao,
    • Tecla pgdn → procura explicacao anterior,
    • Quaquer outra tecla → Flag = fim.
*/
procura_nova_explicacao(Tam_lista,Flag):-
 keyb(X,Y),
 (
 /* Teclou PgUp */
 X=0,Y=73,ctr_inc(0,Ind),
```

```
/* Se fim da lista -> volta ao indice anterior */
ifthen(Ind @>= Tam_lista,acerta_indice(Ind));
/* Teclou PgDn */
X=0,Y=81,ctr_dec(0,Ind),
/* Se inicio da lista -> volta ao indice anterior */
ifthen(Ind @=< 1,acerta_indice(Ind));
/* Qualquer outra tecla -> fim das explicacoes */
(X\=0;X=0,Y\=73,Y\=81),
Flag = fim).

/*
    le_lista_de_explicacoes(Lista_alarmes,Tam_lista) :
    Este predicado apresenta as explicações na tela conforme a tecla digitada pelo usuario.
*/
le_lista_de_explicacoes(Lista_alarmes,Tam_lista):-
 repeat,
 procura_nova_explicacao(Tam_lista,Flag),
 escreve_explicacao(Lista_alarmes),
 Flag == fim.

/*
    imprime_conteudo_explicacao :
    Este predicado apresenta todo o conteúdo da explicação na tela.
*/

imprime_conteudo_explicacao([]).

imprime_conteudo_explicacao([A|B]):-
 A = desc(X,Y,Z,W),
 tget(Linha,Coluna),
 write($ Origem: $),
 write(X),
 Coluna1 is Coluna + 21,
 tmove(Linha,Coluna1),
 write($ Local: $),
 write(Y),
 Coluna2 is Coluna1 + 18,
 tmove(Linha,Coluna2),
 write($ Comp: $),
 write(Z),
 Coluna2 is Coluna1 + 18,
 write($ Aux: $),
 write(W),
 nl,
 imprime_conteudo_explicacao(B).


/*****************************************************************************

                    PROGRAMA PRINCIPAL

*****************************************************************************/

main :-
escreve_titulo,
asserta(( prolog_cl(X):- asserta(X))),/*write(X))*/
pede_alarmes_observados(Lista_Alarmes),
/* Transforma os alarmes observdos em fatos para o Theorist */
```

```
Lista_Alarmes = [alr(X1,Y1)|B],
marca_anc(obs_alr(X1,Y1)),
insere_observacoes(Lista_Alarmes),
abolish(prolog_cl/1),
asserta((prolog_cl(X):- asserta(X), recordz(fatos_ins,X,_))),
 /* Zera o numero de explicacoes validas */
 ctr_set(1,0),
 /* Procura por todas as explicacoes de falhas possiveis para os alarmes obser-
    vados */
 explique_alarmes(Lista_Alarmes),
  /* Descobre o numero total de explicacoes */
 ctr_is(1,Num_Explicacoes),
 /* Apresenta na tela as explicacoes */
 imprime_explicacoes(Lista_Alarmes,Num_Explicacoes),
 cls,
 eraseall(exp),
 abolish(prolog_cl/1).
```

B) DESCRI00.ARI :

```
:-segment(farseg1).

:- module descri00.

:- visible prove_alr / 4.
:- visible prove_not_alr / 4.
:- visible ex_alr / 5.
:-visible ex_not_alr / 5.
:- visible prove_falha / 6.
:- visible prove_not_falha / 6.
:- visible ex_falha / 7.


:- public prove_alr / 4.
:- public prove_not_alr / 4.
:- public ex_alr / 5.
:- public ex_not_alr / 5.
:- public prove_falha / 6.
:- public prove_not_falha / 6.
:- public ex_falha / 7.


:- extrn member /2:far.
:- extrn predicado_instanciado/1:far.
:- extrn prove_comp_auxiliado_pelo_by_pass / 3:far.
:- extrn ex_comp_auxiliado_pelo_by_pass / 4:far.
:- extrn prove_desc / 6:far.
:- extrn ex_desc / 7:far.
:- extrn prove_atuado_alarme / 4:far.
:- extrn prove_minima_caracterizacao / 6:far.
:- extrn ex_minima_caracterizacao / 7:far.


prove_alr(A,B,C,anc(D,E)) :-
   member(alr(A,B),E).
prove_alr(abriu,lt_01,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_01,-,A,anc([alr(abriu,lt_01)|B],C)).
prove_alr(terra,lt_01,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_01,-,A,anc([alr(terra,lt_01)|B],C)).
prove_alr(inst,lt_01,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_01,-,A,anc([alr(inst,lt_01)|B],C)).
prove_alr(abriu,lt_01,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_01,-,A,anc([alr(abriu,lt_01)|B],C)).
prove_alr(fase,lt_01,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_01,-,A,anc([alr(fase,lt_01)|B],C)).
prove_alr(inst,lt_01,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_01,-,A,anc([alr(inst,lt_01)|B],C)).
prove_alr(abriu,lt_01,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_01,-,A,anc([alr(abriu,lt_01)|B],C)).
prove_alr(fase,lt_01,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_01,-,A,anc([alr(fase,lt_01)|B],C)).
prove_alr(temp,lt_01,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_01,-,A,anc([alr(temp,lt_01)|B],C)).
```

```
prove_alr(abriu,lt_01,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_01,-,A,anc([alr(abriu,lt_01)|B],C)).
prove_alr(terra,lt_01,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_01,-,A,anc([alr(terra,lt_01)|B],C)).
prove_alr(temp,lt_01,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_01,-,A,anc([alr(temp,lt_01)|B],C)).
prove_alr(abriu,lt_02,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_02,-,A,anc([alr(abriu,lt_02)|B],C)).
prove_alr(terra,lt_02,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_02,-,A,anc([alr(terra,lt_02)|B],C)).
prove_alr(inst,lt_02,A,anc(B,C)) :-
   prove_falha(terra,proximo,lt_02,-,A,anc([alr(inst,lt_02)|B],C)).
prove_alr(abriu,lt_02,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_02,-,A,anc([alr(abriu,lt_02)|B],C)).
prove_alr(fase,lt_02,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_02,-,A,anc([alr(fase,lt_02)|B],C)).
prove_alr(inst,lt_02,A,anc(B,C)) :-
   prove_falha(fase,proximo,lt_02,-,A,anc([alr(inst,lt_02)|B],C)).
prove_alr(abriu,lt_02,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_02,-,A,anc([alr(abriu,lt_02)|B],C)).
prove_alr(fase,lt_02,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_02,-,A,anc([alr(fase,lt_02)|B],C)).
prove_alr(temp,lt_02,A,anc(B,C)) :-
   prove_falha(fase,distante,lt_02,-,A,anc([alr(temp,lt_02)|B],C)).
prove_alr(abriu,lt_02,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_02,-,A,anc([alr(abriu,lt_02)|B],C)).
prove_alr(terra,lt_02,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_02,-,A,anc([alr(terra,lt_02)|B],C)).
prove_alr(temp,lt_02,A,anc(B,C)) :-
   prove_falha(terra,distante,lt_02,-,A,anc([alr(temp,lt_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(fase,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_01,A,anc([alr(fase,by_pass)|B],C)).
prove_alr(fase,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_02,A,anc([alr(fase,by_pass)|B],C)).
prove_alr(inst,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_01,A,anc([alr(inst,by_pass)|B],C)).
prove_alr(inst,by_pass,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_02,A,anc([alr(inst,by_pass)|B],C)).
prove_alr(aux,lt_01,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C)).
prove_alr(aux,lt_02,A,anc(B,C)) :-
   prove_falha(fase,proximo,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(terra,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_01,A,anc([alr(terra,by_pass)|B],C)).
prove_alr(terra,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_02,A,anc([alr(terra,by_pass)|B],C)).
prove_alr(inst,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_01,A,anc([alr(inst,by_pass)|B],C)).
prove_alr(inst,by_pass,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_02,A,anc([alr(inst,by_pass)|B],C)).
```

```
prove_alr(aux,lt_01,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C)).
prove_alr(aux,lt_02,A,anc(B,C)) :-
   prove_falha(terra,proximo,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(fase,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_01,A,anc([alr(fase,by_pass)|B],C)).
prove_alr(fase,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_02,A,anc([alr(fase,by_pass)|B],C)).
prove_alr(temp,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_01,A,anc([alr(temp,by_pass)|B],C)).
prove_alr(temp,by_pass,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_02,A,anc([alr(temp,by_pass)|B],C)).
prove_alr(aux,lt_01,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C)).
prove_alr(aux,lt_02,A,anc(B,C)) :-
   prove_falha(fase,distante,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(terra,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_01,A,anc([alr(terra,by_pass)|B],C)).
prove_alr(terra,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_02,A,anc([alr(terra,by_pass)|B],C)).
prove_alr(temp,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_01,A,anc([alr(temp,by_pass)|B],C)).
prove_alr(temp,by_pass,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_02,A,anc([alr(temp,by_pass)|B],C)).
prove_alr(aux,lt_01,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C)).
prove_alr(aux,lt_02,A,anc(B,C)) :-
   prove_falha(terra,distante,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C)).
prove_alr(abriu,lt_01,A,anc(B,C)) :-
   prove_falha(barra_princ,-,todos,D,A,anc([alr(abriu,lt_01)|B],C)).
prove_alr(abriu,lt_02,A,anc(B,C)) :-
   prove_falha(barra_princ,-,todos,D,A,anc([alr(abriu,lt_02)|B],C)).
prove_alr(abriu,transf_01,A,anc(B,C)) :-
   prove_falha(barra_princ,-,todos,D,A,anc([alr(abriu,transf_01)|B],C)).
prove_alr(abriu,transf_02,A,anc(B,C)) :-
   prove_falha(barra_princ,-,todos,D,A,anc([alr(abriu,transf_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_comp_auxiliado_pelo_by_pass(D,A,anc([alr(abriu,by_pass)|B],C)),
   prove_falha(barra_princ,-,todos,D,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(aux,A,B,anc(C,D)) :-
   prove_falha(barra_princ,-,todos,A,B,anc([alr(aux,A)|C],D)).
prove_alr(aux,A,B,anc(C,D)) :-
   prove_falha(barra_aux,-,by_pass,A,B,anc([alr(aux,A)|C],D)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_comp_auxiliado_pelo_by_pass(D,A,anc([alr(abriu,by_pass)|B],C)),
   prove_falha(barra_aux,-,by_pass,D,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,transf_01,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_01,-,A,anc([alr(abriu,transf_01)|B],C)).
prove_alr(sobre,transf_01,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_01,-,A,anc([alr(sobre,transf_01)|B],C)).
```

```
prove_alr(abriu,transf_02,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_02,-,A,anc([alr(abriu,transf_02)|B],C)).
prove_alr(sobre,transf_02,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_02,-,A,anc([alr(sobre,transf_02)|B],C)).
prove_alr(sobre,transf_01,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_01,A,anc([alr(sobre,transf_01)|B],C)).
prove_alr(sobre,transf_02,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_02,A,anc([alr(sobre,transf_02)|B],C)).
prove_alr(aux,transf_01,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_01,A,anc([alr(aux,transf_01)|B],C)).
prove_alr(aux,transf_02,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_02,A,anc([alr(aux,transf_02)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_01,A,anc([alr(abriu,by_pass)|B],C)).
prove_alr(abriu,by_pass,A,anc(B,C)) :-
   prove_falha(sobre,-,by_pass,transf_02,A,anc([alr(abriu,by_pass)|B],C)).


prove_not_alr(A,B,C,anc(D,E)) :-
   member(alr(A,B),D).


ex_alr(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(alr(A,B),F).
ex_alr(abriu,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_01,-,A,anc([alr(abriu,lt_01)|B],C),D).
ex_alr(terra,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_01,-,A,anc([alr(terra,lt_01)|B],C),D).
ex_alr(inst,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_01,-,A,anc([alr(inst,lt_01)|B],C),D).
ex_alr(abriu,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_01,-,A,anc([alr(abriu,lt_01)|B],C),D).
ex_alr(fase,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_01,-,A,anc([alr(fase,lt_01)|B],C),D).
ex_alr(inst,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_01,-,A,anc([alr(inst,lt_01)|B],C),D).
ex_alr(abriu,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_01,-,A,anc([alr(abriu,lt_01)|B],C),D).
ex_alr(fase,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_01,-,A,anc([alr(fase,lt_01)|B],C),D).
ex_alr(temp,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_01,-,A,anc([alr(temp,lt_01)|B],C),D).
ex_alr(abriu,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_01,-,A,anc([alr(abriu,lt_01)|B],C),D).
ex_alr(terra,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_01,-,A,anc([alr(terra,lt_01)|B],C),D).
ex_alr(temp,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_01,-,A,anc([alr(temp,lt_01)|B],C),D).
ex_alr(abriu,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_02,-,A,anc([alr(abriu,lt_02)|B],C),D).
ex_alr(terra,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_02,-,A,anc([alr(terra,lt_02)|B],C),D).
ex_alr(inst,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,proximo,lt_02,-,A,anc([alr(inst,lt_02)|B],C),D).
ex_alr(abriu,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_02,-,A,anc([alr(abriu,lt_02)|B],C),D).
ex_alr(fase,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_02,-,A,anc([alr(fase,lt_02)|B],C),D).
```

```
ex_alr(inst,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,proximo,lt_02,-,A,anc([alr(inst,lt_02)|B],C),D).
ex_alr(abriu,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_02,-,A,anc([alr(abriu,lt_02)|B],C),D).
ex_alr(fase,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_02,-,A,anc([alr(fase,lt_02)|B],C),D).
ex_alr(temp,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,distante,lt_02,-,A,anc([alr(temp,lt_02)|B],C),D).
ex_alr(abriu,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_02,-,A,anc([alr(abriu,lt_02)|B],C),D).
ex_alr(terra,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_02,-,A,anc([alr(terra,lt_02)|B],C),D).
ex_alr(temp,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,distante,lt_02,-,A,anc([alr(temp,lt_02)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(fase,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_01,A,anc([alr(fase,by_pass)|B],C),D).
ex_alr(fase,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_02,A,anc([alr(fase,by_pass)|B],C),D).
ex_alr(inst,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_01,A,anc([alr(inst,by_pass)|B],C),D).
ex_alr(inst,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_02,A,anc([alr(inst,by_pass)|B],C),D).
ex_alr(aux,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C),D).
ex_alr(aux,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,proximo,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(terra,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_01,A,anc([alr(terra,by_pass)|B],C),D).
ex_alr(terra,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_02,A,anc([alr(terra,by_pass)|B],C),D).
ex_alr(inst,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_01,A,anc([alr(inst,by_pass)|B],C),D).
ex_alr(inst,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_02,A,anc([alr(inst,by_pass)|B],C),D).
ex_alr(aux,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C),D).
ex_alr(aux,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,proximo,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(fase,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_01,A,anc([alr(fase,by_pass)|B],C),D).
ex_alr(fase,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_02,A,anc([alr(fase,by_pass)|B],C),D).
ex_alr(temp,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_01,A,anc([alr(temp,by_pass)|B],C),D).
ex_alr(temp,by_pass,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_02,A,anc([alr(temp,by_pass)|B],C),D).
```

```
ex_alr(aux,lt_01,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C),D).
ex_alr(aux,lt_02,A,anc(B,C),D) :-
   ex_falha(fase,distante,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_01,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_02,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(terra,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_01,A,anc([alr(terra,by_pass)|B],C),D).
ex_alr(terra,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_02,A,anc([alr(terra,by_pass)|B],C),D).
ex_alr(temp,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_01,A,anc([alr(temp,by_pass)|B],C),D).
ex_alr(temp,by_pass,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_02,A,anc([alr(temp,by_pass)|B],C),D).
ex_alr(aux,lt_01,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_01,A,anc([alr(aux,lt_01)|B],C),D).
ex_alr(aux,lt_02,A,anc(B,C),D) :-
   ex_falha(terra,distante,by_pass,lt_02,A,anc([alr(aux,lt_02)|B],C),D).
ex_alr(abriu,lt_01,A,anc(B,C),D) :-
   ex_falha(barra_princ,-,todos,E,A,anc([alr(abriu,lt_01)|B],C),D).
ex_alr(abriu,lt_02,A,anc(B,C),D) :-
   ex_falha(barra_princ,-,todos,E,A,anc([alr(abriu,lt_02)|B],C),D).
ex_alr(abriu,transf_01,A,anc(B,C),D) :-
   ex_falha(barra_princ,-,todos,E,A,anc([alr(abriu,transf_01)|B],C),D).
ex_alr(abriu,transf_02,A,anc(B,C),D) :-
   ex_falha(barra_princ,-,todos,E,A,anc([alr(abriu,transf_02)|B],C),D).
ex_alr(abriu,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_comp_auxiliado_pelo_by_pass(I,ths(A,J,C,K),anc([alr(abriu,by_pass)|E],F),ans(G,L)),
   ex_falha(barra_princ,-,todos,I,ths(J,B,K,D),anc([alr(abriu,by_pass)|E],F),ans(L,H)).
ex_alr(aux,A,B,anc(C,D),E) :-
   ex_falha(barra_princ,-,todos,A,B,anc([alr(aux,A)|C],D),E).
ex_alr(aux,A,B,anc(C,D),E) :-
   ex_falha(barra_aux,-,by_pass,A,B,anc([alr(aux,A)|C],D),E).
ex_alr(abriu,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_comp_auxiliado_pelo_by_pass(I,ths(A,J,C,K),anc([alr(abriu,by_pass)|E],F),ans(G,L)),
   ex_falha(barra_aux,-,by_pass,I,ths(J,B,K,D),anc([alr(abriu,by_pass)|E],F),ans(L,H)).
ex_alr(abriu,transf_01,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_01,-,A,anc([alr(abriu,transf_01)|B],C),D).
ex_alr(sobre,transf_01,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_01,-,A,anc([alr(sobre,transf_01)|B],C),D).
ex_alr(abriu,transf_02,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_02,-,A,anc([alr(abriu,transf_02)|B],C),D).
ex_alr(sobre,transf_02,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_02,-,A,anc([alr(sobre,transf_02)|B],C),D).
ex_alr(sobre,transf_01,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_01,A,anc([alr(sobre,transf_01)|B],C),D).
ex_alr(sobre,transf_02,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_02,A,anc([alr(sobre,transf_02)|B],C),D).
ex_alr(aux,transf_01,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_01,A,anc([alr(aux,transf_01)|B],C),D).
ex_alr(aux,transf_02,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_02,A,anc([alr(aux,transf_02)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_01,A,anc([alr(abriu,by_pass)|B],C),D).
ex_alr(abriu,by_pass,A,anc(B,C),D) :-
   ex_falha(sobre,-,by_pass,transf_02,A,anc([alr(abriu,by_pass)|B],C),D).
```

```
ex_not_alr(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(alr(A,B),E).


prove_falha(A,B,C,D,E,anc(F,G)) :-
   member(falha(A,B,C,D),G).
prove_falha(A,B,C,D,E,anc(F,G)) :-
   prove_minima_caracterizacao(A,B,C,D,E,anc([falha(A,B,C,D)|F],G)),
   prove_desc(A,B,C,D,E,anc([falha(A,B,C,D)|F],G)).


prove_not_falha(A,B,C,D,E,anc(F,G)) :-
   member(falha(A,B,C,D),F).
prove_not_falha(terra,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C])).
prove_not_falha(terra,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(terra,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C])).
prove_not_falha(terra,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(inst,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C])).
prove_not_falha(fase,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C])).
prove_not_falha(fase,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(fase,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C])).
prove_not_falha(fase,proximo,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(inst,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C])).
prove_not_falha(fase,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C])).
prove_not_falha(fase,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(fase,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C])).
prove_not_falha(fase,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(temp,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C])).
prove_not_falha(terra,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C])).
prove_not_falha(terra,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(terra,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C])).
prove_not_falha(terra,distante,lt_01,-,A,anc(B,C)) :-
   prove_not_alr(temp,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C])).
prove_not_falha(terra,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C])).
prove_not_falha(terra,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(terra,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C])).
prove_not_falha(terra,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(inst,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C])).
prove_not_falha(fase,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C])).
prove_not_falha(fase,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(fase,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C])).
prove_not_falha(fase,proximo,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(inst,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C])).
prove_not_falha(fase,distante,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(abriu,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C])).
prove_not_falha(fase,distante,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(fase,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C])).
prove_not_falha(fase,distante,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(temp,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C])).
prove_not_falha(terra,distante,lt_02,-,A,anc(B,C)) :-
```

```
   prove_not_alr(abriu,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C])).
prove_not_falha(terra,distante,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(terra,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C])).
prove_not_falha(terra,distante,lt_02,-,A,anc(B,C)) :-
   prove_not_alr(temp,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C])).
prove_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C])).
prove_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C])).
prove_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(fase,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C])).
prove_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(fase,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C])).
prove_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(inst,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C])).
prove_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(inst,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C])).
prove_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(aux,lt_01,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C])).
prove_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(aux,lt_02,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C])).
prove_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C])).
prove_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C])).
prove_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(terra,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C])).
prove_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(terra,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C])).
prove_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(inst,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C])).
prove_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(inst,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C])).
prove_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(aux,lt_01,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C])).
prove_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(aux,lt_02,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C])).
prove_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C])).
prove_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C])).
prove_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(fase,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C])).
prove_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(fase,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C])).
prove_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(temp,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C])).
prove_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(temp,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C])).
prove_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(aux,lt_01,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C])).
prove_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(aux,lt_02,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C])).
prove_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C])).
prove_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C])).
prove_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C)) :-
```

```
   prove_not_alr(terra,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C])).
prove_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(terra,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C])).
prove_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(temp,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C])).
prove_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(temp,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C])).
prove_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C)) :-
   prove_not_alr(aux,lt_01,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C])).
prove_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C)) :-
   prove_not_alr(aux,lt_02,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_not_alr(abriu,lt_01,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_not_alr(abriu,lt_02,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_not_alr(abriu,transf_01,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_not_alr(abriu,transf_02,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_comp_auxiliado_pelo_by_pass(A,B,anc(C,[falha(barra_princ,-,todos,A)|D])),
   prove_not_alr(abriu,by_pass,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_not_alr(aux,A,B,anc(C,[falha(barra_princ,-,todos,A)|D])).
prove_not_falha(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(aux,D,A,anc(B,[falha(barra_princ,-,todos,-)|C])).
prove_not_falha(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(D,by_pass,A,anc(B,[falha(barra_princ,-,todos,-)|C])).
prove_not_falha(barra_aux,-,by_pass,A,B,anc(C,D)) :-
   prove_not_alr(aux,A,B,anc(C,[falha(barra_aux,-,by_pass,A)|D])).
prove_not_falha(barra_aux,-,by_pass,A,B,anc(C,D)) :-
   prove_comp_auxiliado_pelo_by_pass(A,B,anc(C,[falha(barra_aux,-,by_pass,A)|D])),
   prove_not_alr(abriu,by_pass,B,anc(C,[falha(barra_aux,-,by_pass,A)|D])).
prove_not_falha(sobre,-,transf_01,-,A,anc(B,C)) :-
   prove_not_alr(abriu,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C])).
prove_not_falha(sobre,-,transf_01,-,A,anc(B,C)) :-
   prove_not_alr(sobre,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C])).
prove_not_falha(sobre,-,transf_02,-,A,anc(B,C)) :-
   prove_not_alr(abriu,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C])).
prove_not_falha(sobre,-,transf_02,-,A,anc(B,C)) :-
   prove_not_alr(sobre,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C])).
prove_not_falha(sobre,-,transf_01,-,A,anc(B,C)) :-
   prove_atuado_alarme(aux,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C])).
prove_not_falha(sobre,-,transf_02,-,A,anc(B,C)) :-
   prove_atuado_alarme(aux,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C])).
prove_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C)) :-
   prove_not_alr(sobre,transf_01,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C])).
prove_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C)) :-
   prove_not_alr(sobre,transf_02,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C])).
prove_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C)) :-
   prove_not_alr(aux,transf_01,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C])).
prove_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C)) :-
   prove_not_alr(aux,transf_02,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C])).
prove_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C])).
prove_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C)) :-
   prove_not_alr(abriu,by_pass,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C])).
```

```
ex_falha(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(falha(A,B,C,D),H).
ex_falha(A,B,C,D,ths(E,F,G,H),anc(I,J),ans(K,L)) :-
   ex_minima_caracterizacao(A,B,C,D,ths(E,M,G,N),anc([falha(A,B,C,D)|I],J),ans(K,O)),
   ex_desc(A,B,C,D,ths(M,F,N,H),anc([falha(A,B,C,D)|I],J),ans(O,L)).
```

C) DESCRI01.ARI :

```
:-segment(farseg2).

:- module descri01.

:- visible prove_minima_caracterizacao / 6.
:- visible prove_not_minima_caracterizacao / 6.
:- visible ex_minima_caracterizacao / 7.
:- visible ex_not_minima_caracterizacao / 7.
:- visible prove_not_atuado_alarme / 4.

:- public prove_minima_caracterizacao / 6.
:- public prove_not_minima_caracterizacao / 6.
:- public ex_minima_caracterizacao / 7.
:- public ex_not_minima_caracterizacao / 7.
:- public prove_not_atuado_alarme / 4.

:- extrn member /2:far.
:- extrn predicado_instanciado/1:far.
:- extrn prove_atuado_alarme / 4:far.
:- extrn ex_atuado_alarme / 5:far.
:- extrn prove_desc / 6:far.
:- extrn ex_desc / 7:far.
:- extrn ex_not_falha / 7:far.
:- extrn prove_falha / 6:far.
:- extrn prove_not_falha / 6:far.


prove_minima_caracterizacao(A,B,C,D,E,anc(F,G)) :-
   member(minima_caracterizacao(A,B,C,D),G).
prove_minima_caracterizacao(terra,proximo,lt_01,-,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_01,A,anc([minima_caracterizacao(terra,proximo,lt_01,-)|B],C)),
   prove_atuado_alarme(inst,lt_01,A,anc([minima_caracterizacao(terra,proximo,lt_01,-)|B],C)).
prove_minima_caracterizacao(fase,proximo,lt_01,-,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_01,A,anc([minima_caracterizacao(fase,proximo,lt_01,-)|B],C)),
   prove_atuado_alarme(inst,lt_01,A,anc([minima_caracterizacao(fase,proximo,lt_01,-)|B],C)).
prove_minima_caracterizacao(fase,distante,lt_01,-,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_01,A,anc([minima_caracterizacao(fase,distante,lt_01,-)|B],C)),
   prove_atuado_alarme(temp,lt_01,A,anc([minima_caracterizacao(fase,distante,lt_01,-)|B],C)).
prove_minima_caracterizacao(terra,distante,lt_01,-,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_01,A,anc([minima_caracterizacao(terra,distante,lt_01,-)|B],C)),
   prove_atuado_alarme(temp,lt_01,A,anc([minima_caracterizacao(terra,distante,lt_01,-)|B],C)).
prove_minima_caracterizacao(terra,proximo,lt_02,-,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_02,A,anc([minima_caracterizacao(terra,proximo,lt_02,-)|B],C)),
   prove_atuado_alarme(inst,lt_02,A,anc([minima_caracterizacao(terra,proximo,lt_02,-)|B],C)).
prove_minima_caracterizacao(fase,proximo,lt_02,-,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_02,A,anc([minima_caracterizacao(fase,proximo,lt_02,-)|B],C)),
   prove_atuado_alarme(inst,lt_02,A,anc([minima_caracterizacao(fase,proximo,lt_02,-)|B],C)).
prove_minima_caracterizacao(fase,distante,lt_02,-,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_02,A,anc([minima_caracterizacao(fase,distante,lt_02,-)|B],C)),
   prove_atuado_alarme(temp,lt_02,A,anc([minima_caracterizacao(fase,distante,lt_02,-)|B],C)).
prove_minima_caracterizacao(terra,distante,lt_02,-,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_02,A,anc([minima_caracterizacao(terra,distante,lt_02,-)|B],C)),
   prove_atuado_alarme(temp,lt_02,A,anc([minima_caracterizacao(terra,distante,lt_02,-)|B],C)).
```

```
prove_minima_caracterizacao(fase,proximo,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(fase,proximo,by_pass,A)|C],D)),
   prove_atuado_alarme(fase,by_pass,B,anc([minima_caracterizacao(fase,proximo,by_pass,A)|C],D)),
   prove_atuado_alarme(inst,by_pass,B,anc([minima_caracterizacao(fase,proximo,by_pass,A)|C],D)).
prove_minima_caracterizacao(terra,proximo,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(terra,proximo,by_pass,A)|C],D)),
   prove_atuado_alarme(terra,by_pass,B,anc([minima_caracterizacao(terra,proximo,by_pass,A)|C],
D)),
   prove_atuado_alarme(inst,by_pass,B,anc([minima_caracterizacao(terra,proximo,by_pass,A)|C],D)).

prove_minima_caracterizacao(fase,distante,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(fase,distante,by_pass,A)|C],D)),
   prove_atuado_alarme(fase,by_pass,B,anc([minima_caracterizacao(fase,distante,by_pass,A)|C],D)),
   prove_atuado_alarme(temp,by_pass,B,anc([minima_caracterizacao(fase,distante,by_pass,A)|C],D)).

prove_minima_caracterizacao(terra,distante,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(terra,distante,by_pass,A)|C],D)),
   prove_atuado_alarme(terra,by_pass,B,anc([minima_caracterizacao(terra,distante,by_pass,A)|C],
D)),
   prove_atuado_alarme(temp,by_pass,B,anc([minima_caracterizacao(terra,distante,by_pass,A)|C],
D)).

prove_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,lt_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)).

prove_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,lt_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)).

prove_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)).

prove_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_01,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)),
   prove_atuado_alarme(abriu,transf_02,A,anc([minima_caracterizacao(barra_princ,-,todos,-)|B],C)).

prove_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,C)) :-
   prove_atuado_alarme(aux,D,A,
   anc([minima_caracterizacao(barra_princ,-,todos,D)|B],C)),
   prove_atuado_alarme(abriu,lt_01,A,
   anc([minima_caracterizacao(barra_princ,-,todos,D)|B],C)),
   prove_atuado_alarme(abriu,lt_02,A,
   anc([minima_caracterizacao(barra_princ,-,todos,D)|B],C)).

prove_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,
   anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,lt_01,B,
   anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_01,B,
   anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)).
```

```
prove_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,lt_01,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_02,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)).

prove_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,lt_02,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_01,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)).

prove_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,lt_02,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_02,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)).

prove_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_01,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)),
   prove_atuado_alarme(abriu,transf_02,B,anc([minima_caracterizacao(barra_princ,-,todos,A)|C],D)).

prove_minima_caracterizacao(barra_aux,-,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,by_pass,B,anc([minima_caracterizacao(barra_aux,-,by_pass,A)|C],D)),
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(barra_aux,-,by_pass,A)|C],D)).

prove_minima_caracterizacao(sobre,-,A,-,B,anc(C,D)) :-
   prove_atuado_alarme(sobre,A,B,anc([minima_caracterizacao(sobre,-,A,-)|C],D)).

prove_minima_caracterizacao(sobre,-,by_pass,A,B,anc(C,D)) :-
   prove_atuado_alarme(sobre,A,B,anc([minima_caracterizacao(sobre,-,by_pass,A)|C],D)),
   prove_atuado_alarme(aux,A,B,anc([minima_caracterizacao(sobre,-,by_pass,A)|C],D)).

prove_not_minima_caracterizacao(A,B,C,D,E,anc(F,G)) :-
   member(minima_caracterizacao(A,B,C,D),F).
prove_not_minima_caracterizacao(A,B,C,D,E,anc(F,G)) :-
   prove_not_falha(A,B,C,D,E,anc(F,[minima_caracterizacao(A,B,C,D)|G])),
   prove_desc(A,B,C,D,E,anc(F,[minima_caracterizacao(A,B,C,D)|G])).

ex_minima_caracterizacao(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(minima_caracterizacao(A,B,C,D),H).
ex_minima_caracterizacao(terra,proximo,lt_01,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(terra,proximo,lt_01,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(inst,lt_01,ths(I,B,J,D),anc([minima_caracterizacao(terra,proximo,lt_01,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(fase,proximo,lt_01,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(fase,proximo,lt_01,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(inst,lt_01,ths(I,B,J,D),anc([minima_caracterizacao(fase,proximo,lt_01,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(fase,distante,lt_01,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(fase,distante,lt_01,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(temp,lt_01,ths(I,B,J,D),anc([minima_caracterizacao(fase,distante,lt_01,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(terra,distante,lt_01,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
```

```
   ex_atuado_alarme(terra,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(terra,distante,lt_01,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(temp,lt_01,ths(I,B,J,D),anc([minima_caracterizacao(terra,distante,lt_01,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(terra,proximo,lt_02,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_02,ths(A,I,C,J),anc([minima_caracterizacao(terra,proximo,lt_02,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(inst,lt_02,ths(I,B,J,D),anc([minima_caracterizacao(terra,proximo,lt_02,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(fase,proximo,lt_02,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_02,ths(A,I,C,J),anc([minima_caracterizacao(fase,proximo,lt_02,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(inst,lt_02,ths(I,B,J,D),anc([minima_caracterizacao(fase,proximo,lt_02,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(fase,distante,lt_02,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_02,ths(A,I,C,J),anc([minima_caracterizacao(fase,distante,lt_02,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(temp,lt_02,ths(I,B,J,D),anc([minima_caracterizacao(fase,distante,lt_02,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(terra,distante,lt_02,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_02,ths(A,I,C,J),anc([minima_caracterizacao(terra,distante,lt_02,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(temp,lt_02,ths(I,B,J,D),anc([minima_caracterizacao(terra,distante,lt_02,-)
|E],F),ans(K,H)).

ex_minima_caracterizacao(fase,proximo,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(fase,proximo,by_pass,A)|F],G),
ans(H,L)),
   ex_atuado_alarme(fase,by_pass,ths(J,M,K,N),anc([minima_caracterizacao(fase,proximo,by_pass,
A)| F],G),ans(L,O)),
   ex_atuado_alarme(inst,by_pass,ths(M,C,N,E),anc([minima_caracterizacao(fase,proximo,by_pass,
A)|F],G),ans(O,I)).

ex_minima_caracterizacao(terra,proximo,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(terra,proximo,by_pass,A)|F],
G),ans(H,L)),
   ex_atuado_alarme(terra,by_pass,ths(J,M,K,N),anc([minima_caracterizacao(terra,proximo,by_pass,
A)|F],G),ans(L,O)),
   ex_atuado_alarme(inst,by_pass,ths(M,C,N,E),anc([minima_caracterizacao(terra,proximo,by_pass,
A)|F],G),ans(O,I)).

ex_minima_caracterizacao(fase,distante,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(fase,distante,by_pass,A)|F],
G),ans(H,L)),
   ex_atuado_alarme(fase,by_pass,ths(J,M,K,N),anc([minima_caracterizacao(fase,distante,by_pass,
A)|F],G),ans(L,O)),
   ex_atuado_alarme(temp,by_pass,ths(M,C,N,E),anc([minima_caracterizacao(fase,distante,by_pass,
A)|F],G),ans(O,I)).

ex_minima_caracterizacao(terra,distante,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(terra,distante,by_pass,A)|F],
G),ans(H,L)),
   ex_atuado_alarme(terra,by_pass,ths(J,M,K,N),anc([minima_caracterizacao(terra,distante,by_pass,
A)|F],G),ans(L,O)),
```

```
   ex_atuado_alarme(temp,by_pass,ths(M,C,N,E),anc([minima_caracterizacao(terra,distante,by_pass,
A)|F],G),ans(O,I)).

ex_minima_caracterizacao(barra_princ,-,todos,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(abriu,lt_02,ths(I,L,J,M),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(K,N)),
  ex_atuado_alarme(abriu,transf_01,ths(L,B,M,D),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(N,H)).

ex_minima_caracterizacao(barra_princ,-,todos,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(abriu,lt_02,ths(I,L,J,M),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(K,N)),
  ex_atuado_alarme(abriu,transf_02,ths(L,B,M,D),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(N,H)).

ex_minima_caracterizacao(barra_princ,-,todos,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(abriu,transf_01,ths(I,L,J,M),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(K,N)),
  ex_atuado_alarme(abriu,transf_02,ths(L,B,M,D),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(N,H)).

ex_minima_caracterizacao(barra_princ,-,todos,-,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(G,K)),
   ex_atuado_alarme(abriu,transf_01,ths(I,L,J,M),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(K,N)),
  ex_atuado_alarme(abriu,transf_02,ths(L,B,M,D),anc([minima_caracterizacao(barra_princ,-,todos,-)
|E],F),ans(N,H)).

ex_minima_caracterizacao(barra_princ,-,todos,I,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(aux,I,ths(A,J,C,K), anc([minima_caracterizacao(barra_princ,-,todos,I) |E],F) ,
ans(G,L)),
   ex_atuado_alarme(abriu,lt_01,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-,todos,I)|E],
F),ans(L,O)),
   ex_atuado_alarme(abriu,lt_02,ths(M,B,N,D),anc([minima_caracterizacao(barra_princ,-,todos,I)|E],
F),ans(O,H)).

ex_minima_caracterizacao(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(barra_princ,-,todos,A)|F],
G),ans(H,L)),
   ex_atuado_alarme(abriu,lt_01,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-,todos,A)|F],
G),ans(L,O)),
   ex_atuado_alarme(abriu,transf_01,ths(M,C,N,E),anc([minima_caracterizacao(barra_princ,-,todos
,A)| F], G),ans(O,I)).

ex_minima_caracterizacao(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(barra_princ,-,todos,A)|F],
G),ans(H,L)),
   ex_atuado_alarme(abriu,lt_01,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-,todos,A)|F],
G),ans(L,O)),
   ex_atuado_alarme(abriu,transf_02,ths(M,C,N,E),anc([minima_caracterizacao(barra_princ,-,todos,
A)|F], G),ans(O,I)).
```

```
ex_minima_caracterizacao(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(H,L)),
   ex_atuado_alarme(abriu,lt_02,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(L,O)),
   ex_atuado_alarme(abriu,transf_01,ths(M,C,N,E),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(O,I)).

ex_minima_caracterizacao(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(H,L)),
   ex_atuado_alarme(abriu,lt_02,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(L,O)),
   ex_atuado_alarme(abriu,transf_02,ths(M,C,N,E),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(O,I)).

ex_minima_caracterizacao(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(H,L)),
   ex_atuado_alarme(abriu,transf_01,ths(J,M,K,N),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(L,O)),
   ex_atuado_alarme(abriu,transf_02,ths(M,C,N,E),anc([minima_caracterizacao(barra_princ,-
,todos,A)|F],G),ans(O,I)).

ex_minima_caracterizacao(barra_aux,-,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,by_pass,ths(B,J,D,K),anc([minima_caracterizacao(barra_aux,-
,by_pass,A)|F],G),ans(H,L)),
   ex_atuado_alarme(aux,A,ths(J,C,K,E),anc([minima_caracterizacao(barra_aux,-
,by_pass,A)|F],G),ans(L,I)).

ex_minima_caracterizacao(sobre,-,A,-,B,anc(C,D),E) :-
   ex_atuado_alarme(sobre,A,B,anc([minima_caracterizacao(sobre,-,A,-)|C],D),E).
ex_minima_caracterizacao(sobre,-,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(sobre,A,ths(B,J,D,K),anc([minima_caracterizacao(sobre,-
,by_pass,A)|F],G),ans(H,L)),
   ex_atuado_alarme(aux,A,ths(J,C,K,E),anc([minima_caracterizacao(sobre,-
,by_pass,A)|F],G),ans(L,I)).


ex_not_minima_caracterizacao(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(minima_caracterizacao(A,B,C,D),G).
ex_not_minima_caracterizacao(A,B,C,D,ths(E,F,G,H),anc(I,J),ans(K,L)) :-
   ex_not_falha(A,B,C,D,ths(E,M,G,N),anc(I,[minima_caracterizacao(A,B,C,D)|J]),ans(K,O)),
   ex_desc(A,B,C,D,ths(M,F,N,H),anc(I,[minima_caracterizacao(A,B,C,D)|J]),ans(O,L)).


prove_not_atuado_alarme(A,B,C,anc(D,E)) :-
   member(atuado_alarme(A,B),D).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_falha(barra_princ,-,todos,-,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(A,by_pass,B,anc(C,D)) :-
   prove_falha(barra_princ,-,todos,-,B,anc(C,[atuado_alarme(A,by_pass)|D])).
prove_not_atuado_alarme(aux,transf_01,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_01,-,A,anc(B,[atuado_alarme(aux,transf_01)|C])).
prove_not_atuado_alarme(aux,transf_02,A,anc(B,C)) :-
   prove_falha(sobre,-,transf_02,-,A,anc(B,[atuado_alarme(aux,transf_02)|C])).
prove_not_atuado_alarme(terra,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(inst,lt_01,A,anc(B,[atuado_alarme(terra,lt_01)|C])),
```

```
   prove_not_minima_caracterizacao(terra,proximo,lt_01,-,A,anc(B,[atuado_alarme(terra,lt_01)|C])).
prove_not_atuado_alarme(inst,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_01,A,anc(B,[atuado_alarme(inst,lt_01)|C])),
   prove_not_minima_caracterizacao(terra,proximo,lt_01,-,A,anc(B,[atuado_alarme(inst,lt_01)|C])).
prove_not_atuado_alarme(fase,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(inst,lt_01,A,anc(B,[atuado_alarme(fase,lt_01)|C])),
   prove_not_minima_caracterizacao(fase,proximo,lt_01,-,A,anc(B,[atuado_alarme(fase,lt_01)|C])).
prove_not_atuado_alarme(inst,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_01,A,anc(B,[atuado_alarme(inst,lt_01)|C])),
   prove_not_minima_caracterizacao(fase,proximo,lt_01,-,A,anc(B,[atuado_alarme(inst,lt_01)|C])).
prove_not_atuado_alarme(fase,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(temp,lt_01,A,anc(B,[atuado_alarme(fase,lt_01)|C])),
   prove_not_minima_caracterizacao(fase,distante,lt_01,-,A,anc(B,[atuado_alarme(fase,lt_01)|C])).
prove_not_atuado_alarme(temp,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_01,A,anc(B,[atuado_alarme(temp,lt_01)|C])),
   prove_not_minima_caracterizacao(fase,distante,lt_01,-,A,anc(B,[atuado_alarme(temp,lt_01)|C])).
prove_not_atuado_alarme(terra,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(temp,lt_01,A,anc(B,[atuado_alarme(terra,lt_01)|C])),
   prove_not_minima_caracterizacao(terra,distante,lt_01,-,A,anc(B,[atuado_alarme(terra,lt_01)|C])).
prove_not_atuado_alarme(temp,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_01,A,anc(B,[atuado_alarme(temp,lt_01)|C])),
   prove_not_minima_caracterizacao(terra,distante,lt_01,-,A,anc(B,[atuado_alarme(temp,lt_01)|C])).
prove_not_atuado_alarme(terra,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(inst,lt_02,A,anc(B,[atuado_alarme(terra,lt_02)|C])),
   prove_not_minima_caracterizacao(terra,proximo,lt_02,-,A,anc(B,[atuado_alarme(terra,lt_02)|C])).
prove_not_atuado_alarme(inst,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_02,A,anc(B,[atuado_alarme(inst,lt_02)|C])),
   prove_not_minima_caracterizacao(terra,proximo,lt_02,-,A,anc(B,[atuado_alarme(inst,lt_02)|C])).
prove_not_atuado_alarme(fase,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(inst,lt_02,A,anc(B,[atuado_alarme(fase,lt_02)|C])),
   prove_not_minima_caracterizacao(fase,proximo,lt_02,-,A,anc(B,[atuado_alarme(fase,lt_02)|C])).
prove_not_atuado_alarme(inst,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_02,A,anc(B,[atuado_alarme(inst,lt_02)|C])),
   prove_not_minima_caracterizacao(fase,proximo,lt_02,-,A,anc(B,[atuado_alarme(inst,lt_02)|C])).
prove_not_atuado_alarme(fase,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(temp,lt_02,A,anc(B,[atuado_alarme(fase,lt_02)|C])),
   prove_not_minima_caracterizacao(fase,distante,lt_02,-,A,anc(B,[atuado_alarme(fase,lt_02)|C])).
prove_not_atuado_alarme(temp,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(fase,lt_02,A,anc(B,[atuado_alarme(temp,lt_02)|C])),
   prove_not_minima_caracterizacao(fase,distante,lt_02,-,A,anc(B,[atuado_alarme(temp,lt_02)|C])).
prove_not_atuado_alarme(terra,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(temp,lt_02,A,anc(B,[atuado_alarme(terra,lt_02)|C])),
   prove_not_minima_caracterizacao(terra,distante,lt_02,-,A,anc(B,[atuado_alarme(terra,lt_02)|C])).
prove_not_atuado_alarme(temp,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(terra,lt_02,A,anc(B,[atuado_alarme(temp,lt_02)|C])),
   prove_not_minima_caracterizacao(terra,distante,lt_02,-,A,anc(B,[atuado_alarme(temp,lt_02)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(fase,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(inst,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(fase,proximo,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(fase,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(inst,by_pass,A,anc(B,[atuado_alarme(fase,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(fase,by_pass)|C])),
   prove_not_minima_caracterizacao(fase,proximo,by_pass,D,A,anc(B,[atuado_alarme(fase,
by_pass)|C])).
prove_not_atuado_alarme(inst,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(fase,by_pass,A,anc(B,[atuado_alarme(inst,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(inst,by_pass)|C])),
```

```
   prove_not_minima_caracterizacao(fase,proximo,by_pass,D,A,anc(B,[atuado_alarme(inst,
by_pass)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(terra,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(inst,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(terra,proximo,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).

prove_not_atuado_alarme(terra,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(inst,by_pass,A,anc(B,[atuado_alarme(terra,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(terra,by_pass)|C])),
   prove_not_minima_caracterizacao(terra,proximo,by_pass,D,A,anc(B,[atuado_alarme(terra,
by_pass)|C])).

prove_not_atuado_alarme(inst,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(terra,by_pass,A,anc(B,[atuado_alarme(inst,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(inst,by_pass)|C])),
   prove_not_minima_caracterizacao(terra,proximo,by_pass,D,A,anc(B,[atuado_alarme(inst,
by_pass)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(fase,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(temp,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(fase,distante,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).

prove_not_atuado_alarme(fase,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(temp,by_pass,A,anc(B,[atuado_alarme(fase,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(fase,by_pass)|C])),
   prove_not_minima_caracterizacao(fase,distante,by_pass,D,A,anc(B,[atuado_alarme(fase,
by_pass)|C])).

prove_not_atuado_alarme(temp,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(fase,by_pass,A,anc(B,[atuado_alarme(temp,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(temp,by_pass)|C])),
   prove_not_minima_caracterizacao(fase,distante,by_pass,D,A,anc(B,[atuado_alarme(temp,
by_pass)|C])).

prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(terra,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(temp,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(terra,distante,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).

prove_not_atuado_alarme(terra,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(temp,by_pass,A,anc(B,[atuado_alarme(terra,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(terra,by_pass)|C])),
   prove_not_minima_caracterizacao(terra,distante,by_pass,D,A,anc(B,[atuado_alarme(terra,
by_pass)|C])).
prove_not_atuado_alarme(temp,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(terra,by_pass,A,anc(B,[atuado_alarme(temp,by_pass)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(temp,by_pass)|C])),
   prove_not_minima_caracterizacao(terra,distante,by_pass,D,A,anc(B,[atuado_alarme(temp,
by_pass)|C])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_01)|C])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_02)|C])).
```

```
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B, [atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_01)|C])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_02)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_01)|C])).
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,lt_02)|C])).
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,-,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,lt_01,B,
   anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,lt_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_01)|C])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
```

```
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_02)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,lt_01,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,transf_01,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_01)|C])).
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,lt_01,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,transf_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,lt_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_01)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,lt_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,transf_01,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_02)|C])).
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,lt_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,transf_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,lt_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,lt_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
lt_02)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,lt_02,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
```

```
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,transf_01,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_atuado_alarme(abriu,transf_02,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(abriu,transf_01,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_02,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_01)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_01)|C])).
prove_not_atuado_alarme(abriu,transf_02,A,anc(B,C)) :-
   prove_atuado_alarme(abriu,transf_01,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,transf_02)|C])),
   prove_not_minima_caracterizacao(barra_princ,-,todos,D,A,anc(B,[atuado_alarme(abriu,
transf_02)|C])).
prove_not_atuado_alarme(abriu,by_pass,A,anc(B,C)) :-
   prove_atuado_alarme(aux,D,A,anc(B,[atuado_alarme(abriu,by_pass)|C])),
   prove_not_minima_caracterizacao(barra_aux,-,by_pass,D,A,anc(B,[atuado_alarme(abriu,
by_pass)|C])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(abriu,by_pass,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(barra_aux,-,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).
prove_not_atuado_alarme(sobre,A,B,anc(C,D)) :-
   prove_not_minima_caracterizacao(sobre,-,A,-,B,anc(C,[atuado_alarme(sobre,A)|D])).
prove_not_atuado_alarme(sobre,A,B,anc(C,D)) :-
   prove_atuado_alarme(aux,A,B,anc(C,[atuado_alarme(sobre,A)|D])),
   prove_not_minima_caracterizacao(sobre,-,by_pass,A,B,anc(C,[atuado_alarme(sobre,A)|D])).
prove_not_atuado_alarme(aux,A,B,anc(C,D)) :-
   prove_atuado_alarme(sobre,A,B,anc(C,[atuado_alarme(aux,A)|D])),
   prove_not_minima_caracterizacao(sobre,-,by_pass,A,B,anc(C,[atuado_alarme(aux,A)|D])).
```

D) DESCRI02.ARI :

```
:-segment(farseg3).

:- module descri02.

:- visible ex_not_atuado_alarme / 5.
:- visible prove_obs_alr / 4.
:- visible prove_not_obs_alr / 4.
:- visible ex_obs_alr / 5.
:- visible prove_atuado_alarme / 4.
:- visible ex_atuado_alarme / 5.
:- visible prove_comp_auxiliado_pelo_by_pass / 3.
:- visible prove_not_comp_auxiliado_pelo_by_pass / 3.
:- visible ex_comp_auxiliado_pelo_by_pass / 4.
:- visible ex_not_comp_auxiliado_pelo_by_pass / 4.
:- visible prove_e_by_pass / 3.
:- visible prove_not_e_by_pass / 3.
:- visible ex_e_by_pass / 4.
:- visible ex_not_e_by_pass / 4.
:- visible prove_e_transformador / 3.
:- visible prove_not_e_transformador / 3.
:- visible ex_e_transformador / 4.
:- visible ex_not_e_transformador / 4.
:- visible prove_e_linha_transmissao / 3.
:- visible prove_not_e_linha_transmissao / 3.
:- visible ex_e_linha_transmissao / 4.
:- visible ex_not_e_linha_transmissao / 4.
:- visible prove_desc / 6.
:- visible prove_not_desc / 6.
:- visible ex_desc / 7.
:- visible ex_not_desc / 7.
:- visible ex_not_falha / 7.

:- public ex_not_atuado_alarme / 5.
:- public prove_obs_alr / 4.
:- public prove_not_obs_alr / 4.
:- public ex_obs_alr / 5.
:- public ex_not_obs_alr / 5.
:- public prove_atuado_alarme / 4.
:- public ex_atuado_alarme / 5.
:- public prove_comp_auxiliado_pelo_by_pass / 3.
:- public prove_not_comp_auxiliado_pelo_by_pass / 3.
:- public ex_comp_auxiliado_pelo_by_pass / 4.
:- public ex_not_comp_auxiliado_pelo_by_pass / 4.
:- public prove_e_by_pass / 3.
:- public prove_not_e_by_pass / 3.
:- public ex_e_by_pass / 4.
:- public ex_not_e_by_pass / 4.
:- public prove_e_transformador / 3.
:- public prove_not_e_transformador / 3.
:- public ex_e_transformador / 4.
:- public ex_not_e_transformador / 4.
:- public prove_e_linha_transmissao / 3.
:- public prove_not_e_linha_transmissao / 3.
:- public ex_e_linha_transmissao / 4.
:- public ex_not_e_linha_transmissao / 4.
```

```
:- public prove_desc / 6.
:- public prove_not_desc / 6.
:- public ex_desc / 7.
:- public ex_not_desc / 7.
:- public ex_not_falha / 7.

:- extrn member /2:far.
:- extrn predicado_instanciado/1:far.
:- extrn ex_not_falha / 7:far.
:- extrn ex_falha / 7:far.
:- extrn prove_not_falha / 6:far.
:- extrn prove_falha / 6:far.
:- extrn ex_atuado_alarme / 5:far.
:- extrn prove_not_atuado_alarme / 4:far.
:- extrn ex_not_atuado_alarme / 5:far.
:- extrn prove_not_alr / 4:far.
:- extrn ex_not_alr / 5:far.
:- extrn prove_minima_caracterizacao / 6:far.
:- extrn ex_minima_caracterizacao / 7:far.
:- extrn ex_not_minima_caracterizacao / 7:far.


ex_not_atuado_alarme(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(atuado_alarme(A,B),E).
ex_not_atuado_alarme(aux,A,B,anc(C,D),E) :-
   ex_falha(barra_princ,-,todos,-,B,anc(C,[atuado_alarme(aux,A)|D]),E).
ex_not_atuado_alarme(A,by_pass,B,anc(C,D),E) :-
   ex_falha(barra_princ,-,todos,-,B,anc(C,[atuado_alarme(A,by_pass)|D]),E).
ex_not_atuado_alarme(aux,transf_01,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_01,-,A,anc(B,[atuado_alarme(aux,transf_01)|C]),D).
ex_not_atuado_alarme(aux,transf_02,A,anc(B,C),D) :-
   ex_falha(sobre,-,transf_02,-,A,anc(B,[atuado_alarme(aux,transf_02)|C]),D).
ex_not_atuado_alarme(terra,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(terra,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,proximo,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(terra,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(inst,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(inst,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,proximo,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(inst,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(fase,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(fase,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,proximo,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(fase,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(inst,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(inst,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,proximo,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(inst,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(fase,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(fase,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,distante,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(fase,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(temp,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(temp,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,distante,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(temp,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(terra,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(terra,lt_01)|F]),ans(G,K)),
```

```
   ex_not_minima_caracterizacao(terra,distante,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(terra,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(temp,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(temp,lt_01)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,distante,lt_01,-
,ths(I,B,J,D),anc(E,[atuado_alarme(temp,lt_01)|F]),ans(K,H)).
ex_not_atuado_alarme(terra,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(terra,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,proximo,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(terra,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(inst,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(inst,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,proximo,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(inst,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(fase,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(fase,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,proximo,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(fase,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(inst,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(inst,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,proximo,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(inst,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(fase,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(fase,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,distante,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(fase,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(temp,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(temp,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(fase,distante,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(temp,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(terra,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(terra,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,distante,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(terra,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(temp,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(temp,lt_02)|F]),ans(G,K)),
   ex_not_minima_caracterizacao(terra,distante,lt_02,-
,ths(I,B,J,D),anc(E,[atuado_alarme(temp,lt_02)|F]),ans(K,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(fase,by_pass,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(inst,by_pass,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),

ex_not_minima_caracterizacao(fase,proximo,by_pass,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|
G]),ans(O,I)).
ex_not_atuado_alarme(fase,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(fase,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(fase,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(fase,proximo,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(fase,by_
pass)|F]),ans(O,H)).
ex_not_atuado_alarme(inst,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(inst,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(inst,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(fase,proximo,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(inst,by_
pass)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(terra,by_pass,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
```

```
   ex_atuado_alarme(inst,by_pass,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),

ex_not_minima_caracterizacao(terra,proximo,by_pass,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|
G]),ans(O,I)).
ex_not_atuado_alarme(terra,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(inst,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(terra,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(terra,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(terra,proximo,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(terra,by
_pass)|F]),ans(O,H)).
ex_not_atuado_alarme(inst,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(inst,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(inst,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(terra,proximo,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(inst,by_
pass)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(fase,by_pass,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(temp,by_pass,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),

ex_not_minima_caracterizacao(fase,distante,by_pass,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G
]),ans(O,I)).
ex_not_atuado_alarme(fase,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(fase,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(fase,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(fase,distante,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(fase,by_
pass)|F]),ans(O,H)).
ex_not_atuado_alarme(temp,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(fase,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(temp,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(temp,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(fase,distante,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(temp,by_
pass)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(terra,by_pass,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(temp,by_pass,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),

ex_not_minima_caracterizacao(terra,distante,by_pass,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|
G]),ans(O,I)).
ex_not_atuado_alarme(terra,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(temp,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(terra,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(terra,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(terra,distante,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(terra,by
_pass)|F]),ans(O,H)).
ex_not_atuado_alarme(temp,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(terra,by_pass,ths(A,I,C,J),anc(E,[atuado_alarme(temp,by_pass)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(temp,by_pass)|F]),ans(K,O)),

ex_not_minima_caracterizacao(terra,distante,by_pass,L,ths(M,B,N,D),anc(E,[atuado_alarme(temp,by
_pass)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,transf_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
```

```
   ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,transf_02,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,transf_02,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_01,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,transf_02,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_02,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(N,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(abriu,lt_02,ths(I,L,J,M),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,N)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,-
,ths(L,B,M,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(N,H)).
```

```
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,lt_01,ths(B,J,D,K),
   anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,lt_02,ths(J,M,K,N),
   anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-,todos,A,
   ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,lt_01,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,transf_01,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,lt_01,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,transf_02,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,lt_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_01)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,lt_02,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,transf_01,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
```

```
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,lt_02,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,transf_02,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,lt_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,lt_02)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(abriu,lt_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(O,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,transf_01,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_atuado_alarme(abriu,transf_02,ths(J,M,K,N),anc(F,[atuado_alarme(aux,A)|G]),ans(L,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,A,ths(M,C,N,E),anc(F,[atuado_alarme(aux,A)|G]),ans(O,I)).
ex_not_atuado_alarme(abriu,transf_01,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_02,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_01)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,transf_02,ths(A,B,C,D),anc(E,F),ans(G,H)) :-

ex_atuado_alarme(abriu,transf_01,ths(A,I,C,J),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(G,K)),
   ex_atuado_alarme(aux,L,ths(I,M,J,N),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(K,O)),
   ex_not_minima_caracterizacao(barra_princ,-
,todos,L,ths(M,B,N,D),anc(E,[atuado_alarme(abriu,transf_02)|F]),ans(O,H)).
ex_not_atuado_alarme(abriu,by_pass,ths(A,B,C,D),anc(E,F),ans(G,H)) :-
   ex_atuado_alarme(aux,I,ths(A,J,C,K),anc(E,[atuado_alarme(abriu,by_pass)|F]),ans(G,L)),
   ex_not_minima_caracterizacao(barra_aux,-
,by_pass,I,ths(J,B,K,D),anc(E,[atuado_alarme(abriu,by_pass)|F]),ans(L,H)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(abriu,by_pass,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_not_minima_caracterizacao(barra_aux,-
,by_pass,A,ths(J,C,K,E),anc(F,[atuado_alarme(aux,A)|G]),ans(L,I)).
ex_not_atuado_alarme(sobre,A,B,anc(C,D),E) :-
   ex_not_minima_caracterizacao(sobre,-,A,-,B,anc(C,[atuado_alarme(sobre,A)|D]),E).
ex_not_atuado_alarme(sobre,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(aux,A,ths(B,J,D,K),anc(F,[atuado_alarme(sobre,A)|G]),ans(H,L)),
   ex_not_minima_caracterizacao(sobre,-
,by_pass,A,ths(J,C,K,E),anc(F,[atuado_alarme(sobre,A)|G]),ans(L,I)).
ex_not_atuado_alarme(aux,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_atuado_alarme(sobre,A,ths(B,J,D,K),anc(F,[atuado_alarme(aux,A)|G]),ans(H,L)),
   ex_not_minima_caracterizacao(sobre,-
,by_pass,A,ths(J,C,K,E),anc(F,[atuado_alarme(aux,A)|G]),ans(L,I)).

prove_atuado_alarme(A,B,C,anc(D,E)) :-
   member(atuado_alarme(A,B),E).
prove_atuado_alarme(A,B,C,anc(D,E)) :-
```

```
   call(prove_obs_alr(A,B,C,anc([atuado_alarme(A,B)|D],E))).

ex_atuado_alarme(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(atuado_alarme(A,B),F).
ex_atuado_alarme(A,B,C,anc(D,E),F) :-
   call(ex_obs_alr(A,B,C,anc([atuado_alarme(A,B)|D],E),F)).

prove_obs_alr(A,B,C,anc(D,E)) :-
   member(obs_alr(A,B),E).

prove_not_obs_alr(A,B,C,anc(D,E)) :-
   member(obs_alr(A,B),D).
prove_not_obs_alr(A,B,C,anc(D,E)) :-
   prove_not_atuado_alarme(A,B,C,anc(D,[obs_alr(A,B)|E])).

ex_obs_alr(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(obs_alr(A,B),F).

ex_not_obs_alr(A,B,ths(C,C,D,D),anc(E,F),ans(G,G)) :-
   member(obs_alr(A,B),E).
ex_not_obs_alr(A,B,C,anc(D,E),F) :-
   ex_not_atuado_alarme(A,B,C,anc(D,[obs_alr(A,B)|E]),F).

prove_e_linha_transmissao(A,B,anc(C,D)) :-
   member(e_linha_transmissao(A),D).
prove_e_linha_transmissao(lt_01,A,B).
prove_e_linha_transmissao(lt_02,A,B).

prove_not_e_linha_transmissao(A,B,anc(C,D)) :-
   member(e_linha_transmissao(A),C).
prove_not_e_linha_transmissao(A,B,anc(C,D)) :-
   prove_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,[e_linha_transmissao(A)|D])).

ex_e_linha_transmissao(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_linha_transmissao(A),E).
ex_e_linha_transmissao(lt_01,ths(A,A,B,B),C,ans(D,D)).
ex_e_linha_transmissao(lt_02,ths(A,A,B,B),C,ans(D,D)).

ex_not_e_linha_transmissao(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_linha_transmissao(A),D).
ex_not_e_linha_transmissao(A,B,anc(C,D),E) :-
   ex_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,[e_linha_transmissao(A)|D]),E).

prove_e_transformador(A,B,anc(C,D)) :-
   member(e_transformador(A),D).
prove_e_transformador(transf_01,A,B).
prove_e_transformador(transf_02,A,B).

prove_not_e_transformador(A,B,anc(C,D)) :-
   member(e_transformador(A),C).
prove_not_e_transformador(A,B,anc(C,D)) :-
   prove_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,[e_transformador(A)|D])).

ex_e_transformador(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_transformador(A),E).
ex_e_transformador(transf_01,ths(A,A,B,B),C,ans(D,D)).
ex_e_transformador(transf_02,ths(A,A,B,B),C,ans(D,D)).
```

```
ex_not_e_transformador(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_transformador(A),D).
ex_not_e_transformador(A,B,anc(C,D),E) :-
   ex_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,[e_transformador(A)|D]),E).

prove_e_by_pass(A,B,anc(C,D)) :-
   member(e_by_pass(A),D).
prove_e_by_pass(by_pass,A,B).

prove_not_e_by_pass(A,B,anc(C,D)) :-
   member(e_by_pass(A),C).

ex_e_by_pass(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_by_pass(A),E).
ex_e_by_pass(by_pass,ths(A,A,B,B),C,ans(D,D)).

ex_not_e_by_pass(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(e_by_pass(A),D).

prove_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   member(comp_auxiliado_pelo_by_pass(A),D).
prove_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   prove_e_linha_transmissao(A,B,anc([comp_auxiliado_pelo_by_pass(A)|C],D)).
prove_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   prove_e_transformador(A,B,anc([comp_auxiliado_pelo_by_pass(A)|C],D)).

prove_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   member(comp_auxiliado_pelo_by_pass(A),C).
prove_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   prove_falha(barra_princ,-,todos,A,B,anc(C,[comp_auxiliado_pelo_by_pass(A)|D])),
   prove_not_alr(abriu,by_pass,B,anc(C,[comp_auxiliado_pelo_by_pass(A)|D])).
prove_not_comp_auxiliado_pelo_by_pass(A,B,anc(C,D)) :-
   prove_falha(barra_aux,-,by_pass,A,B,anc(C,[comp_auxiliado_pelo_by_pass(A)|D])),
   prove_not_alr(abriu,by_pass,B,anc(C,[comp_auxiliado_pelo_by_pass(A)|D])).

ex_comp_auxiliado_pelo_by_pass(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(comp_auxiliado_pelo_by_pass(A),E).
ex_comp_auxiliado_pelo_by_pass(A,B,anc(C,D),E) :-
   ex_e_linha_transmissao(A,B,anc([comp_auxiliado_pelo_by_pass(A)|C],D),E).
ex_comp_auxiliado_pelo_by_pass(A,B,anc(C,D),E) :-
   ex_e_transformador(A,B,anc([comp_auxiliado_pelo_by_pass(A)|C],D),E).

ex_not_comp_auxiliado_pelo_by_pass(A,ths(B,B,C,C),anc(D,E),ans(F,F)) :-
   member(comp_auxiliado_pelo_by_pass(A),D).
ex_not_comp_auxiliado_pelo_by_pass(A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_falha(barra_princ,-
,todos,A,ths(B,J,D,K),anc(F,[comp_auxiliado_pelo_by_pass(A)|G]),ans(H,L)),
   ex_not_alr(abriu,by_pass,ths(J,C,K,E),anc(F,[comp_auxiliado_pelo_by_pass(A)|G]),ans(L,I)).
ex_not_comp_auxiliado_pelo_by_pass(A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_falha(barra_aux,-
,by_pass,A,ths(B,J,D,K),anc(F,[comp_auxiliado_pelo_by_pass(A)|G]),ans(H,L)),
   ex_not_alr(abriu,by_pass,ths(J,C,K,E),anc(F,[comp_auxiliado_pelo_by_pass(A)|G]),ans(L,I)).


prove_desc(A,B,C,D,E,anc(F,G)) :-
   member(desc(A,B,C,D),G).
prove_desc(A,B,C,D,E,F) :-
   member(desc(A,B,C,D),E).
```

```
prove_not_desc(A,B,C,D,E,anc(F,G)) :-
   member(desc(A,B,C,D),F).
prove_not_desc(A,B,C,D,E,anc(F,G)) :-
   prove_not_falha(A,B,C,D,E,anc(F,[desc(A,B,C,D)|G])),
   prove_minima_caracterizacao(A,B,C,D,E,anc(F,[desc(A,B,C,D)|G])).

ex_desc(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(desc(A,B,C,D),H).
ex_desc(A,B,C,D,ths(E,E,F,F),G,ans(H,H)) :-
   member(desc(A,B,C,D),E).
ex_desc(A,B,C,D,ths(E,[desc(A,B,C,D)|E],F,F),G,ans(H,H)) :-
   predicado_instanciado(desc(A,B,C,D)),
   \+ member(desc(A,B,C,D),E),
   \+ prove_not_desc(A,B,C,D,[desc(A,B,C,D)|E],anc([],[])).
ex_desc(A,B,C,D,ths(E,E,F,[desc(A,B,C,D)|F]),G,ans(H,H)) :-
   \+ predicado_instanciado(desc(A,B,C,D)).

ex_not_desc(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(desc(A,B,C,D),G).
ex_not_desc(A,B,C,D,ths(E,F,G,H),anc(I,J),ans(K,L)) :-
   ex_not_falha(A,B,C,D,ths(E,M,G,N),anc(I,[desc(A,B,C,D)|J]),ans(K,O)),
   ex_minima_caracterizacao(A,B,C,D,ths(M,F,N,H),anc(I,[desc(A,B,C,D)|J]),ans(O,L)).

ex_not_falha(A,B,C,D,ths(E,E,F,F),anc(G,H),ans(I,I)) :-
   member(falha(A,B,C,D),G).
ex_not_falha(terra,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C]),D).
ex_not_falha(terra,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(terra,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C]),D).
ex_not_falha(terra,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(inst,lt_01,A,anc(B,[falha(terra,proximo,lt_01,-)|C]),D).
ex_not_falha(fase,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C]),D).
ex_not_falha(fase,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(fase,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C]),D).
ex_not_falha(fase,proximo,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(inst,lt_01,A,anc(B,[falha(fase,proximo,lt_01,-)|C]),D).
ex_not_falha(fase,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C]),D).
ex_not_falha(fase,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(fase,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C]),D).
ex_not_falha(fase,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(temp,lt_01,A,anc(B,[falha(fase,distante,lt_01,-)|C]),D).
ex_not_falha(terra,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C]),D).
ex_not_falha(terra,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(terra,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C]),D).
ex_not_falha(terra,distante,lt_01,-,A,anc(B,C),D) :-
   ex_not_alr(temp,lt_01,A,anc(B,[falha(terra,distante,lt_01,-)|C]),D).
ex_not_falha(terra,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C]),D).
ex_not_falha(terra,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(terra,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C]),D).
ex_not_falha(terra,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(inst,lt_02,A,anc(B,[falha(terra,proximo,lt_02,-)|C]),D).
ex_not_falha(fase,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C]),D).
```

```
ex_not_falha(fase,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(fase,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C]),D).
ex_not_falha(fase,proximo,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(inst,lt_02,A,anc(B,[falha(fase,proximo,lt_02,-)|C]),D).
ex_not_falha(fase,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C]),D).
ex_not_falha(fase,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(fase,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C]),D).
ex_not_falha(fase,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(temp,lt_02,A,anc(B,[falha(fase,distante,lt_02,-)|C]),D).
ex_not_falha(terra,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C]),D).
ex_not_falha(terra,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(terra,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C]),D).
ex_not_falha(terra,distante,lt_02,-,A,anc(B,C),D) :-
   ex_not_alr(temp,lt_02,A,anc(B,[falha(terra,distante,lt_02,-)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(fase,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(fase,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(inst,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(inst,by_pass,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_01,A,anc(B,[falha(fase,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(fase,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_02,A,anc(B,[falha(fase,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(terra,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(terra,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(inst,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(inst,by_pass,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_01,A,anc(B,[falha(terra,proximo,by_pass,lt_01)|C]),D).
ex_not_falha(terra,proximo,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_02,A,anc(B,[falha(terra,proximo,by_pass,lt_02)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(fase,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(fase,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(temp,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C]),D).
```

```
ex_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(temp,by_pass,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_01,A,anc(B,[falha(fase,distante,by_pass,lt_01)|C]),D).
ex_not_falha(fase,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_02,A,anc(B,[falha(fase,distante,by_pass,lt_02)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(terra,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(terra,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(temp,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(temp,by_pass,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_01,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_01,A,anc(B,[falha(terra,distante,by_pass,lt_01)|C]),D).
ex_not_falha(terra,distante,by_pass,lt_02,A,anc(B,C),D) :-
   ex_not_alr(aux,lt_02,A,anc(B,[falha(terra,distante,by_pass,lt_02)|C]),D).
ex_not_falha(barra_princ,-,todos,A,B,anc(C,D),E) :-
   ex_not_alr(abriu,lt_01,B,anc(C,[falha(barra_princ,-,todos,A)|D]),E).
ex_not_falha(barra_princ,-,todos,A,B,anc(C,D),E) :-
   ex_not_alr(abriu,lt_02,B,anc(C,[falha(barra_princ,-,todos,A)|D]),E).
ex_not_falha(barra_princ,-,todos,A,B,anc(C,D),E) :-
   ex_not_alr(abriu,transf_01,B,anc(C,[falha(barra_princ,-,todos,A)|D]),E).
ex_not_falha(barra_princ,-,todos,A,B,anc(C,D),E) :-
   ex_not_alr(abriu,transf_02,B,anc(C,[falha(barra_princ,-,todos,A)|D]),E).
ex_not_falha(barra_princ,-,todos,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_comp_auxiliado_pelo_by_pass(A,ths(B,J,D,K),anc(F,[falha(barra_princ,-
,todos,A)|G]),ans(H,L)),
   ex_not_alr(abriu,by_pass,ths(J,C,K,E),anc(F,[falha(barra_princ,-,todos,A)|G]),ans(L,I)).
ex_not_falha(barra_princ,-,todos,A,B,anc(C,D),E) :-
   ex_not_alr(aux,A,B,anc(C,[falha(barra_princ,-,todos,A)|D]),E).
ex_not_falha(barra_princ,-,todos,-,A,anc(B,C),D) :-
   ex_atuado_alarme(aux,E,A,anc(B,[falha(barra_princ,-,todos,-)|C]),D).
ex_not_falha(barra_princ,-,todos,-,A,anc(B,C),D) :-
   ex_atuado_alarme(E,by_pass,A,anc(B,[falha(barra_princ,-,todos,-)|C]),D).
ex_not_falha(barra_aux,-,by_pass,A,B,anc(C,D),E) :-
   ex_not_alr(aux,A,B,anc(C,[falha(barra_aux,-,by_pass,A)|D]),E).
ex_not_falha(barra_aux,-,by_pass,A,ths(B,C,D,E),anc(F,G),ans(H,I)) :-
   ex_comp_auxiliado_pelo_by_pass(A,ths(B,J,D,K),anc(F,[falha(barra_aux,-
,by_pass,A)|G]),ans(H,L)),
   ex_not_alr(abriu,by_pass,ths(J,C,K,E),anc(F,[falha(barra_aux,-,by_pass,A)|G]),ans(L,I)).
ex_not_falha(sobre,-,transf_01,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C]),D).
ex_not_falha(sobre,-,transf_01,-,A,anc(B,C),D) :-
   ex_not_alr(sobre,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C]),D).
ex_not_falha(sobre,-,transf_02,-,A,anc(B,C),D) :-
   ex_not_alr(abriu,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C]),D).
ex_not_falha(sobre,-,transf_02,-,A,anc(B,C),D) :-
   ex_not_alr(sobre,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C]),D).
ex_not_falha(sobre,-,transf_01,-,A,anc(B,C),D) :-
   ex_atuado_alarme(aux,transf_01,A,anc(B,[falha(sobre,-,transf_01,-)|C]),D).
ex_not_falha(sobre,-,transf_02,-,A,anc(B,C),D) :-
   ex_atuado_alarme(aux,transf_02,A,anc(B,[falha(sobre,-,transf_02,-)|C]),D).
```

```
ex_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C),D) :-
   ex_not_alr(sobre,transf_01,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C]),D).
ex_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C),D) :-
   ex_not_alr(sobre,transf_02,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C]),D).
ex_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C),D) :-
   ex_not_alr(aux,transf_01,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C]),D).
ex_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C),D) :-
   ex_not_alr(aux,transf_02,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C]),D).
ex_not_falha(sobre,-,by_pass,transf_01,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(sobre,-,by_pass,transf_01)|C]),D).
ex_not_falha(sobre,-,by_pass,transf_02,A,anc(B,C),D) :-
   ex_not_alr(abriu,by_pass,A,anc(B,[falha(sobre,-,by_pass,transf_02)|C]),D).
```

Apêndice F - Interpretador dos Testes de Diagnóstico da Rede Neural

Neste apêndice, é apresentada a listagem do interpretador de resultados gerados pelos padrões de teste de diagnóstico da Rede Neural.

Este programa foi feito em C e apresenta seguinte estrutura :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>


/*
  Definição das constantes :
*/

/* Caracteriza o número máximo de neurônios na camada de entrada */
#define NUM_MAX_ENT 23
/* Caracteriza o número máximo de neurônios na camada de saída */
#define NUM_MAX_SAI 29


/*
  Definição das estruturas de dados utilizadas :
*/


/*
  Caracteriza a estrutura de dados do arquivo gerado durante os testes de Diagnostico das Redes
  Neurais. Este arquivo contêm os resultados obtidos.
*/
struct
result_teste_struct
{
 float entrada[NUM_MAX_ENT];
 float saida_desejada[NUM_MAX_SAI];
 float saida_real[NUM_MAX_SAI];
};
struct result_teste_struct teste;

/*
  Associa cada entrada da Rede Neural ao seu respectivo alarme :
*/
struct
 Nome_alarmes
 {
```

```
 char nome[30];
};
struct Nome_alarmes alarme[NUM_MAX_ENT]=
{ " Abert. Disj. Alim 01 -1 " , " Fase Alim 01 -2", " Terra Alim 01 -3" ,
  " Tempo Alim 01 -4", " Inst Alim 01 -5", " Auxi Alim 01 -6" ,
  " Abert. Disj. Alim 02 -7", " Fase Alim 02 -8", " Terra Alim 02 -9" ,
  " Tempo Alim 02 -10", " Inst Alim 02 -11", " Auxi Alim 02 -12" ,
  " Abert. Disj. By-pass -13", " Fase By-pass -14", " Terra By-pass -15" ,
  " Tempo By-pass -16", " Inst By-pass -17", " Abert. Disj. Transf 01 -18" ,
  " Aux Transf 01 -19" , " Sobre Transf 01 -20" ," Abert. Disj. Transf 02 -21" ,
  " Aux Transf 02 -22" , " Sobre Transf 02 -23"};

/*
  Associa cada saida da Rede Neural a sua respectiva falha :
*/
struct
Nome_falhas
{
 char nome[30];
};
struct Nome_falhas falha[NUM_MAX_SAI]=
{ " Prox ao Alim 01 F/F -1", " Prox ao Alim 01 F/F (1) -2",
  " Prox ao Alim 01 F/T -3", " Prox ao Alim 01 F/T (1) -4",
  " Dist ao Alim 01 F/F -5", " Dist ao Alim 01 F/F (1) -6",
  " Dist ao Alim 01 F/T -7", " Dist ao Alim 01 F/T (1) -8",
  " Prox ao Alim 02 F/F -9", " Prox ao Alim 02 F/F (2) -10",
  " Prox ao Alim 02 F/T -11", " Prox ao Alim 02 F/T (2) -12",
  " Dist ao Alim 02 F/F -13", " Dist ao Alim 02 F/F (2) -14",
  " Dist ao Alim 02 F/T -15", " Dist ao Alim 02 F/T (2) -16",
  " Barra Principal -17" , " Barra Principal(1) -18",
  " Barra Principal(2) -19",
  " Barra Principal(3) -20" , " Barra Principal(4) -21",
  " Barra Auxiliar(1) -22" , " Barra Auxiliar(2) -23",
  " Barra Auxiliar(3) -24" , " Barra Auxiliar(4) -25",
  " Transformador 01 -26" , " Transformador 02 -27" ,
  " Transformador 01(3) -28",   " Transformador 02(4) -29"};

/*
  Caracteriza a estrutura de dados de uma falha, indicando :
  • seu índice, ou seja, qual o neurônio de saída que foi ativado e
  • o valor de ativação correspondente a este neurônio de saída.
*/
struct
 struct_falha
 {
  int indice;
  float valor;
 };

/*****************************************************************************
```

PROGRAMA PRINCIPAL

```
*****************************************************************************/

main(argc,argv)
 int argc;
 char **argv;
```

```
{

/* Define a estrutura e o nome do arquivo de entrada */
FILE *arq_ent;
char Nome_arq[12];

/*
    Informa quais os alarmes que foram ativados durante o teste com a  Rede Neural.
*/
int Alarmes_ativados[NUM_MAX_ENT];

/*
    Informa quais as falhas que deveriam ter sido diagosticadas durante os testes com a Rede Neural.
*/
int Falha_desejada[NUM_MAX_SAI];

/* Caracteriza a estrutura de cada uma das falhas */
struct struct_falha Falha_prevista[NUM_MAX_SAI];
struct struct_falha Falha_provavel[NUM_MAX_SAI];
struct struct_falha Indicio_de_falha[NUM_MAX_SAI];



/*
    Define os contadores utilizados :
*/
int Num_Alarmes_ativados;
int Num_Falha_desejada;
int Num_Falha_prevista;
int Num_Falha_provavel;
int Num_Indicio_de_falha;
int i;

/*
    Verifica se o número de argumentos fornecidos ao programa não está correto.
*/
if (argc != 2)
{ /* Caso afirmativo, pede para digitar o nome do arquivo e sai do programa */
  printf(" Digite o nome do arquivo\n");
  exit(1);
}

/* Associa o nome fornecido pelo usuário à variável nome do arquivo */
sscanf(argv[1],"%s",Nome_arq);
strcat(Nome_arq,".nnr");
/* Testa se o arquivo não pode ser aberto */
if( (arq_ent = fopen(Nome_arq,"r")) == NULL)
{ /* Caso afirmativo, avisa que ocorreu um erro e sai do programa */
  printf(" Erro na abertura do arquivo\n");
  exit(1);
}


while(!feof(arq_ent))
{
  /* Zera os contadores */
  Num_Alarmes_ativados = 0;
  Num_Falha_desejada = 0;
```

```
Num_Falha_prevista = 0;
Num_Falha_provavel = 0;
Num_Indicio_de_falha = 0;

/* Lê os valores dos neurônios de entrada */
for(i=0; i<NUM_MAX_ENT;i++)
{
 fscanf(arq_ent,"%f",&teste.entrada[i]);


 /*
     Caso esta entrada esteja ativa, informa o índice deste neurônio e incrementa o contador de
     alarmes ativados.
 */
 if ( teste.entrada[i] == 1)
 {
  Alarmes_ativados[Num_Alarmes_ativados] = i;
  Num_Alarmes_ativados = Num_Alarmes_ativados + 1;
 }
}

/* Lê os valores desejados para os neurônios de saída */
for(i=0; i<NUM_MAX_SAI;i++)
{
 fscanf(arq_ent,"%f",&teste.saida_desejada[i]);
 /*
     Caso este valor desejado seja igual a 1, informa o índice do neurônio de saída correspondente e
     incrementa o contador de falhas desejadas.
 */
 if ( teste.saida_desejada[i] == 1)
 {
  Falha_desejada[Num_Falha_desejada] = i;
  Num_Falha_desejada = Num_Falha_desejada + 1;
 }
}

/* Lê os valores dos neurônios de saída */
for(i=0; i<NUM_MAX_SAI;i++)
{
 fscanf(arq_ent,"%f",&teste.saida_real[i]);
 /*
     Caso esta saída esteja com seu  valor entre 1 e 0.65, devemos classificá-la como uma falha
     prevista, informando o índice deste neurônio e incrementa o contador de falhas previstas.
     Devemos ressaltar que classificamos de falha prevista, as falha que a rede neural tem certeza
     que ocorreram.
 */
 if ( ( teste.saida_real[i] <= 1) && (teste.saida_real[i] >= 0.65) )
 {
  Falha_prevista[Num_Falha_prevista].indice = i;
  Falha_prevista[Num_Falha_prevista].valor = teste.saida_real[i];
  Num_Falha_prevista = Num_Falha_prevista + 1;
 }
 else
 {

  /*
     Caso esta saída esteja com seu valor entre 0.65 e 0.35, devemos classificá-la como uma falha
     provável, informando o índice deste neurônio e incrementa o contador de falhas prováveis.
```

```
      Devemos ressaltar que classificamos de falha provável, as falha que a rede neural não tem uma
      convicção total de que ocorreram.
   */
   if ( ( teste.saida_real[i] < 0.65) && (teste.saida_real[i] >= 0.35) )
   {
    Falha_provavel[Num_Falha_provavel].indice = i;
    Falha_provavel[Num_Falha_provavel].valor = teste.saida_real[i];
    Num_Falha_provavel = Num_Falha_provavel + 1;
   }
   else
   {
    /*
      Caso esta saída esteja com seu valor entre 0.35 e 0.10, devemos classificá-la apenas e como um
      indício de falha, informando o índice deste neurônio e incrementa o contador de indício de
      falhas.
    */
    if ( ( teste.saida_real[i] < 0.35) && (teste.saida_real[i] >= 0.1) )
    {
     Indicio_de_falha[Num_Indicio_de_falha].indice = i;
     Indicio_de_falha[Num_Indicio_de_falha].valor = teste.saida_real[i];
     Num_Indicio_de_falha = Num_Indicio_de_falha + 1;
    }
   }
  }
 }

 fscanf(arq_ent,"\n");

}/* Fim da leitura de um teste de diagnóstico */


/*
      Apos a leitura do arquivo de testes, passamos a informar este resultado ao usuário.
*/

/* Informa quais os alarmes que estavam atuados neste tese */
printf(" \n \n  Alarmes Ativados : \n");
for(i=0; i<Num_Alarmes_ativados;i++)
{
  printf(" %s \n" ,alarme[Alarmes_ativados[i]].nome);
}


/* Informa quais as falhas que deveriam ter sido diagnosticadas */
printf(" Falhas Desejadas : \n");
if (Num_Falha_desejada != 0)
{
 for(i=0; i<Num_Falha_desejada;i++)
 {
   printf(" %s \n" ,falha[Falha_desejada[i]].nome);
 }
}
else
{
  printf("  - \n");
}
```

```
/*
      Informa quais as falhas que foram diagnosticadas com certeza pela rede neural.
*/
printf(" Falhas Previstas : \n");
if (Num_Falha_prevista != 0)
{
 for(i=0; i<Num_Falha_prevista;i++)
 {
  printf(" %s : %f\n" ,falha[Falha_prevista[i].indice].nome,
                       Falha_prevista[i].valor );
 }
}
else
{
 printf("  - \n");
}


/*
      Informa quais as falhas que foram diagnosticadas como possíveis pela rede neural.
*/
printf(" Falhas Provaveis : \n");
if (Num_Falha_provavel != 0)
{
 for(i=0; i<Num_Falha_provavel;i++)
 {
  printf(" %s : %f \n" ,
   falha[Falha_provavel[i].indice].nome, Falha_provavel[i].valor);
 }
}
else
{
 printf("  - \n");
}

/*
      Informa quais os indícios de falhas detectados pela rede neural.
*/

printf(" Indicios de Falha : \n");
if (Num_Indicio_de_falha != 0)
{
 for(i=0; i<Num_Indicio_de_falha;i++)
 {
  printf(" %s : %f \n" ,
   falha[Indicio_de_falha[i].indice].nome,Indicio_de_falha[i].valor);
 }
}
else
{
 printf("  - \n");
}


/*

 printf( " Tecle Enter para continuar ");
 getch();
```

```
*/

 } /* Fim da leitura do arquivo */

 fclose(arq_ent);

}/* Fim do programa principal */
```