DIAGNÓSTICO DE ACIDENTES DA USINA NUCLEAR DE ANGRA 2 BASEADO EM AGENTES INTELIGENTES DE AQUISIÇÃO EM TEMPO REAL, E EM UM MODELO DE ÁRVORE LÓGICA

Gustavo Varanda Paiva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Nuclear, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Nuclear.

Orientador: Roberto Schirru

Rio de Janeiro

Fevereiro de 2017

DIAGNÓSTICO DE ACIDENTES DA USINA NUCLEAR DE ANGRA 2 BASEADO EM AGENTES INTELIGENTES DE AQUISIÇÃO EM TEMPO REAL, E EM UM MODELO DE ÁRVORE LÓGICA

Gustavo Varanda Paiva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA NUCLEAR.

Examinada por:

_____
Prof. Roberto Schirru, D.Sc

_____
Prof. Jose Antonio Carlos Canedo Medeiros, D.Sc

_____
Prof. Claudio Marcio de Nascimento Abreu Pereira, D.Sc

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2017

Paiva, Gustavo Varanda

Diagnóstico de Acidentes da Usina Nuclear de Angra 2 Baseado em Agentes Inteligentes de Aquisição em Tempo Real, e em um Modelo de Árvore Lógica / Gustavo Varanda Paiva. – Rio de Janeiro: UFRJ/COPPE, 2017.

XI, 80 p.: il.; 29,7 cm.

Orientador: Roberto Schirru

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia Nuclear, 2017.

Referências Bibliográficas: p. 70-72.

1. Diagnóstico de Acidentes. 2. Sistema Especialista . 3. Usina Nuclear. I. Schirru, Roberto. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Nuclear. III. Título.

Agradecimentos:

A todos da minha família que sempre acreditaram e me apoiaram.

A todos os funcionários que viabilizam meu aprendizado: aos professores dos diversos departamentos da UFRJ envolvidos, aos funcionários que cuidam da limpeza e organização da universidade, sem a qual não possuiríamos um ambiente propício para o desenvolvimento acadêmico. Além dos muitos outros empregados que prestam serviços sem os quais essa meta não poderia ser atingida.

Ao meu orientador, prof. Roberto Schirru, que me incentivou, teve paciência e me ensinou muito.

Aos meus colegas de turma que ao longo do curso sempre me apoiaram.

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

DIAGNÓSTICO DE ACIDENTES DA USINA NUCLEAR DE ANGRA 2 BASEADO EM AGENTES INTELIGENTES DE AQUISIÇÃO EM TEMPO REAL, E EM UM MODELO DE ÁRVORE LÓGICA

Gustavo Varanda Paiva

Fevereiro/2017

Orientadores: Roberto Schirru

Programa: Engenharia Nuclear

Este trabalho possui como objetivo a criação de um modelo, usando a linguagem Python, que com a aplicação de um Sistema Especialista utiliza regras de produção para analisar os dados obtidos em tempo real da usina e auxilie o operador a identificar a ocorrência de transientes/acidentes. Na ocorrência de um transiente o programa alerta o operador e sinaliza qual sessão do Manual do Operador deverá ser consultada para levar de volta a usina ao seu estado normal. A estrutura genérica utilizada para representar o conhecimento do Sistema Especialista foi uma Árvore de Falha e a obtenção dos dados da usina foi realizada por meio de agentes inteligentes que transformam os dados obtidos da usina em valores booleanos usados na Árvore de Falha, inclusive utilizando Lógica Nebulosa.

Para testar a validade do programa foi utilizado um modelo simplificado dos manuais da Central Nuclear Almirante Alvaro Alberto 2 (Angra 2) e com este modelo foram realizadas simulações para analisar o funcionamento do programa e a redução no tempo de identificação de um transiente, quando comparado com métodos manuais. Os resultados dos testes apresentaram significativa redução no tempo e grande acurácia, demonstrando a aplicabilidade do modelo ao problema.

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

ACCIDENT DIAGNOSIS OF THE ANGRA 2 NUCLEAR POWER PLANT BASED ON INTELLIGENT REAL-TIME ACQUISITION AGENTS AND A LOGICAL TREE MODEL

Gustavo Varanda Paiva

February/2017

Advisors: Roberto Schirru

Department: Nuclear Engineering

This work aims to create a model, using the Python language, which with the application of an Expert System uses production rules to analyze the data obtained in real time from the plant and help the operator to identify the occurrence of transients / accidents. In the event of a transient the program alerts the operator and indicates which section of the Operator's Manual should be consulted to bring the plant back to its normal state. The generic structure used to represent the knowledge of the Expert System was a Fault Tree and the data obtained from the plant was done through intelligent agents that transform the data obtained from the plant into Boolean values used in the Fault Tree, including using Fuzzy Logic.

In order to test validate the program, a simplified model of the Almirante Alvaro Alberto 2 Nuclear Power Plant (Angra 2) manuals was used and with this model, simulations were performed to analyze the program's operation and the reduction in the transient identification time when compared with manual methods. The results of the tests presented significant reduction in the time and great accuracy, demonstrating the applicability of the model to the problem.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

1. Introdução

O uso da energia nuclear com intenção de gerar energia elétrica foi iniciado na década de 1950 e nos dias de hoje, segundo a Agência Internacional de Energia Atômica (IAEA), existem 441 reatores de potência em operação espalhados pelo mundo e estes representam uma geração de aproximadamente 382 MW_e [1]. No Brasil, a geração nuclear é dada por 2 usinas, que somadas geram cerca 2,78% da produção elétrica brasileira [2].

Desde os primórdios da geração nucleoeletrica o fator segurança é algo de extrema importância para as instalações nucleares. Devido ao acúmulo de experiência com o tempo, foram adicionados fatores de segurança, como a adição de sistemas redundantes e de alta confiabilidade, com o intuito de assegurar a integridade da usina.

Na história das usinas nucleares ocorreram alguns acidentes que obtiveram grande repercussão. Alguns exemplos destes acidentes são os ocorridos em *Three Mile Island* (Pennsylvania – EUA), Chernobyl (Pripyat – Ucrânia) e Fukushima Daiichi (Fukushima – Japão). Assim como em outras áreas tecnológicas, através destes acidentes foram obtidos mais conhecimentos que proporcionaram também novas abordagens para aumentar a segurança nas usinas nucleares.

Um dos conhecimentos adquiridos com a experiência na operação foi a necessidade de possuir uma equipe de operadores qualificados junto a criação de formas eficazes de interação homem-maquina para aumentar a eficiência e segurança das usinas e com isso, reduzir o estresse sobre os operadores.

Com os conhecimentos adquiridos foram enriquecidos os documentos em que os operadores devem consultar e seguir suas recomendações para cada situação que venha a ocorrer. Como exemplo de um destes documentos pode-se citar o Manual de Operação (MO) que foi atualizado para o formato de colunas onde o manual é dividido em 3 colunas sendo uma para descrever as ações, a segunda descrevendo a resposta esperada ao realizar a ação relacionada, e a ultima descreve o que fazer caso a reação da ação não for igual a resposta esperada. Outro documento importante presente nos manuais de Angra 2 é a Árvore Lógica de Diagnósticos (ALD). Ambos os documentos citados anteriormente terão importância neste trabalho.

A ALD é um conjunto de procedimentos listados no documento de identificação de acidentes presente no MO da usina. Estes procedimentos têm como objetivo auxiliar os operadores na identificação de perturbações ou acidentes internos da usina depois de uma mudança de potência do reator ou uma Iniciação do Desarme do Reator (IDR) e fornecer rapidamente a localização do capítulo do MO que descreve as contramedidas necessárias para lidar com o evento [3].

As motivações para iniciar este trabalho foram: 1) Obter e representar o conhecimento quanto aos sistemas de segurança das usinas nucleares. 2) Pesquisar e desenvolver o uso de softwares de programação na aplicação de técnicas de Inteligência

Artificial (IA). 3) Criar um protótipo de software que poderá ser usado para aumentar o nível de segurança nas usinas nucleares e com isso aumentar a aceitação publica de usinas nucleares no futuro.

De modo mais especifico, o objetivo deste trabalho é de criar um programa que com o uso de IA possibilite automatizar a identificação em tempo real dos estados (acidente / transiente) de uma usina nuclear por meio do uso da ALD e dê suporte ao operador, no caso de mau funcionamento da planta, na identificação de qual capítulo do MO deve ser analisado para fazer com que a usina volte à operação normal.

Para a construção do programa que irá identificar os eventos ocorridos com o uso da ALD, foi utilizado o modelo de Sistema Especialista (SE) que é uma técnica de IA que possui como objetivo obter uma solução por meio de inferência de regras e heurísticas. A estrutura de um SE é composta de uma base de conhecimento onde nesta esta o conhecimento necessário para solucionar o problema analisado e um motor de inferência que obtém os fatos e heurísticas da base de conhecimento e os utiliza para solucionar o problema analisado.

Para representar a base de conhecimento do SE foi utilizada uma estrutura baseada em árvore de falha por ser uma estrutura que permite especial atenção ao pensamento em tempo real. Uma árvore de falha é estruturada por um conjunto de eventos interligados por portões lógicos e possui como objetivo a obtenção do valor lógico de cada evento após a realização das operações lógicas com os valores inicialmente possuídos. Os portões lógicos utilizados neste trabalho incluem "*OR*", "*AND*", "*NOT*", "*NOR*", "*NAND*" e um portão que foi criado para este trabalho e foi

chamado de "*INHIBIT-NONE*". Normalmente usam-se os valores "Verdadeiro" e "Falso" para definir o estado de um evento, porém, devido à natureza em tempo real deste trabalho, será utilizado o valor "*none*" para indicar que não foi possível definir o valor do evento, onde um exemplo para este caso seria a falha de um sensor. O uso do valor "*none*" também é devido a importância do sistema responder "não sei" quando não for possível dar um diagnóstico preciso do estado da usina e um diagnóstico errado poderia agravar a situação da usina.

Para a criação do programa foi utilizada a linguagem Python que foi selecionada por ser uma linguagem de programação de alto nível, disponível gratuitamente e por possuir características que facilitaram a criação do programa, como a recursividade e a utilização de lista.

Na obtenção dos dados de entrada foram utilizados agentes inteligentes de aquisição para tratar os dados de entrada de forma que estes possam ser usados para definir qual o estado que se encontra as premissas das regras da árvore de falha. Agentes inteligentes de aquisição são programas capazes de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por meio de atuadores [4]. Neste trabalho foi vinculado um agente inteligente para cada tipo de aquisição de dados e com isso, foi usado um modelo de multi-agente onde os agentes possuem a capacidade de interagir entre si para solucionar um problema.

Devido à presença de valores subjetivos nas regras do sistema, por exemplo: "Diminuindo", "Muito Alta", foi necessário utilizar de meios que possam interpretar estas palavras. A forma utilizada para isto foi o uso da Lógica Nebulosa. Diferente da

Lógica Booleana, onde os valores lógicos são 0 ou 1, na Lógica Nebulosa os dados são tratados com um grau de verdade onde este esta compreendido entre 0 e 1. Portanto, para um determinado conjunto existe um grau de verdade, se um elemento pertence ao conjunto ou não. Este grau de verdade é definido por uma função.

Para transmitir as informações produzidas pelo programa e permitir que o usuário tenha a capacidade de interagir com o programa, foi criado uma interface gráfica utilizando o modulo Tkinter disponível na linguagem Python.

Nos últimos anos foram publicados diversos trabalhos que utilizam de técnicas de IA, como SE, redes neurais artificiais, algoritmo genético e lógica nebulosa para solucionar problemas relacionados à monitoração e diagnósticos de falhas de usinas nucleares.

Um dos problemas constantemente analisado é a identificação de transientes por meio da analise de padrões das variáveis da usina. Um exemplo de trabalho relacionado à identificação de transientes em centrais nucleares foi feito em 1992 por BARTLETT e UHRIG [5] onde eles utilizaram uma rede neural artificial no diagnóstico de 7 cenários de uma usina nuclear. Para a tomada de dados de entrada foi utilizado uma rede multicamadas perceptron que recebia os valores de 27 variáveis a cada 0,5 segundos por um período de 250 segundos. Os resultados do trabalho foram satisfatórios, porém a analise não constava com a capacidade dar como resposta “não sei” em casos que o evento analisado esteja fora do escopo de treinamento do sistema.

Utilizando de uma rede neural probabilística, BARTAL et al [6] criaram um modelo capaz de identificar 72 casos relacionados a 13 transientes diferentes onde para isso usaram como dados de entrada 76 variáveis do sistema. Neste trabalho o sistema consta com a capacidade de responder "não sei" no caso em que o evento analisado esteja fora do escopo do sistema. Outro diferencial deste trabalho é o uso de um mecanismo chamado de "acumulação de evidência" que faz com que os resultados das classificações feitas em intervalos de tempo anteriores sejam usados para ajudar na classificação final. Com esse método, o classificador neural continua trabalhando independente do tempo, mas a classificação final é computada usando os valores obtidos a cada instante de tempo.

Utilizando redes neurais artificiais, lógica nebulosa e algoritmos genéticos, ALVARENGA et al [7] criaram um trabalho para diagnosticar 16 acidentes postulados da usina nuclear Angra 2. Em seu trabalho a rede neural artificial do tipo *Adaptive Vector Quantization* gera os centroides para cada acidente analisado e em seguida, usa-se lógica nebulosa para definir a zona de influência de cada acidente e por fim, usa-se o algoritmo genético para definir a posição dos centroides no eixo do tempo.

Rafael Gomes [8] criou um protótipo que utiliza modelagem Neuro-Fuzzy na identificação de transientes em usinas nucleares. Seu modelo é constituído de 2 níveis, onde no primeiro nível é utilizado redes neurais artificiais e no segundo nível é utilizado Lógica Nebulosa.

Douglas Salmon [9] utilizou um SE baseado em regras para a identificação de transientes em usinas nucleares por meio da ALD. Em seu trabalho ele utiliza de linguagem proposicional para a criação das regras do sistema.

ALLALOU et al [10] publicaram em 2016 um trabalho sobre um sistema de monitoração e diagnóstico para o reator de pesquisa Nur. Neste sistema são utilizadas redes neurais artificiais onde foram usados os modelos de funções de bases radiais e o modelo de múltiplas camadas perceptrons. O trabalho analisa os cenários de acidente devido a inserção de reatividade, blackout e acidente por perda de fluxo e para isso são analisadas 5 variáveis. Segundo o trabalho os melhores resultados de classificação foram obtidos ao usar o modelo de múltiplas camadas perceptrons ajustados pelo algoritmo de *backpropagation* resiliente.

Como exemplo do uso de agentes inteligentes na área nuclear existe o trabalho publicado por Robert E. Uhrig e Lefteri H. Tsoukalas [11], onde o objetivo deste trabalho foi criar um programa que crie e teste uma nova abordagem de controle antecipatório baseado em um modelo multi-agente para melhorar a segurança e o desempenho das usinas nucleares da geração IV durante a operação semiautônoma de longo prazo. Para o programa foram criados agentes que utilizam o software chamado Legacy para realizar funções especificas de monitoração e relacionadas à garantia da segurança do sistema.

A estrutura deste trabalho foi realizada de forma que no capítulo 1 seja demonstrado um breve histórico sobre a produção de energia nuclear no mundo, em seguida é apresentado o objetivo do trabalho junto das motivações para sua realização e

por fim é feita uma introdução aos tópicos que serão apresentados nos capítulos posteriores e são citados trabalhos que compartilham do uso de técnicas de inteligência artificial no diagnóstico como tema.

São descritos no capítulo 2 os conceitos que serão utilizados para a realização deste trabalho junto da explicação de cada um destes conceitos.

No capítulo 3 é feita uma introdução sobre a ALD onde é descrito seus objetivos, características e estrutura e em seguida é apresentada a metodologia que foi utilizada para a criação deste trabalho, onde é mostrado os processos de criação do trabalho foi dividido em 4 etapas onde as 3 primeiras falam sobre a função e características dos programas usado no trabalho e a ultima é a realização das simulações para testar o programa. No capítulo 4 descreve como foi o processo de criação dos 3 programas utilizados no trabalho.

O capítulo 5 apresenta como foi feito os testes e simulações dos programas e quais foram os resultados dos testes realizados. No capítulo 6 são feitas as conclusões sobre o trabalho.

CAPÍTULO 2

2. Fundamentação Teórica

Nesta sessão, será feita uma introdução ao conteúdo necessário para fundamentar o trabalho feito nesta dissertação.

Devido ao uso de SE neste trabalho, inicialmente será explicado o que é a IA para em seguida explicar a ideia de SE.

Para representar as regras presentes na ALD será utilizada uma árvore de falha e, com isso, será necessária a compreensão da estrutura de uma árvore de falha e das operações lógicas presentes nesta estrutura.

Em seguida, serão apresentados os conceitos de Lógica Nebulosa e por fim serão introduzidos conceitos sobre a linguagem de programação Python.

2.1. Inteligência Artificial

A criação da IA ocorreu devido à curiosidade do ser humano em criar uma forma de processamento que se assemelhasse com a inteligência dos seres vivos. Com a evolução na tecnologia dos computadores, pesquisas relacionadas à IA começaram a surgir e no ano de 1956 o termo IA começou a ser usado [4].

A IA pode ser utilizada para resolução de problemas de lógica, equações matemáticas, criação de sistemas de jogos, reconhecimento de padrões, diagnósticos, etc.

Foi proposta por Alan Turing uma forma de identificar se um programa possui ou não IA. O teste foi nomeado como Teste de Turing [4]. O programa a ser testado deveria ser interrogado por uma pessoa que não possuía informação se ela estava interrogando uma máquina ou um ser humano. Para o programa testado ser considerado uma IA, a pessoa que estava interrogando deveria ser incapaz de afirmar se o interrogado era outra pessoa ou uma máquina se baseando apenas nas respostas concedidas pelo interrogado. São exemplos de IA: Sistemas Especialistas, Rede Neurais, Algoritmos Genéticos, etc.

2.2. Sistema Especialista

Sistema Especialista é um exemplo de técnica de IA que simula a forma de resolução de problemas utilizada pelos seres humanos.

Segundo Emmanuel Lopes Passos em seu livro "Inteligência Artificial e Sistemas Especialistas Ao Alcance de Todos" [12], a aplicação de sistemas especialistas se torna vantajosa quando comparadas aos convencionais, nos seguintes casos:

- Casos com um grande número de combinações que necessitam de muito tempo para que todas as opções sejam avaliadas;

- Casos onde os processos requerem grandes quantidades de dados ou que haja a necessidade de uso e recuperação da informação com rapidez.

Segundo Robert Keller [13], a estrutura básica de um SE possui 3 componentes: A interface gráfica com o usuário, que informa ao usuário os eventos que estão ocorrendo no sistema e possibilita a interação do usuário com o sistema; a Base de Conhecimento, que contém os dados fornecidos pelo especialista e os dados obtidos do sistema para que seja possível solucionar o problema e, por fim, o Motor de Inferência que é o responsável por selecionar os dados da base de conhecimento e realizar operações com o intuito de obter uma solução.. A Figura 1 representa um exemplo de estrutura de um SE.

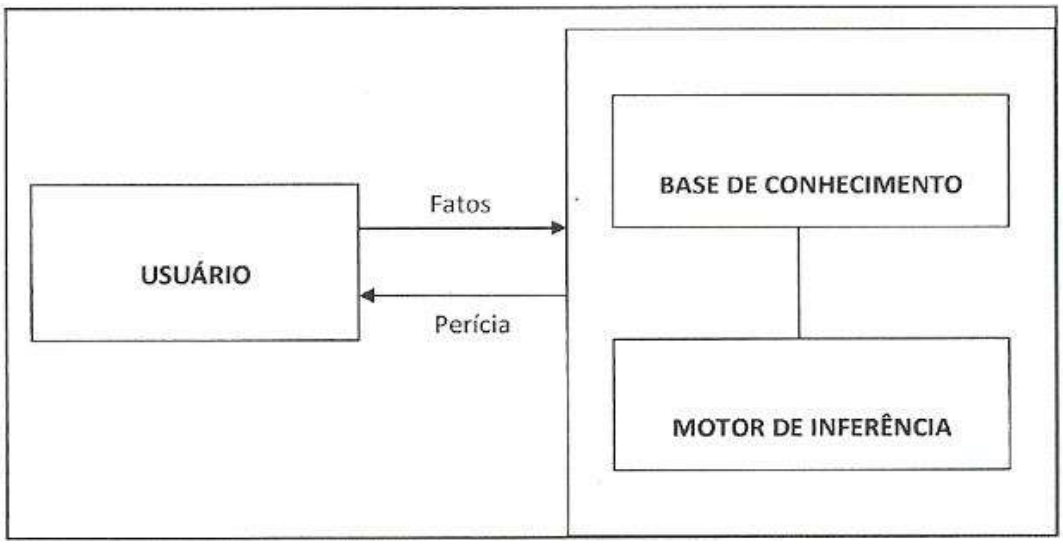


Figura 1: Estrutura de um Sistema Especialista.

Algumas das características relevantes de um SE são:

- Divisão entre Base de Conhecimento e Motor de Inferência. Esta característica possibilita uma facilidade na alteração dos dados de entrada do

programa sem que seja necessária uma alteração no código que processa os dados, dando ao SE uma versatilidade quanto a sua aplicabilidade;

- Capacidade de apresentar o caminho que foi feito para se chegar a uma resposta. Com esta característica o usuário tem a possibilidade de identificar todo o processo de tomada de decisão que foi feito até se chegar ao resultado final.

2.2.1. Base de Conhecimento

Na criação da Base de Conhecimento de um SE é necessário decidir como será feita a representação do conhecimento. Algumas das formas de representar são: regras, rede semântica, frames e orientação a objeto. Neste trabalho foi utilizado um modelo de regras representado no formato de uma árvore de falha.

O modelo por regra é o mais comum para representar o conhecimento por se assemelhar com a forma que os especialistas aplicam seus conhecimentos para resolver um problema. Este método se baseia no conceito de *modus ponens* onde uma série de condições é descrita e quando são atendidas apresentam uma consequência que também será atendida, porém, isto não implica que quando a consequência de uma regra for atendida, as condições também serão. Este formato é exemplificado por:

If (Se) <Condição> -> *Then* (Então) <Consequência>
Exemplo: *If* A = Verdadeiro → *Then* B = Falso

2.2.2. Motor de Inferência

O Motor de Inferência é o elemento responsável pela execução das regras. Ao criar um SE é necessário tomar a decisão de quais serão as características do Motor de Inferência. É papel do criador do programa decidir quais características se aplicam melhor ao sistema em que o SE será usado e esta decisão determina a eficiência do SE. Algumas das características a serem decididas é o modelo de encadeamento e um modelo que defina qual será a ordem de execução das regras.

2.2.2.1. Modelo de Encadeamento

Encadeamento é a forma a qual o sistema irá executar as regras. Dois exemplos de encadeamento são: Encadeamento para frente (*foward chaining*) e encadeamento para trás (*backward chaining*).

No encadeamento para frente, a solução é alcançada ao se aplicar as regras partindo dos fatos iniciais e com isso, ao realizar as regras, novos fatos são criados até que todas as regras que forem aptas a serem executadas sejam executadas. Na Figura 2 são apresentadas as etapas presentes no encadeamento para frente.

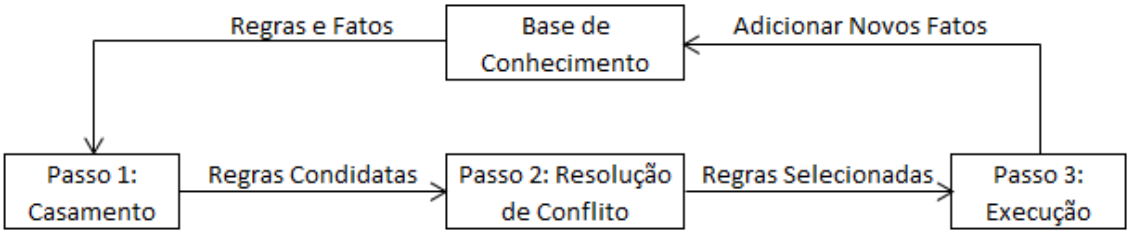


Figura 2: Processo de Inferência do Encadeamento para Frente

São 3 as etapas do modelo de encadeamento para frente: 1) Casamento, 2) Resolução de conflitos, 3) Execução.

1. Casamento: Nesta etapa são verificadas quais regras possuem todas suas condições na Base de Fatos. As que possuírem todas as condições serão candidatas para o passo seguinte.
2. Resolução de Conflitos: Devido à possibilidade de mais de uma regra passar pela fase de casamento, se torna necessário que exista um modelo para determinar qual será a regra escolhida para execução. A escolha do modelo a ser adotado influencia a eficiência do SE. Alguns dos critérios usados são: Regência (Dar preferência a regras que referem a fatos criados recentemente), Especificidade (Dar preferência a regras mais especificas), Hierarquização (Dar preferência a regras com maior prioridade, seguindo uma ordem previamente definida) [4].
3. Execução: Nesta etapa é executada a regra selecionada e com isso pode ser gerados novos fatos que serão adicionados a Base de Fatos. Outro ciclo recomeça após o termino da terceira etapa.

No encadeamento para trás é pressuposto uma solução e deve-se provar esta hipótese com os fatos do sistema. Nas suas interações este modelo possui etapas semelhantes ao do encadeamento para frente onde são selecionadas as regras que possuam como consequência algum dos indivíduos presentes na Base de Fatos, mas com o objetivo de mostrar que um fato (objetivo) hierarquicamente superior é verdadeiro ou falso. Em seguida é feito a resolução de conflitos e por fim as condições da regra escolhida são adicionadas a Base de Fatos.

A Figura 3 demonstra um exemplo do encadeamento para frente (à esquerda) e do encadeamento para trás (à direita) onde foram usadas as seguintes regras: Se B e C então A; Se D então B; Se E e F então C.

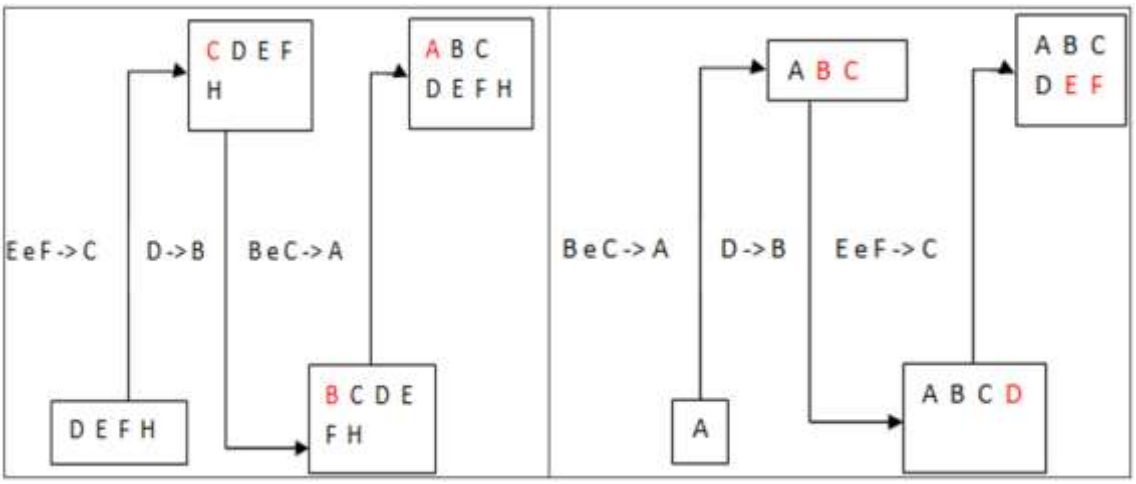


Figura 3: Exemplo de Encadeamento para Frente (à esquerda) e Encadeamento para Trás (à direita).

2.3. Árvore de Falha

O uso de uma árvore de falha possibilita a representação gráfica de um sistema de regras. Na Figura 4 é dado um exemplo de árvore de falha.

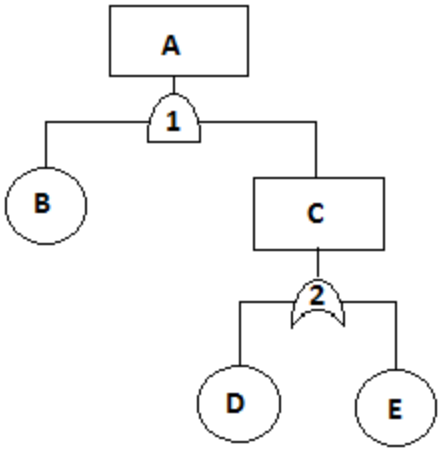


Figura 4: Exemplo de Árvore de Falha

É comum o uso de árvore de falha na análise de risco de um sistema onde geralmente o número de elementos é elevado e estes elementos se relacionam com operações lógicas. Alguns dos benefícios de usar uma árvore de falha são os seguintes:

- Representação gráfica de uma cadeia de eventos
- Identificação dos pontos críticos da cadeia de eventos
- Análise qualitativa e/ou quantitativa do sistema

É possível obter dois tipos de resultados ao analisar uma árvore de falha e estes são: qualitativos ou quantitativos. Os resultados qualitativos incluem: Os cortes mínimos da árvore, análise qualitativa da importância dos componentes e identificação dos cortes mínimos com possibilidade de falhas de causa comum. Para a obtenção de resultados quantitativos, é necessário saber as probabilidades de falha dos componentes da árvore, desta forma se torna possível a obtenção da probabilidade de falha do sistema e obtenção da análise de importância dos cortes mínimos [14].

A estrutura de uma árvore de falha é composta por Eventos e Portões Lógicos. Eventos são os elementos do sistema analisado e estes podem apresentar valores lógicos como “verdadeiro” e “falso”. Na Figura 4 os elementos definidos como evento foram nomeados com as letras do alfabeto. Portões Lógicos são estruturas que conectam os eventos de forma que um portão sempre possuirá um evento de saída e um ou mais eventos de entrada. Seu papel é realizar uma operação lógica usando os eventos de entrada e retornar o resultado para o evento de saída. Os portões lógicos presentes na Figura 4 foram nomeados com números. É comum nomear os eventos que não são

evento de saída de nenhum portão lógico como "*leaf*". Na Figura 4 são eventos "*leaf*" os eventos B, D e E.

2.3.1. Portões Lógicos e Operações Lógicas

Como dito anteriormente, portões lógicos são elementos de uma árvore de falha que conectam os eventos. A Figura 5 contem os símbolos utilizados para representar os portões lógicos mais utilizados e em seguida são apresentadas as funções de cada um dos citados.

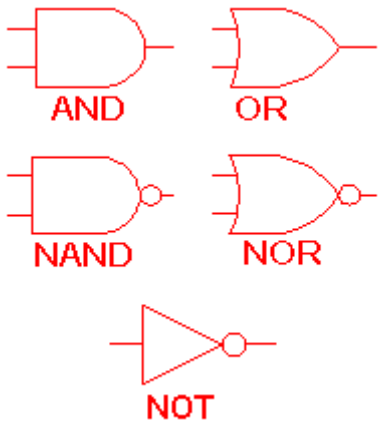


Figura 5: Simbologia e Nomenclatura dos Portões Lógicos.

- Portão não (*NOT*): O evento de saída é atendido se o evento de entrada não for atendido.
- Portão "E" (*AND*): O evento de saída é atendido se todos os eventos de entrada forem atendidos.
- Portão "OU" (*OR*): O evento de saída é atendido se pelo menos um evento de entrada for atendido.

- Portão Não “E” (*NAND*): O evento de saída é atendido se um ou mais eventos de entrada não forem atendidos
- Portão Não “OU” (*NOR*): O evento de saída é atendido se todos os eventos de entrada não forem atendidos

Na Tabela 1 são apresentadas as tabelas verdade das operações citadas acima onde 1 representa “verdadeiro” e 0 representa “falso”.

Tabela 1: Tabela verdade das seguintes operações lógica: *AND, OR, NAND, NOR, NOT*.

AND			OR			NOT	
A	B	X	A	B	X	A	X
1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1
0	1	0	0	1	1		
0	0	0	0	0	0		
NAND			NOR				
A	B	X	A	B	X		
1	1	0	1	1	0		
1	0	1	1	0	0		
0	1	1	0	1	0		
0	0	1	0	0	1		

2.3.2. Modelos de Busca

Ao executar as operações lógicas de uma árvore de falha é preciso decidir qual modelo de busca será usado. Por exemplo: Busca em Profundidade ou Busca em Amplitude.

Uma busca em profundidade ocorre quando: “Dentre todos os nodos marcados (nós com chances de serem os próximos a ser executados) e incidentes a alguma aresta (responsável por conectar os nodos) ainda não explorada, escolher aquele mais recentemente alcançado na busca” [15]. A Figura 6 demonstra um exemplo para este modo.

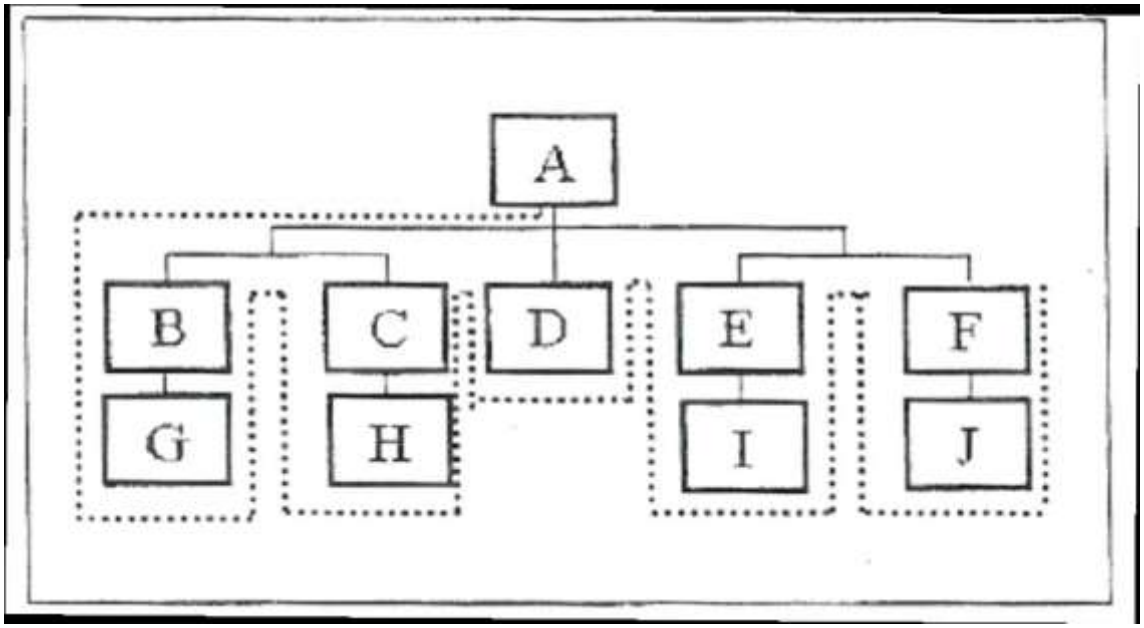


Figura 6: Exemplo de Busca em Profundidade. [15]

Uma busca em amplitude ocorre quando: “Dentre todos os nodos marcados (nós com chances de serem os próximos a ser executados) e incidentes a uma aresta (responsável por conectar os nodos) ainda não explorada, escolher aquele menos recentemente alcançado na busca” [15]. A Figura 7 demonstra um exemplo para este modo.

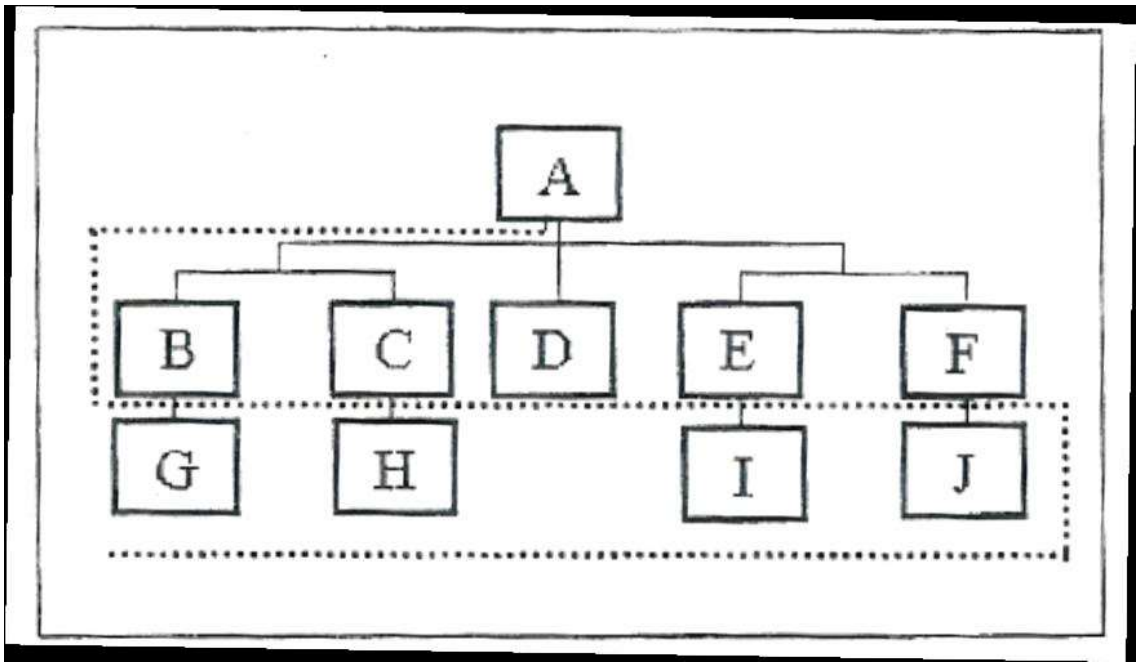


Figura 7: Exemplo de Busca em Amplitude. [15]

2.4. Lógica Nebulosa

Devido a presença de palavras subjetivas como “aumentando” e “quente” nas regras da ALD, usou-se neste trabalho a Lógica Nebulosa para interpretar os valores obtidos de entrada e definir com qual grau de certeza pode-se dizer que uma regra que contenha essas palavras é verdadeira ou falsa

As primeiras ideais relacionada à Lógica Nebulosa surgiram em 1920, por Jan Lukasiewicz, que passou do conceito de 0 ou 1 para o conceito de graus de pertinência sendo 0, ½ e 1 e mais tarde expandiu para um número infinito de valores no intervalo de 0 a 1 [16]. O termo “Lógica Nebulosa” foi criado por Lotfi Asker Zadeh em 1965 [17].

2.4.1. Diferenças entre a Lógica Clássica e a Lógica Nebulosa

Com o intuito de explicar as diferenças entre a Lógica Clássica e a Lógica Nebulosa será utilizada a Figura 8 como exemplo. A Figura 8 apresenta um exemplo de três funções que classificam se uma pessoa é baixa, média ou alta, levando em conta a altura da pessoa. O nome dado a estas funções é Função de Pertinência. Em uma Função de Pertinência, o eixo das abscissas representa o parâmetro analisado, onde neste caso é a altura em centímetros, e o eixo das ordenadas representa o grau de pertinência (μ) onde este é limitado entre os valores 0 e 1. A função de pertinência pode apresentar qualquer formato, porém, neste trabalho foi considerado que elas teriam formato triangular ou trapezoidal. Ao lado esquerdo é apresentada a representação pela Lógica Clássica e à direita pela Lógica Nebulosa.

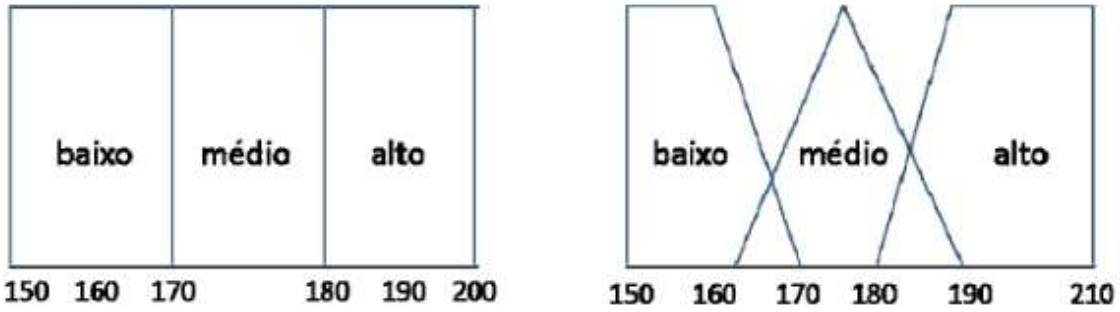


Figura 8: Função de pertinência para determinar a altura de uma pessoa. À esquerda: Lógica Clássica. À direita: Lógica Nebulosa [18].

Pela Lógica Clássica só existem dois possíveis valores em uma Função de Pertinência sendo estes 0 (representando a certeza que o elemento não pertence ao conjunto) e 1 (representando a certeza de que o elemento pertence ao conjunto). Outra característica da Lógica Clássica é que para cada valor no eixo das abscissas existe apenas uma Função de Pertinência que possui valor diferente de 0.

Uma forma de exemplificar a Lógica Clássica é dada adiante: Segundo o gráfico à esquerda na Figura 8, uma pessoa de 181 cm é classificada exclusivamente como alta, possuindo então um grau de pertinência igual a 1 para esta classificação e 0 para as demais.

Na Lógica Nebulosa a Função de Pertinência pode possuir qualquer valor entre 0 e 1 de forma a representar o grau de certeza ao classificar o elemento ao conjunto da função analisada. Na Lógica Nebulosa, um elemento pode pertencer a várias classificações, porém, com diferentes graus de pertinência onde a soma deste deverá totalizar em 1.

Um Exemplo de análise utilizando Lógica Nebulosa é dado a seguir: No gráfico à direita na Figura 8, uma pessoa de 181 cm é classificada como alta com um grau de pertinência de aproximadamente 10% e classificada como média com um grau de pertinência de aproximadamente 90%.

2.4.2. Aplicação da Lógica Nebulosa

Segundo o livro “Fuzzy Control” [19] o processo de aplicação da Lógica Nebulosa consiste em 3 etapas na seguinte ordem: *Fuzzyficação*, Inferência e *Desfuzzyficação*. A Figura 9 apresenta as etapas citadas anteriormente.

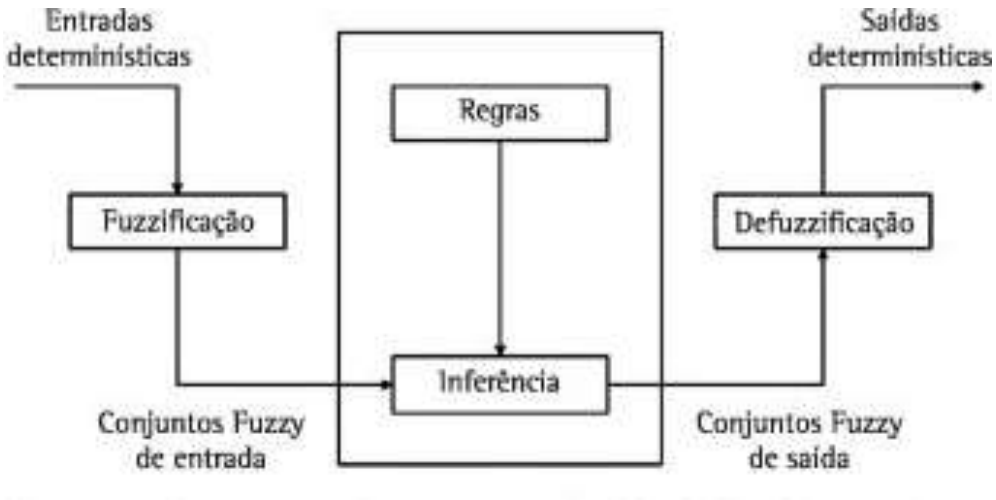


Figura 9: Processo de aplicação da Lógica Nebulosa.

Neste trabalho foi utilizada apenas a etapa de *fuzzyficação* já que o objetivo é verificar em quais grupos semânticos as variáveis de entrada do sistema se enquadra e obter o grau de pertinência da variável para cada um dos grupos. Desta forma neste capítulo será elaborada apenas a etapa de *fuzzyficação*.

A etapa de *fuzzyficação* consiste em transformar o valor numérico obtido como entrada do sistema e avaliar a quais grupos semânticos este valor se enquadra. Para realizar este processo é necessário usar a função de pertinência da variável.

De forma a exemplificar o processo de *fuzzyficação* é dado um exemplo de como o agente inteligente, cujo objetivo de tratar as variáveis que demandam lógica fuzzy, realiza esta etapa.

Supondo que exista uma variável chamada de “u(t)” que deva ser tratada pelo o agente inteligente e esta variável no momento analisado possui o valor de 8 N. Desta forma, o agente inteligente irá usar a função de pertinência para identificar a quais

grupos semânticos ela se enquadra e quais valores de grau de pertinência ela assume. A Figura 10 apresenta a função do caso analisado e sinaliza o valor assumido para a variável.

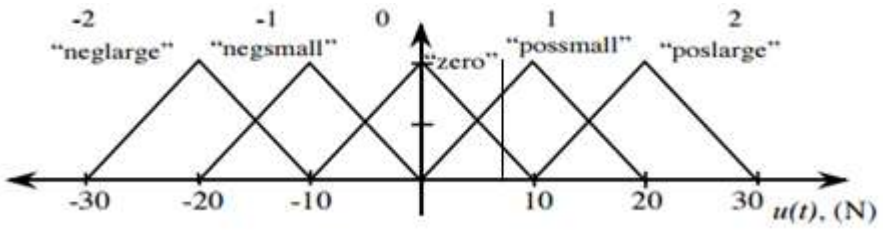


Figura 10: Função de pertinência do exemplo citado. [19]

Observando a Figura 10, pode-se ver que com o valor pré-definido, esta variável pertence ao grupo semântico "zero" com um grau de pertinência de aproximadamente 0,25 e ao grupo "possmall" com um grau de pertinência de aproximadamente 0,75 e os demais grupos possui um grau de pertinência de 0.

2.5. Python

Para a criação do programa foi utilizada a linguagem computacional Python e neste tópico serão explicados os motivos da escolha.

Uma das linguagens usualmente utilizada em programas que utilizam SE é a linguagem LISP. Algumas de suas notáveis características é o uso de listas em sua programação.

Os motivos para a escolha do Python foi a semelhança com a linguagem LISP, a disponibilidade gratuita do software, facilidade em aprender a linguagem e disponibilidade de diversos módulos que adicionam ferramentas ao programa.

Python é uma linguagem computacional com orientação a objeto que possui a característica de ser uma linguagem interpretada, o que aumenta a agilidade no desenvolvimento de programas devido a maior rapidez em depurar, porém, apresenta velocidade de atuação baixa comparada com as linguagens compiladas. Nos dias de hoje, a perda na velocidade de atuação ao se usar uma linguagem interpretada é minimizada devido à alta capacidade computacional dos processadores existentes.

Uma das ferramentas do Python que foi muito utilizada neste trabalho foi o uso de listas. Uma lista é um conjunto de elementos ordenados em forma de vetor, onde estes elementos possuem índex para serem acessados. Em Python uma lista possui a característica de ser fracamente tipificada, o que implica que ao criar uma lista, seus elementos podem possuir características diferentes. Um exemplo para esta característica é que uma lista pode possuir elementos de texto, números, outras listas e etc.

Outro componente que foi utilizado no trabalho foi a capacidade de criar funções recursivas. Funções recursivas são funções que possuem como uma de suas ações a execução da própria função. Funções recursivas possuem potencial de simplificar um programa, porém, devem ser utilizada com cautela, pois podem provocar grande consumo de memória do computador.

Por fim, outro motivo da linguagem Python ter sido escolhida foi devido ao modulo de interface gráfica chamado Tkinter que será utilizado na parte da criação da interface com o usuário do protótipo realizado neste trabalho. O Tkinter conta com uma tela inicial onde pode-se adicionar estruturas chamadas de *widgets* que possuem uma vasta gama de opções, podendo ser estruturas para a escrita de texto, alocação de imagem, botões e outros.

CAPÍTULO 3

3. Metodologia

Nesta seção será explicado o que é a ALD e será descrito o processo de criação do projeto explicando as funções dos programas desenvolvidos.

Para iniciar a criação deste trabalho foi necessário realizar um estudo sobre a ALD com o intuito de identificar seus objetivos e características. Neste estudo foi utilizado como referência o Manual de Operação de Angra 2 [3].

Conforme citado no MO de Angra 2 [3], o objetivo da ALD é "auxiliar o pessoal de operação na identificação de perturbações ou acidentes internos da usina depois de uma mudança não programada de potência do reator ou uma iniciação do desarme do reator (IDR). Ela também permite uma localização rápida do capítulo do MO que descreve as contramedidas necessárias para lidar com o evento".

A estrutura da ALD é dividida nas seguintes sessões:

- Perturbações ou acidentes causando iniciação do desarme do reator (ALD com IDR)
- Perturbações ou acidentes causando apenas variações da potência do reator (ALD sem IDR)

Neste trabalho foi considerada apenas a sessão da ALD com IDR, onde esta contêm todas as perturbações ou acidentes que provocam IDR.

Ao analisar a ALD o operador utiliza os dados da usina para responder uma série de perguntas e com o uso dos resultados, o operador é guiado para o grupo de eventos apropriado e em seguida para o capítulo do MO que deverá ser consultado [3].

Uma característica importante da ALD é a inclusão de uma orientação ao operador no caso de não ser possível identificar com clareza o evento ocorrido. Isto ocorre porque os dados obtidos pelo programa não se enquadram em nenhum dos eventos postulados no MO. A ocorrência deste caso é descrita como a habilidade do programa em dar a resposta "não sei" ao usuário e é de grande importância no diagnóstico em tempo real.

A Figura 11 apresenta um fragmento da ALD com IDR de Angra 2 que será utilizada neste trabalho.

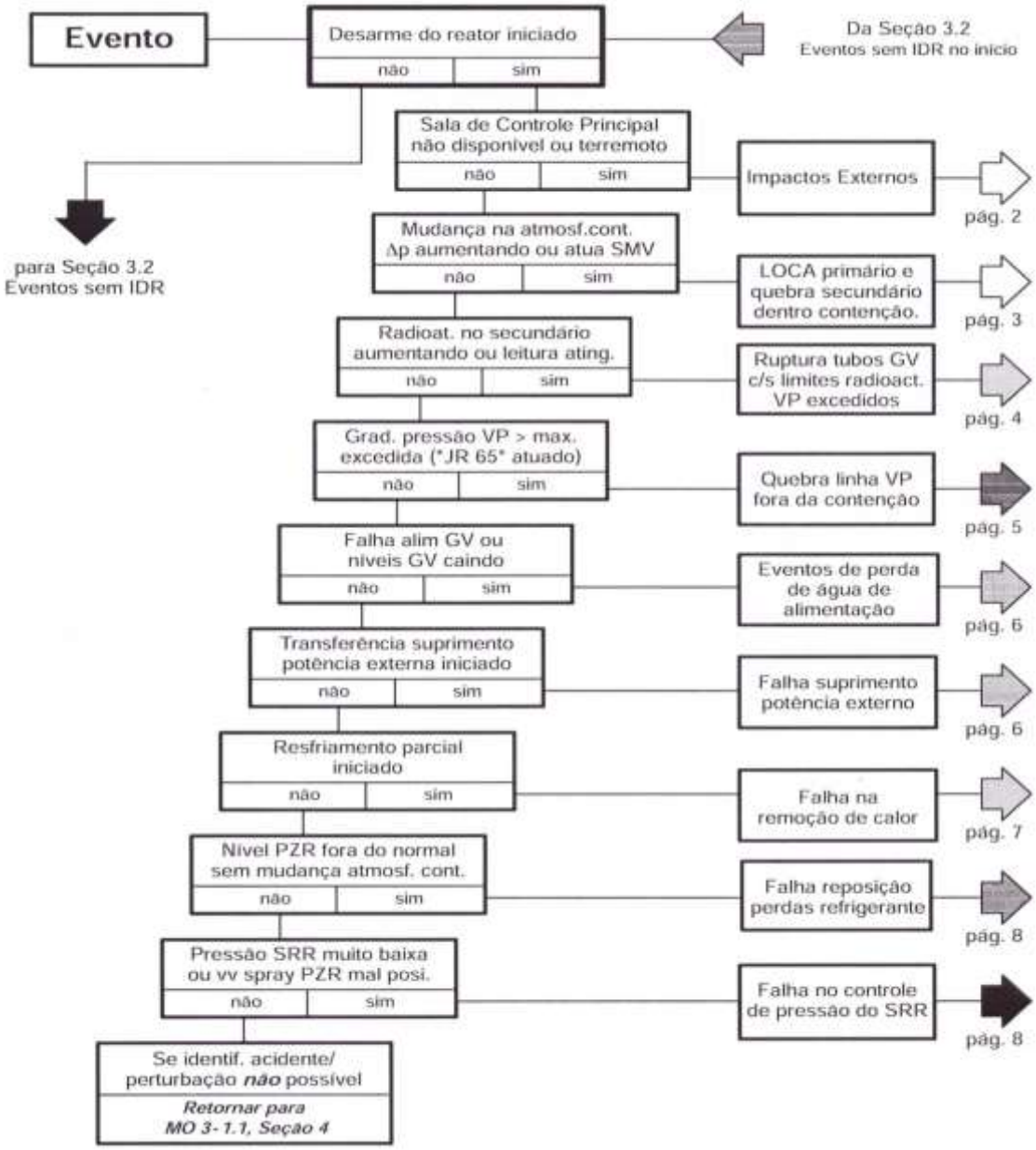


Figura 91: Fragmento da Árvore Lógica de Diagnóstico de Angra 2. Seção de Eventos com IDR [3].

Uma vez realizados os estudos da ALD, a criação deste trabalho foi dividida em 4 etapas, onde estas são: 1) Programa de geração e solução da árvore de falha (PGSAF). 2) Programa de aquisição e manipulação dos dados de entrada por agentes inteligentes (PAMDE). 3) Programa de interface gráfica (PIG). 4) Simulação dos programas criados.

A estrutura do sistema especialista criado é demonstrado na Figura 12 onde esta mostra como os 3 programas se comunicam e como eles interagem com as fontes de dados e com o usuário.

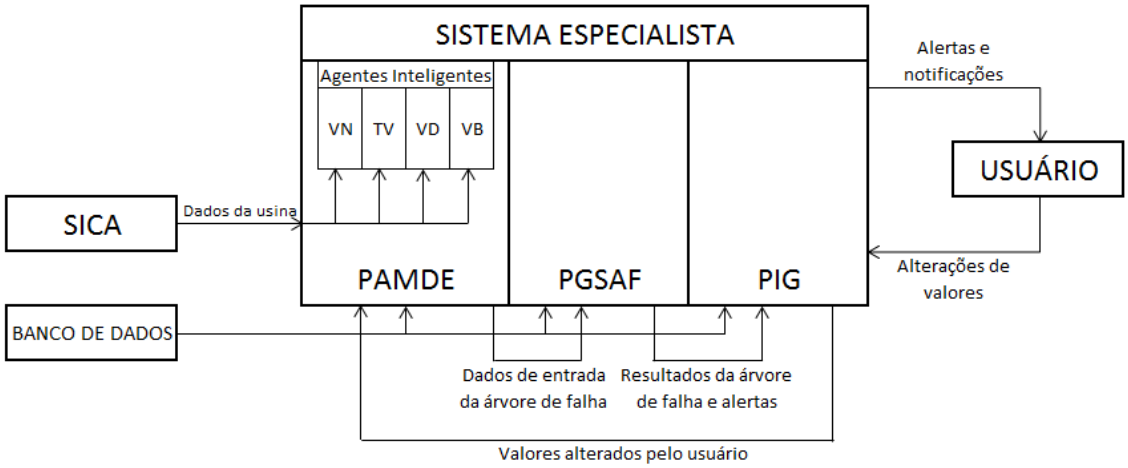


Figura 102: Representação do sistema especialista e suas conexões.

A interação entre os componentes do sistema é dada da seguinte forma: Após o usuário iniciar a tomada de dados pelo PIG o processo de tomada de dados se inicia, onde a cada ciclo de 10 segundos o PIG executa o PGSAF. Uma vez executado, o PGSAF solicita ao PAMDE os valores lógico dos eventos "*leaf*" da árvore de falha e em sua primeira execução também é solicitada a estrutura da árvore de falha a ser analisada. Após ser executado pelo PGSAF, o PAMDE obtêm os dados do Sistema Integrado de Computadores de Angra (SICA) e distribui para os seus agentes inteligentes onde cada agente é designado para tratar um determinado tipo de variável. Com o fim do tratamento dos dados de entrada do SICA, o PAMDE fornece os valores de entrada da árvore de falha para o PGSAF que soluciona a árvore de falha e envia os resultados e os alertas acionados para o PIG para que estes possam ser transmitidos ao usuário. Todos os programas buscam informação em um banco de dados composto por arquivos de texto pré-definidos que contêm informações como a estrutura das mensagens de alerta e

o formato das regras. O usuário ao interagir com a interface gráfica tem a capacidade de alterar os valores dos eventos "*leaf*" da árvore de falha e estas alterações são transmitidas do PIG ao PAMDE para ser usado no ciclo seguinte.

3.1. Programa de Geração e Solução da Árvore de Falha (PGSAF)

Neste programa é realizada a criação e solução da árvore de falha, onde para isso foi necessário o estudo das regras da ALD para que fosse possível organizar estas regras no formato de uma árvore de falha.

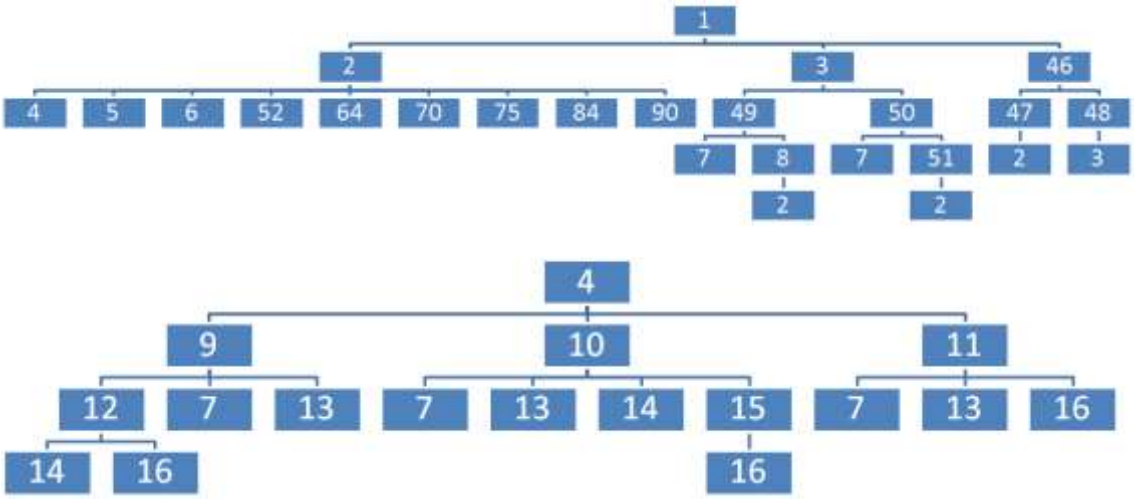


Figura 113: Fragmento da Árvore de Falha utilizada no trabalho.

Um dos objetivos do estudo da ALD foi analisar as regras da ALD com IDR para que fosse possível transformar a árvore lógica, presente no documento, na árvore de falha a ser usada no programa. Na Figura 13 é apresentado dois fragmentos da árvore de falha contendo os evento e suas conexões sem mostrar os portões lógicos. Nos apêndices I, II e III estão representadas todas as informações da árvore de falha. No apêndice I estão listados todos os eventos em uma tabela onde as colunas "*Index*" foram usadas para

enumerar os eventos para que seja possível identifica-los e nas colunas “Nome do Evento” se encontra a descrição do evento como apresentada na ALD. Os eventos que não possuem uma descrição são eventos que tiveram que ser adicionados para que fosse possível a confecção da árvore de falha. No apêndice II estão listados todos os portões lógicos contendo seus *índex* para identificação, eventos de entrada, conector lógico e eventos de saída. O apêndice III apresenta uma forma fragmentada da estrutura da árvore de falha onde não foram adicionados os símbolos que representam os portões lógicos.

Uma vez criada a árvore de falha foi necessário definir como os parâmetros da árvore de falha seriam implementados no programa. O modelo utilizado foi a criação de duas listas com os nomes de Eventos e Gates onde suas funções são respectivamente de armazenar as informações de todos os eventos da árvore de falha e armazenar as informações de todos os seus portões lógicos.

Outro resultado obtido ao analisar a ALD foi a decisão de qual seria o modelo de encadeamento utilizado no programa. O modelo decidido foi o uso do encadeamento para frente devido ao fato de ser a forma mais comum de um ser humano aplicar regras. Como modelo para decidir qual regra será selecionada na etapa de casamento do encadeamento para frente, foi utilizado o modelo de busca em árvore de falha chamado Busca em Amplitude.

A atuação do PGSAF ocorre a cada ciclo de 10 segundos devido ao PIG. Como resposta à sua atuação o programa solicita ao PAMDE os valores dos eventos “*leaf*” da

árvore de falha e em seguida executa os procedimentos para resolvê-la utilizando os modelos de busca e encadeamento definidos.

Para utilizar o modelo de encadeamento para frente e de busca em amplitude foi necessário introduzir o conceito de profundidade do evento. A profundidade do evento indica a posição deste evento na árvore de falha, onde para cada portão lógico o evento de entrada deste portão possui profundidade "n" e os eventos de saída possuem profundidade "n-1". O evento de profundidade 1 é aquele que não é evento de entrada de nenhum portão lógico. A Figura 14 demonstra um exemplo do conceito de profundidade.

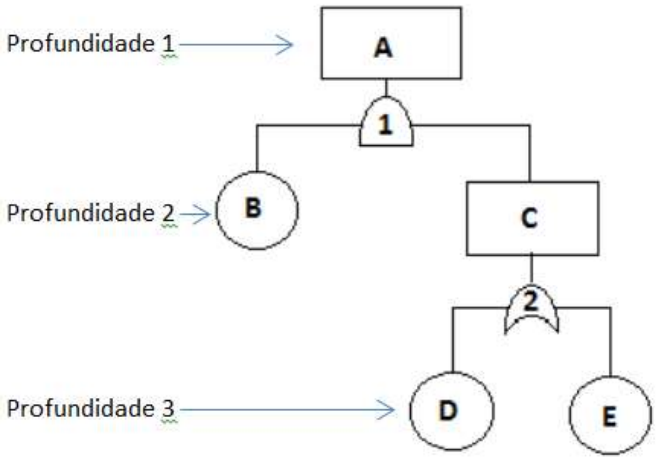


Figura 124: Demonstração do conceito de profundidade em uma árvore de falha

Além do caso em que não é possível definir o evento que esta ocorrendo devido a característica do evento não se assemelhar a nenhum dos eventos postulados, existe o caso onde não é possível definir o evento ocorrido devido a falhas ao obter os dados da usina.

Em programas que trabalham em tempo real, existe a possibilidade de que em um ciclo de obtenção de dados do sistema uma ou mais variáveis não sejam obtidas e estas falhas podem fazer com que seja impossível definir com precisão o estado da usina. Como em sistemas de alta segurança o ato de realizar um diagnóstico errado pode ser extremamente prejudicial, foi necessário adicionar ao programa a possibilidade de dar a resposta "não sei".

Para adicionar esta característica foi necessário adicionar um terceiro valor lógico, para os elementos da árvore de falha, cujo nome é "*none*". Com a adição deste novo valor foi necessário alterar a tabela verdade dos portões lógicos e também foi adicionado um novo portão lógico. A tabela verdade dos portões lógicos utilizado no trabalho considerando a presença do valor "*none*" é apresentada no capítulo 4.

A etapa final do PGSAF é analisar quais alertas foram ativados ao usar os dados da árvore de falha solucionada. Para definir os alertas o programa confere no banco de dados qual são os alertas existentes e suas condições de ativação e ao comparar com a solução da árvore de falha define quais alertas foram ativos. Os alertas ativos devem ser enviados ao PIG e para isso é criado uma lista contendo cada alerta ocorrido e estas listas são armazenados na lista chamada "Resposta". Existem diferentes formatos para

as listas contendo os alertas onde estes formatos são divididos dependendo da característica do alerta e de quais informações devem ser transmitida ao PIG.

3.2. Programa de Aquisição e Manipulação dos Dados de Entrada (PAMDE)

O objetivo deste programa é obter os dados do sistema, utilizar os dados obtidos de forma a definir os valores dos eventos "*leaf*" da árvore de falha e, por fim, formatar os dados de forma que o PGSAF compreenda.

Para garantir a segurança na usina nuclear é esperado que o programa de resolução da ALD seja operado em tempo real. Um sistema em tempo real (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos. O objetivo de um STR é entregar um resultado correto dentro de um prazo pré-definido, onde a falha deste objetivo pode acarretar uma falha temporal.

Uma forma de caracterizar um STR é quanto a sua periodicidade, onde existem as seguintes classificações: Tarefas aperiódicas, onde o processo é desencadeado devido à ocorrência de um evento aleatório ou tarefas periódicas, onde o processo é executado ciclicamente com um período de tempo por ciclo. O modelo usado neste trabalho é o modelo de tarefa periódica onde a cada período de 10 segundos o programa obtém os dados e resolve a árvore de falha.

Após ser ativado pelo PGSAF, o PAMDE consulta arquivos presentes no banco de dados para obter a estrutura das regras presentes em cada evento "*leaf*" da árvore de falha e o formato das funções de pertinência.

Uma regra é composta por uma ou mais premissas de regras que são conectadas por operações lógicas. Para definir o valor lógico do evento relacionado à regra é necessário definir o valor de cada premissa.

Uma premissa é composta por 3 sessões. A primeira sessão determina qual é a variável que será avaliada e se será o valor da variável no ciclo ou sua taxa de variação. Na segunda sessão esta um símbolo representando qual será a operação realizada na premissa. Por fim, na terceira sessão é dado o valor ao qual o valor da variável analisada será comparado. O resultado da operação realizada será o resultado da premissa.

A primeira etapa do processo de solução das regras consiste em ler as premissas da regra para obter quais variáveis devem ser consultadas no arquivo fornecido pelo SICA. Uma vez identificado as variáveis o programa adiciona estas variáveis a uma lista contendo todas as variáveis que serão usadas.

O próximo passo é identificar a qual classe a premissa se enquadra. Neste trabalho existem 4 classes diferentes onde para cada classe esta vinculado um agente inteligente que irá resolver a premissa. Os 4 tipos de classe do trabalho são listados a seguir.

- Variáveis Numéricas: Variáveis que serão utilizadas diretamente como condição de uma regra, sem que seja necessário um tratamento;
    - Exemplo: Temperatura = 45 °C

- Taxa de Variação: Variáveis que dependem do valor de uma variável numérica no ciclo atual e nos anteriores para serem calculadas;
    - Exemplo: Variação da Temperatura = 2 °C/s = $(T_n - T_{n-1})/\Delta t$
- Variáveis Booleanas: Variáveis que apresentam apenas dois valores possíveis;
    - Exemplo: Estado da bomba 1 = "Ligado"
- Variáveis Difusas: Variáveis que demandam da aplicação da lógica difusa para a determinação do valor.
    - Exemplo: Pressão = Alta

Uma vez identificado à quais classes a premissa se enquadra, os agentes inteligentes relacionados as classes identificadas obtêm o valor da variável analisada da lista de variáveis e em seguida realiza o tratamento da variável para que seja possível determinar o valor da premissa. Existe a possibilidade de uma premissa de regra necessitar do tratamento de mais de um agente inteligente. Um exemplo para este caso seria uma regra que pergunta se uma variável esta aumentando, onde neste caso os agentes que fazem o tratamento de variáveis difusas e de taxas são requeridos. Após a resolução de todas as premissas de uma regra o programa determina o valor lógico do evento relacionado à regra.

Quando uma premissa requer uma taxa de variação, o agente inteligente relacionado a este grupo compara os dados dos ciclos anteriores com o atual de forma a calcular a taxa de variação da variável. Caso não seja possível obter os dados dos ciclos anteriores, a condição vinculada a esta taxa será considerada como "*none*" até que seu cálculo seja possível. Devido à inexistência de dados do ciclo anterior, premissas com

esta característica sempre possuem o seu valor como "*none*" na primeira interação do sistema.

Outro caso que requer um tratamento especial é quando ocorre a presença de um valor difuso. Quando isto ocorre, o agente inteligente relacionado a tratar as variáveis difusas realiza o processo de *fuzzyficação* do dado de entrada de forma a verificar qual valor será dado à regra. Ao realizar a *fuzzyficação* o agente inteligente consulta as funções de pertinência, onde estas tiveram suas características previamente definidas por um especialista do sistema analisado.

Com o tratamento dos dados de entrada se torna possível definir o valor lógico dos eventos "*leaf*" da árvore de falha e, por fim, este programa transmite os dados já tratados ao programa que realiza as operações da árvore de falha. Os dados já tratados são inseridos na base de fatos (*working memory*).

Devida a possibilidade do usuário modificar o valor lógico de um evento "*leaf*", o PIG transmite ao PAMDE uma lista contendo quais eventos que tiveram seu valor alterado pelo usuário no ciclo anterior e qual o valor estes eventos devem possuir. Com esta lista, os dados finais a serem transmitidos ao PGSAF são atualizados para conter as mudanças feitas pelo usuário.

3.3. Programa de Interface Gráfica (PIG)

Este programa tem como objetivo criar uma interface gráfica que apresente ao usuário da usina as informações quanto ao estado da usina. Para sua criação foi usado o modulo do Python chamado Tkinter.

A Figura 15 demonstra a janela principal do PIG. Na janela principal à esquerda esta presente uma lista, chamada de Cronograma dos Eventos, que irá listar os alertas que ocorreram na usina junto do momento em que o evento ocorreu. Ao clicar em um dos alertas presentes nesta lista, serão apresentadas na lista à direita, chamada de Detalhamento do Evento, as informações mais detalhadas do alerta. No canto inferior esquerdo da janela principal está presente um botão, nomeado de "*Input*", que irá abrir a janela de input para o usuário e três sinalizadores onde o sinalizador à esquerda mostra se o programa esta operando ou não, o no centro alerta quando houver falha na leitura de algum dado de entrada e o à direita determina se a usina esta em operação normal ou se o desligamento automático do reator (*Trip*) ocorreu.

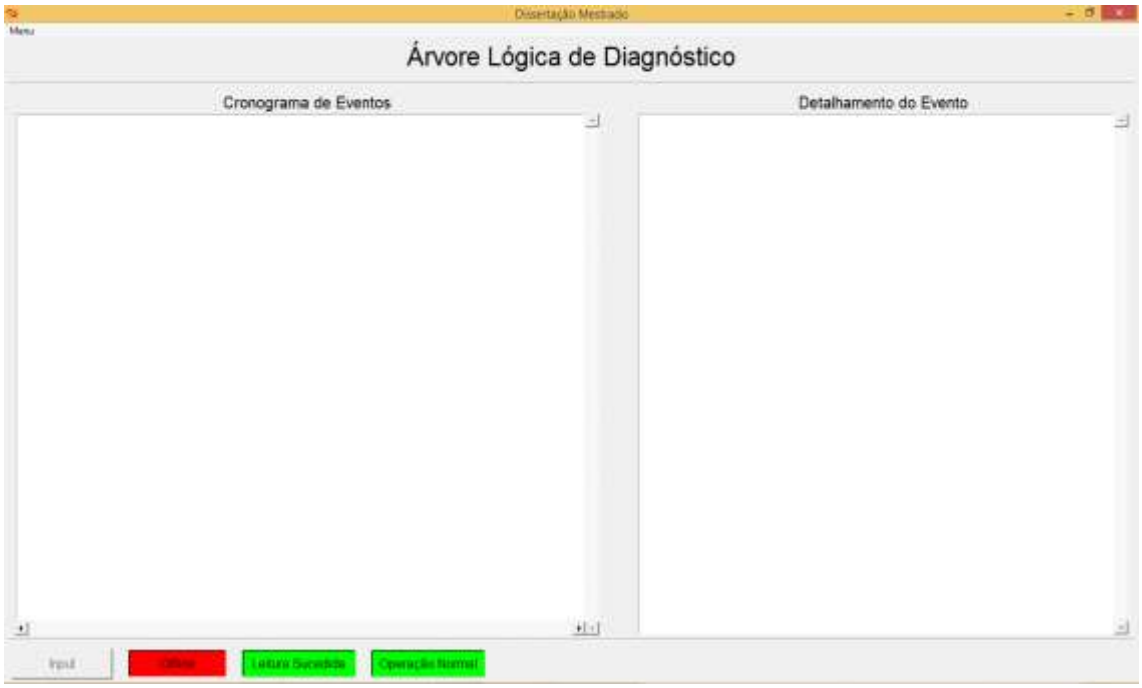


Figura 135: Janela principal do PIG.

Após iniciar o programa o botão Input será desbloqueado e com isso o operador terá acesso à janela de input demonstrada na Figura 16. Na janela de input estão listados todos os eventos de entrada da árvore de falha junto de *checkbuttons* que demonstram o estado do evento no ciclo atual. Ao clicar no nome do evento presente na coluna "*Index* do Evento" uma descrição detalhada sobre o evento será demonstrada na sessão "Descrição do Evento". No canto superior direito está presente um sinalizador que avisa quando ocorre uma leitura dos dados de entrada com alerta escrito "Leitura Realizada" que permanecerá por 1 segundo. No canto inferior esquerdo está presente 2 *checkbuttons* que dão ao usuário a possibilidade de alterar o nível de liberdade que o usuário tem para alterar os valores dos dados de entrada.

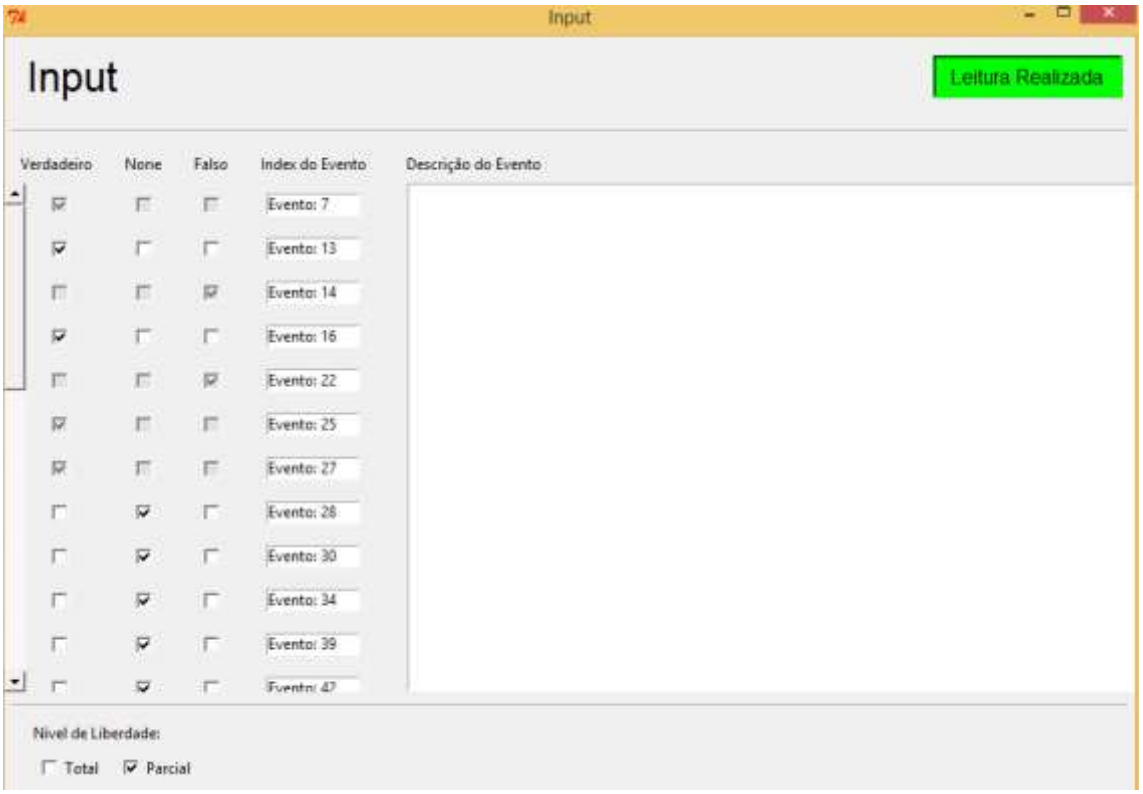


Figura 146: Janela de input do PIG.

Alguns dos dados de entrada do programa devem ser fornecidos pelo o usuário. Devido a isto, é necessário que o programa possua uma forma do usuário impor um valor aos eventos de entrada da árvore de falha e para isso o usuário deve usar os *checkbuttons* vinculados a cada evento. As alterações efetuadas pelo usuário serão enviadas para o PAMDE para serem computadas no próximo ciclo de captura de dados.

Na criação da árvore de falha foi identificado 5 possíveis grupos de respostas ao usuário que poderão ser apresentados na lista de Cronograma dos Eventos. Estes grupos são:

1. Operação normal da usina: Este caso ocorre quando é confirmado que a usina se encontra em operação normal;

2. Ocorrência de um ou mais acidentes: Ocorre quando um ou mais acidentes são confirmados verdadeiros;
3. Falha total na leitura: Ocorre quando acontece a falha na leitura dos dados dos eventos necessários para determinar se ocorreu uma perturbação postulada no projeto ou uma perturbação não postulada;
4. Ocorrência de uma perturbação não postulada no projeto: Neste caso é observada uma anomalia no sistema, porém, não é possível enquadra-la a nenhum dos casos postulados no projeto;
5. Falha na identificação do acidente: Neste caso é identificada uma anomalia no sistema, porém, devido a falta de dados, não foi possível identificar o acidente ou declara-lo fora de projeto.

O PGSAF utiliza a classificação acima para definir quais dados do sistema serão enviados na lista contendo as informações de cada alerta acionado e o PIG obtém estas informações e as formatam utilizando um layout relacionado ao grupo em que o alerta se enquadra.

CAPÍTULO 4

4. Detalhamento dos Programas

No capítulo anterior foram comentadas as funções de cada um dos três programas deste trabalho. Neste capítulo será dada uma explicação mais aprofundada dos programas e como foi feita a criação do programa.

4.1. Programa de Geração e Solução da Árvore de Falha (PGSAF)

Uma vez definida a estrutura da árvore de falha é necessário definir como esta estrutura será representada em forma computacional. Como mencionado no capítulo 3, foi usado uma lista chamada de “Eventos” para armazenar todas as informações dos eventos e uma lista “Gates” para armazenar as informações dos portões lógicos.

A lista Eventos contém uma lista para cada um dos eventos da árvore de falha e cada uma destas listas contém as seguintes informações: *Index* para identificação do evento, descrição do evento, valor lógico do evento e valor booleano. Parte da informação necessária para construir esta lista está presente no arquivo texto chamado de Input1 onde um exemplo para este arquivo é demonstrado na Figura 17. A cada ciclo de tomada de dados a sessão “valor lógico do evento” dos eventos “*leaf*” da árvore de falha são preenchidos com os dados fornecidos pelo PAMDE e os demais são preenchidos com uma string vazia.

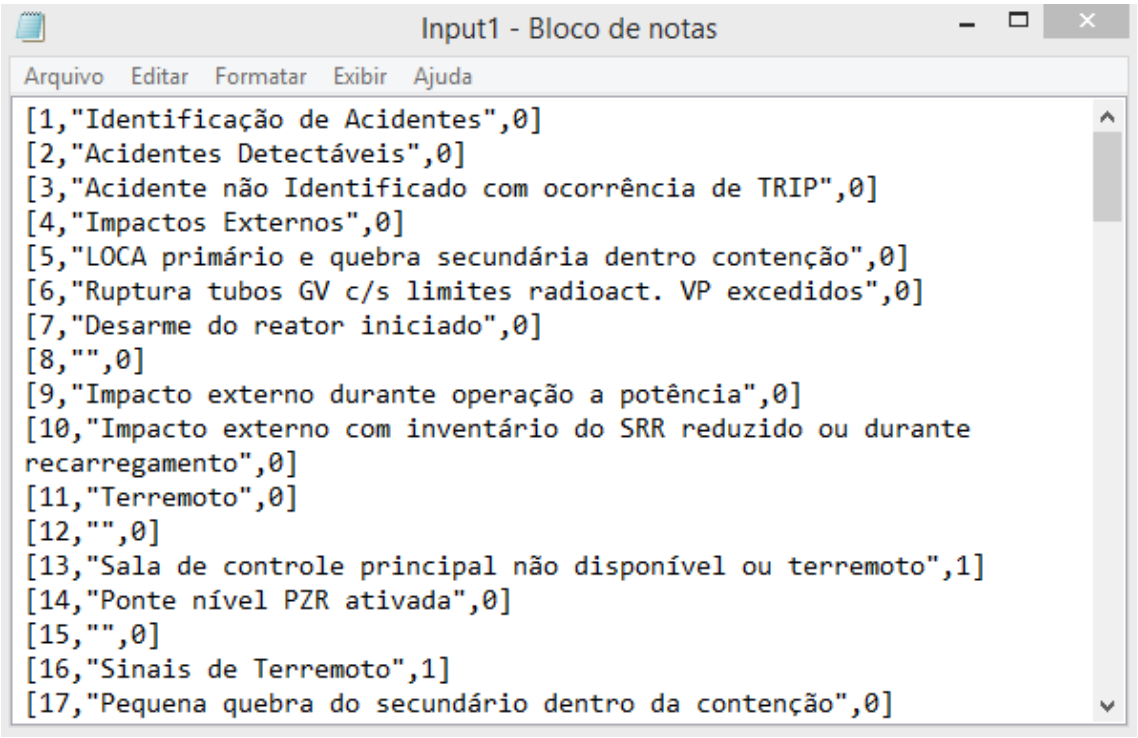


Figura 157: Exemplo de arquivo Input1

Assim como na lista Eventos, a lista Gates possui uma lista para cada portão lógico da árvore de falha onde estas listas contem as seguintes informações: *Index* para identificação do portão lógico, operador lógico relacionado ao evento, evento de saída do portão e uma lista com os eventos de entrada do portão. Diferente da lista Eventos, os parâmetros da lista Gates não se alteram durante as interações do programa.

Abaixo são demonstradas exemplos de listas de identificação para um evento e um portão lógico.

- Evento → [*Index*, Descrição, Valor Lógico, Valor Booleano]
- Portão Lógico → [*Index*, Operador Lógico, Evento de Saída, [Eventos de Entrada]]

O próximo passo do programa é a resolução lógica da árvore de falha. Nesta etapa o programa executa varreduras na lista de eventos, analisando inicialmente os eventos de profundidade mais baixa. Começando pelo evento de profundidade 1, o programa inicialmente verifica se o evento analisado não possui seu valor lógico definido. Se o valor já estiver definido o evento é ignorado e caso o valor seja indefinido, o programa verifica o valor lógico de seus eventos de entrada e, caso possível, define seu valor lógico. Este procedimento é realizado até chegar ao ultimo evento do nível mais profundo e caso ainda exista algum evento cujo valor lógico seja uma string vazia o programa repete todo o procedimento voltando ao elemento de profundidade 1.

Conforme dito no capítulo 3, foi necessária a adição do valor lógico "*none*" para representar o caso de não ser possível definir se o evento é verdadeiro ou falso. Junto ao valor lógico "*none*" foi criado um portão lógico chamado *Inhibit-none*. Devido às adições, é necessário apresentar quais são as influências sobre a tabela verdade das operações lógicas usadas e para isto, foi criada a Tabela 2 que apresenta as tabelas verdade das operações usadas.

Tabela 2: Tabela Verdade dos portões *AND*, *OR*, *NAND*, *NOR*, *NOT* e *INHIBIT_NONE* considerando os valores lógicos "verdadeiro" (1), "falso" (-1) e "*none*" (0)

AND			OR			INHIBIT_NONE	
A	B	X	A	B	X	A	X
1	1	1	1	1	1	1	-1
1	0	0	1	0	1	0	1
1	-1	-1	1	-1	1	-1	-1
0	1	0	0	1	1		
0	0	0	0	0	0		
0	-1	-1	0	-1	0		
-1	1	-1	-1	1	1		
-1	0	-1	-1	0	0		
-1	-1	-1	-1	-1	-1		

NAND			NOR			NOT	
A	B	X	A	B	X	A	X
1	1	-1	1	1	-1	1	-1
1	0	0	1	0	-1	0	0
1	-1	1	1	-1	-1	-1	1
0	1	0	0	1	-1		
0	0	0	0	0	0		
0	-1	1	0	-1	0		
-1	1	1	-1	1	-1		
-1	0	1	-1	0	0		
-1	-1	1	-1	-1	1		

Uma vez resolvida a árvore de falha, a próxima etapa do programa é o tratamento dos dados de saída. Para isso um arquivo texto, criado pelo especialista do sistema, fornecerá as informações necessárias para definir os alertas que foram ativados no ciclo. Neste arquivo texto são listados os alertas conforme demonstrado na Figura 18. O formato escolhido para apresentar os alertas no arquivo é dado da seguinte forma: [*Index* do evento relacionado, valor lógico que ativará este alerta, *index* de classificação do alerta, mensagem de alerta]. O *index* de classificação do alerta será utilizado pelo PIG para definir o formato da mensagem de alerta na interface gráfica.

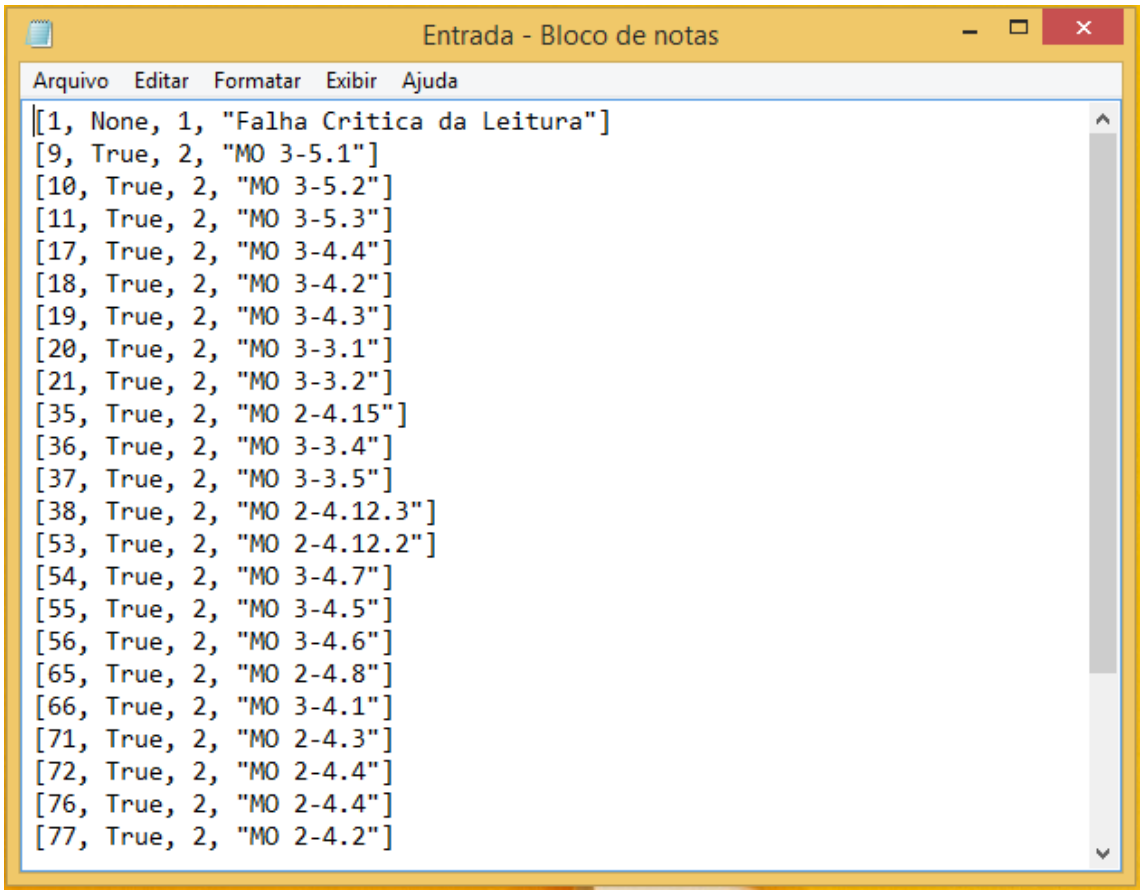


Figura 168: Exemplo de lista de alertas

Os dados finais deste programa é uma lista de evento com os valores lógicos atualizados após resolução da árvore de falha e uma lista com as mensagens de alerta que foram ativadas. Estas duas listas são fornecidas para o PIG para que possa ser transmitida para o usuário.

4.2. Programa de Aquisição e Manipulação dos Dados de Entrada (PAMDE)

Este programa pode ser dividido em duas etapas, onde a primeira possui como objetivo obter os dados de entrada do sistema analisado no momento correto e a segunda trata os dados obtidos de forma que possam ser usados na árvore de falha.

A primeira ação deste programa é obter a estrutura da árvore de falha que será utilizada. Como a árvore de falha não se altera durante a operação do programa, esta etapa ocorre apenas ao iniciar o programa.

Junto à obtenção da estrutura da árvore de falha ocorre a primeira leitura de dados da usina. Este processo irá ocorrer a cada ciclo de 10 segundos para que a análise em tempo real da usina seja possível.

Neste trabalho é considerado que os programas de monitoração da usina forneçam um arquivo texto listando os valores das variáveis relevantes à execução do programa. Um exemplo para este arquivo é demonstrado na Figura 19 onde as variáveis são definidas na seguinte forma: [*index* de identificação da variável, descrição da variável, valor da variável].

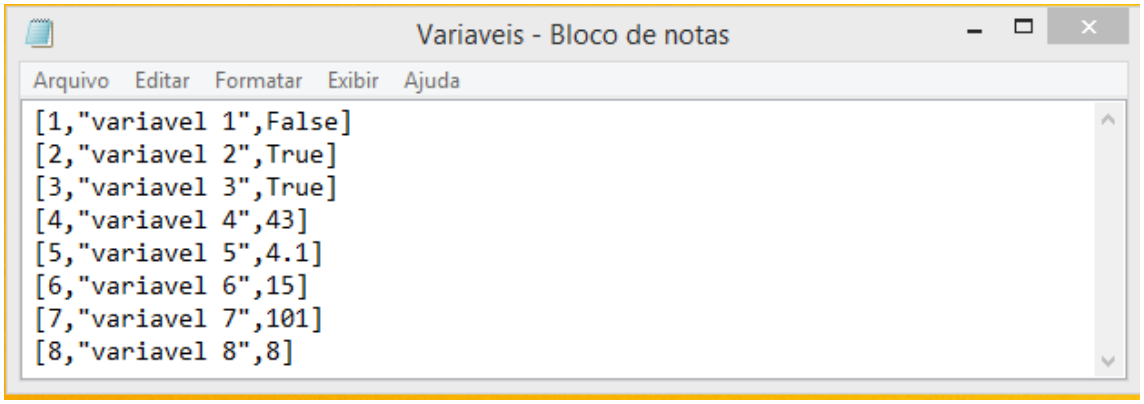


Figura 19: Exemplo de arquivo texto listando as variáveis

Outras informações que devem ser fornecidas para o programa são: 1) Listas contendo as regras que usam as variáveis fornecidas pelo arquivo citado na Figura 19 para definir o estado dos eventos de entrada da árvore de falha. 2) Listas contendo os parâmetros para as funções de pertinência que serão utilizadas no programa. Para

armazenar estas listas foi criado um arquivo texto, chamado Regras, onde um exemplo é citado na Figura 20.

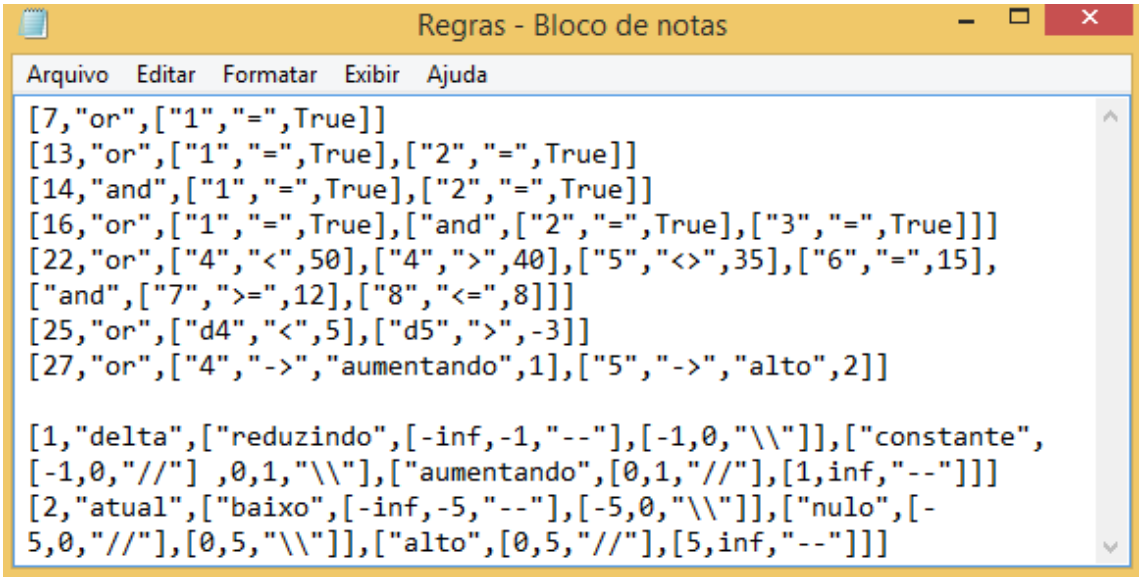


Figura 170: Exemplo do arquivo contendo as regras dos dados de entrada da árvore de falha e os parâmetros para as funções de pertinência que serão usadas no programa.

Na primeira parte do arquivo Regras, são fornecidas as regras para definir o estado dos eventos de entrada. Para armazenar as regras foi criada uma lista onde seu primeiro elemento é o *index* do evento relacionado à regra, o segundo é o operador lógico que será utilizado na regra e, por fim, são listadas as premissas da regra.

Uma premissa de regra pode ser representada por uma ou múltiplas operação relacionando as variáveis do sistema. No caso de múltiplas variáveis será necessário o uso de um operador lógico. Um exemplo de ambos os casos são demonstrados respectivamente abaixo.

- ["1","=",*True*] → Se a variável com *index* 1 é igual à "*True*" então esta premissa é verdadeira

- ["and",["2","=",*True*],["3","=",*True*]] → Se a variável com *index* 2 é igual à "*True*" e a variável com *index* 3 é igual à "*True*" então esta premissa é verdadeira

Para representar as possíveis operações dentro de uma premissa de regra foi utilizado o seguinte formato: [identificador da variável, símbolo da operação, parâmetro, *index* da função de pertinência].

No item "identificador da variável" desta lista é fornecido o *index* da variável a ser usada na operação. No caso da regra estar vinculada com a taxa de variação de uma variável fornecida é necessário adicionar a letra "d" à frente do *index* da variável. Exemplo: "d1" = "taxa de variação da variável com *index* 1".

O item "símbolo da operação" representa qual operação será feita entre o valor da variável e o valor do parâmetro. As possíveis operações são: Igualdade (=), desigualdade (<>), maior (>), menor (<), maior-igual (>=), menor-igual (<=) e por fim um símbolo utilizado para operações que utilizam de lógica nebulosa (->) que indica a função pertinência cuja variável deve possuir maior valor.

O item "parâmetro" indica qual o valor que será utilizado na operação junto ao valor da variável citada na premissa. O item "*index* da função de pertinência" determina qual será a função de pertinência a ser usada nesta premissa e só é utilizado quando o item "símbolo de operação" é "->".

A segunda parte do arquivo Regras representa as funções de pertinência a serem usada no programa. Para representar essas funções foram usadas listas com a seguinte estrutura: [*index*, identificador de ordem, definição dos limites].

A sessão "identificador de ordem" determina se o valor usado na função de pertinência será a própria variável citada na regra ou sua taxa de variação. Em caso que seja a própria variável o valor em "identificador de ordem" será "atual" e caso seja a taxa de variação o valor será "delta". Este indicador é necessário, pois em casos que se usam como parâmetro de uma operação palavras como "aumentando" e "diminuindo" o valor a ser usado para determinar o grau de pertinência será a taxa de variação da variável vinculada a esta operação. Exemplo: "Se Variavel 1 está aumentando → Valor da operação = Verdadeiro". Neste exemplo não será usado o valor da variável 1 e sim a sua taxa de variação.

Existe uma sessão "definição dos limites" para cada curva no gráfico de função de pertinência. A estrutura desta lacuna será em formato de lista onde primeiramente vira o valor difuso que a função de pertinência representa seguida dos limites. Para determinar os limites foram usadas listas contendo o seguinte formato: [valor mínimo do domínio da função, valor máximo do domínio da função, representação da inclinação da reta]. Os possíveis símbolos que representam a inclinação da reta são: "//" quando a reta possuir derivada positiva, "\\" quando a reta possuir derivada negativa e "--" para o caso de derivada nula.

Um exemplo de funções de pertinência para uma regra é dado a seguir e a Figura 21 é a representação gráfica deste exemplo. Na Figura 21, o eixo das abscissas

representa a taxa de variação de uma variável e a ordenada representa o grau de pertinência.

- [1,"delta",["reduzindo",[-inf,-1,"--"],[-1,0,"\\"]],["constante",[-1,0,"//"],[0,1,"\\"]],["aumentando",[0,1,"//"],[1,inf,"--"]]]

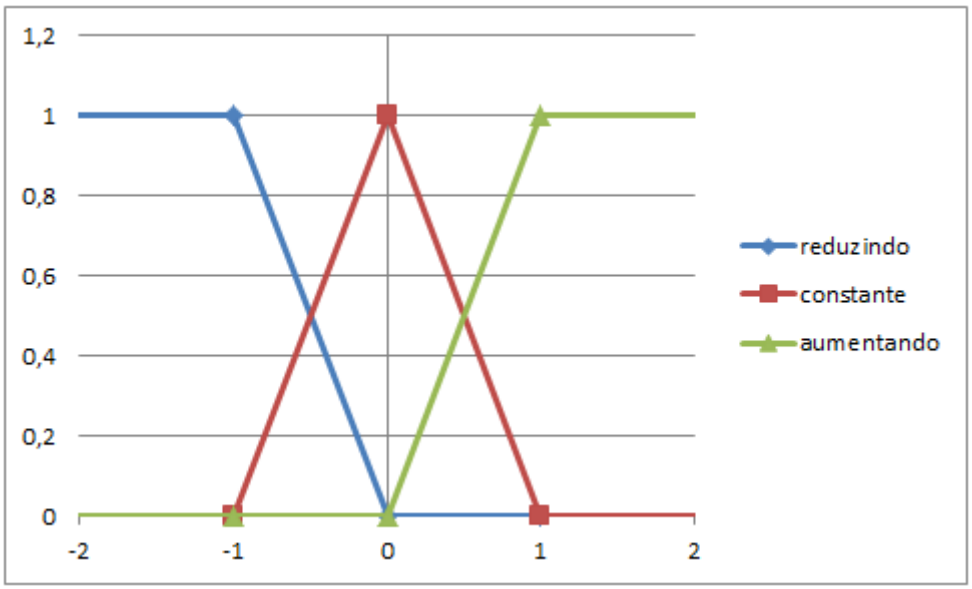


Figura 181: Representação gráfica do exemplo

Após coletar todos os dados de entrada que serão utilizados neste programa, os agentes inteligentes executam o tratamento das variáveis.

O método usado pelo agente inteligente responsável por premissas que possuam taxa de variação foi pegar o valor da variável no ciclo atual, subtrair o valor da mesma variável no ciclo anterior e dividir pela duração de um ciclo.

Para os casos que demandam o uso do agente inteligente vinculado a variáveis difusas, o método usado foi a realizar o processo de *fuzzyficação* da variável. Para isso,

usam-se as funções de pertinência vinculada à regra para definir o grau de certeza para cada categoria e em seguida é determinar qual possui o maior valor. Uma vez definido qual categoria possui o maior grau de certeza, compara-se o resultado com o resultado esperado e é definido o valor lógico da premissa.

4.3. Programa de Interface Gráfica (PIG)

O primeiro passo na criação deste programa foi fazer um estudo de como seria organizada a interface gráfica para que toda a informação seja transmitida ao usuário e que seja possível a interação do usuário com o programa. Neste estudo foram traçados os objetivos da interface gráfica, onde estes são:

- Apresentar uma lista em ordem cronológica com os alertas emitidos pelo sistema;
- Apresentar explicação detalhadas de cada alerta ocorrido;
- Apresentar ao usuário qual modulo do MO deverá ser consultado no alerta selecionado;
- Permitir que o usuário determine o estado de um ou mais eventos do sistema.

Feito o estudo, foi definido a estrutura do programa que possui uma janela principal, demonstrada na Figura 15 e uma janela secundária, demonstrada na Figura 16.

A criação de ambas as janelas usadas neste programa foi dividida em três etapas. A primeira etapa consiste nas ações que serão executadas no momento que a janela for aberta. Alguns exemplos destas ações são: Definição da estrutura da janela e tomada de dados. O processo de definição da estrutura da janela principal e na janela de input foi citado na sessão 3.3 deste texto. Nos programas de ambas as janelas os dados de entrada são obtidos das informações criadas no PAMDE e no PGSAF, porém, o programa da janela principal utiliza também de dados fornecidos por arquivos texto.

Os arquivos texto utilizados são: O arquivo Entrada, que foi mencionado na sessão 4.1 deste texto e o segundo arquivo é o arquivo Layout que contem a estrutura de como serão as mensagens apresentadas em "Detalhamento do Evento" ao selecionar um dos alertas citados em "Cronograma de Eventos". A Figura 22 apresenta um exemplo de arquivo Layout onde todos os modelos são definidos e sua estrutura contém uma descrição do *layout*, o *index* de classificação do alerta ao qual o layout está vinculado e por fim, o texto que será usado em "Detalhamento do Evento".

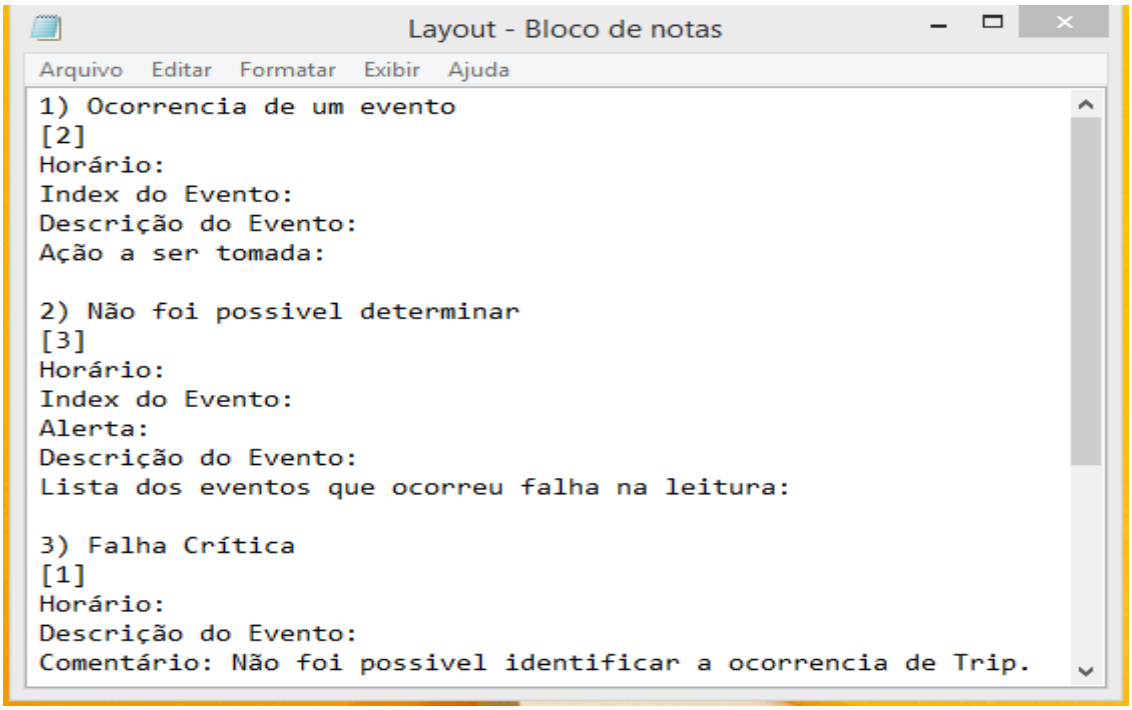


Figura 192: Estrutura do arquivo Layout.

A segunda etapa é definir quais serão as reações ao executar ações como clicar em um botão, selecionar um elemento de uma *listbox* ou mover uma barra de rolagem.

As possíveis interações que o usuário pode ter com a janela principal são: Iniciar o programa, selecionar um alerta em "Cronograma de Eventos" para que seu detalhamento apareça em "Detalhamento do Evento" ou abrir a janela "*Input*".

O processo de inicialização do programa consiste em alterar o sinalizador de estado do programa de "*Offline*" para "*Online*" e ativar a rotina para que a primeira operação cíclica seja serializada.

Ao selecionar um dos alertas em "Cronograma de Eventos" o programa identifica o alerta selecionado e com o auxilio do arquivo "Entrada", determina qual foi

o evento causador e o *index* de classificação do alerta. Com o *index* de classificação do alerta e o arquivo "Layout" é identificado qual será o formato do texto apresentado em "Detalhamento do Evento". No modelo de árvore de falha utilizado neste trabalho existem 3 tipos de layout diferentes, onde estes foram demonstrados na Figura 22. Uma vez definido o formato do texto é necessário preencher as informações sobre o alerta. Algumas das informações a serem preenchidas são: Horário de ocorrência do alerta, *index* do evento causador, ação que o operador deve tomar ao identificar o alerta e uma lista dos eventos que ocorreram falha na leitura.

Após iniciar o programa, torna-se possível usar o botão "*Input*". Ao utiliza-lo será verificada a existência de uma janela de input aberta e caso não tenha, a rotina de iniciação da janela de input será inicializada utilizando os dados do momento em que a janela foi aberta.

Na janela "*Input*", o usuário tem a capacidade de definir o estado dos eventos de entrada do programa utilizando os *checkbuttons*, selecionar um evento para que seja apresentada a informação detalhada do evento na sessão "Descrição do Evento" e alterar o nível de liberdade do usuário em alterar os valores dos eventos de entrada.

Para cada evento de entrada do programa existem 3 *checkbuttons* que estão vinculados com os 3 possíveis valores lógicos do evento. O primeiro passo da criação destes *checkbuttons* foi garantir que apenas um dos 3 *checkbuttons* esteja ligado e que não seja possível desligar todos os *checkbuttons* ao mesmo tempo.

Ao clicar em um *checkbutton* o programa armazena o *index* da variável alterada junto de seu novo valor e utiliza essas informações no próximo ciclo de tomada de dados. Conforme dito no capítulo 4.1, a lista Eventos armazena uma variável chamada de “valor booliano”. Esta variável pode possuir dois valores, sendo estes 0 ou 1 onde quando seu valor é 0 as alterações realizadas pelo usuário usando os *checkbuttons* serão aplicadas apenas no ciclo seguinte e quando seu valor for 1 as alterações serão realizadas em todos os ciclos seguintes até que outras alterações sejam realizadas.

Como descrito no capítulo 3.3, junto dos *checkbuttons* também existe uma lacuna identificando o *index* do evento relacionado aos *checkbuttons*. Ao clicar neste *index* do evento uma descrição do evento será apresentada na sessão “Descrição do Evento” onde nesta descrição estarão listados o *index* do evento selecionado, nome do evento selecionado e o valor lógico em que o evento se encontra no ciclo vigente.

Ao selecionar “total” como o nível de liberdade todos os eventos de entrada terão seus 3 *checkbuttons* liberados para alteração, permitindo assim que o usuário mudo qualquer valor de entrada dos eventos. Caso o nível “parcial” seja selecionado apenas os eventos que possuírem valor “*none*” no ciclo atual ou que possuam a variável “valor booliano” igual a 1 serão liberados para alterações pelo usuário e os demais estarão bloqueados. Ao selecionar o nível “parcial” o processo de liberar e bloquear os *checkbuttons* é realizado a cada ciclo de tomada de dados.

Em ambas as janelas foi necessário criar uma terceira etapa que executa operações cíclicas no programa. Na janela principal do programa o exemplo de operação cíclica é o fato que a cada período de 10 segundos novos dados sobre o

sistema serão adicionados na sessão “Cronograma de Eventos”. As ações realizadas neste ciclo de 10 segundos são: Executar o PGSAF para resolver a árvore de falha do ciclo vigente, formatar as mensagens de alerta e adiciona-las à sessão “Cronograma de Eventos” e alterar o estado dos sinalizadores.

Nos ciclos seguintes ao primeiro, o PGSAF é executado considerando a possibilidade de algum evento ter sido alterado pelo usuário em um ciclo anterior. Para verificar se algum evento foi alterado o programa verifica se o evento esta presente na lista criada ao fazer uma alteração pela janela de input

São 2 os sinalizadores atualizados a cada ciclo, o primeiro sinalizador identifica se ouve ou não falha na leitura dos dados de entrada do programa. Para identificar a falha na leitura é feito uma busca por qualquer evento que possua valor lógico igual a “*none*”. A Figura 23 demonstra os dois possíveis estados deste sinalizador.

Figura 23: Possíveis estados do sinalizador de leitura.

O segundo sinalizador notifica se a usina se encontra em operação normal, se ouve ocorrência de *Trip* ou se não é possível definir o estado da usina. A Figura 24 demonstra os três estados deste sinalizador.

Figura 204: Possíveis estados do sinalizador de ocorrência de *Trip*

As operações cíclicas na janela de input ocorrem a cada período de 0,5 segundos e possuem como objetivo principal, atualizar o estado dos *checkbuttons* de cada evento. Caso não houvesse esta operação cíclica o operador seria obrigado a reabrir a tela de input para que ela pudesse atualizar e obter os dados de um novo ciclo.

Para decidir qual seria o período de tempo de cada ciclo da janela de input foi feito uma análise onde como resultado foi notado que o período não poderia ser maior que o período de tomada de dados de 10 segundos, pois não seria possível atualizar os *checkbuttons* com o estado de cada ciclo. Uma vez definido que o ciclo será menor que 10 segundos, foi observado que quanto menor fosse o período menos tempo seria perdido mostrando os dados do ciclo passado em um novo ciclo. Desta forma foi decidido o valor de 0,5 segundos porque reduz o valor perdido mostrando dados do ciclo passado e não é um valor muito pequeno ao ponto do tempo de processamento do programa interferir nos ciclos e não demandar muito do computador.

O programa da janela de input se comunica com o programa da janela principal por meio de uma variável chamada “contador” onde esta variável pode apresentar os valores de 0 ou 1 que indicam respectivamente se o ciclo vigente no programa da janela de input é o primeiro ciclo realizado após uma tomada de dados ou um ciclo seguinte.

Quando a variável “contador” possui valor igual a 0 é alertado no sinalizador que um novo ciclo começou e são obtidos os valores do resultado da árvore de falha. Quando seu valor é 0 e o nível de liberdade do usuário esta definido como “parcial” será

feito a atualização de quais *checkbuttons* serão bloqueados e quais serão habilitados para alterações pelo usuário.

CAPÍTULO 5

5. Simulação e Resultados

Neste capítulo será comentado como o programa foi testado e quais foram os resultados obtidos destes testes.

5.1. Simulação

Para verificar a funcionalidade do programa, foram feitos testes utilizando 3 variáveis obtidas do simulador da Central Nuclear Almirante Álvaro Alberto (Angra 2) para o acidente de perda de refrigerante (LOCA) juntamente de 7 variáveis criadas para que fosse possível testar todos os casos de LOCA. Também foram criadas as funções de pertinência das variáveis que serão tratadas pelo agente inteligente de lógica nebulosa.

Devido à característica do Sistema Especialista de poder alterar a Base de Regras sem que seja necessário alterar o modelo de inferência do programa, os resultados de um teste com um fragmento da árvore de falha representam o que ocorreria com a árvore de falha completa.

Os objetivos destes testes foram:

- Verificar se a tomada de dados foi realizada corretamente em cada inicio de ciclo;
- Verificar a ação dos agentes inteligentes ao obter os dados de entrada e trata-los de forma a obter os dados de entrada para a árvore de falha;

- Verificar se a criação e solução da árvore de falha foram executadas corretamente;
- Verificar se a interface gráfica do programa funcionou corretamente permitindo a interação do usuário com o programa e informando a ocorrência de eventos ao usuário;
- Observar a redução no tempo necessário para identificar o transiente/acidente.

A Figura 25 mostra as variáveis que foram usadas na simulação onde cada uma foi avaliada em um período de 60 segundos. Como o programa realiza leituras a cada 10 segundos, foi possível executar 7 análises do estado da usina para cada simulação.

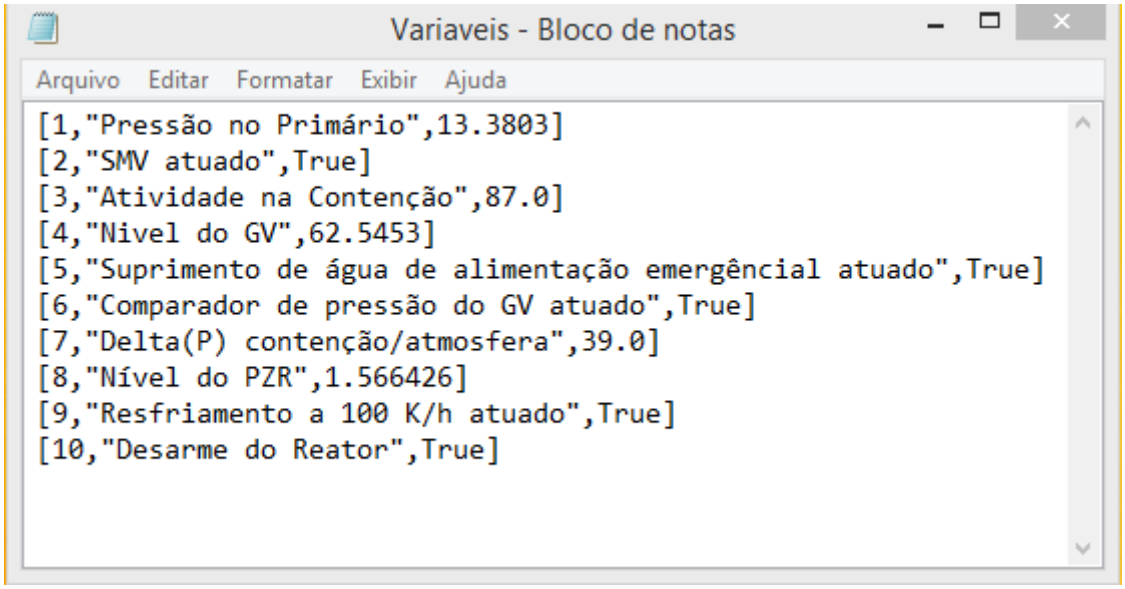


Figura 215: Exemplo das Variáveis Usadas na Simulação

Foram testados 5 cenários diferentes onde foram simulados o caso de operação normal da usina, falha critica na leitura dos dados, ocorrência de um acidente/transiente fora do projeto do trabalho, ocorrência de cada um dos 5 diferentes tipos de LOCA

presentes na ALD e impossibilidade de definir a ocorrência de um evento devido a falha na leitura. Na Figura 26 é dado um exemplo com os valores normalizados das 10 variáveis durante os 60 segundos da simulação. Para as variáveis boolianas, foi considerado o valor 0 para falso e 1 para verdadeiro.

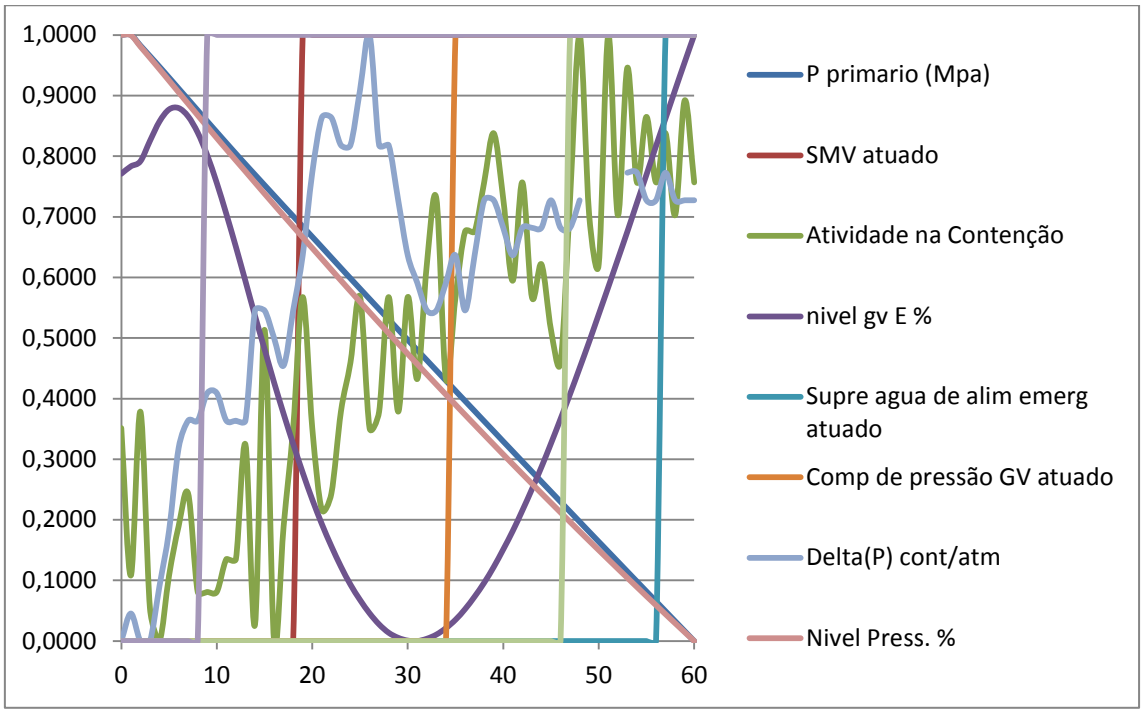


Figura 226: Gráfico Normalizado dos dados utilizados em uma das simulações.

5.2. Resultados

Como mencionado no capítulo 5.1., foram feitas simulações para 5 cenários diferentes. Nesta sessão serão apresentados cada um dos cenários testados e os resultados obtidos.

Com os dados utilizados na primeira simulação foi esperado que a usina mantivesse o estado de operação normal devido ao fato de não ter ocorrido o *Trip* da

usina. A Figura 27 exemplifica como ficou a tela principal do PIG ao termino da simulação.

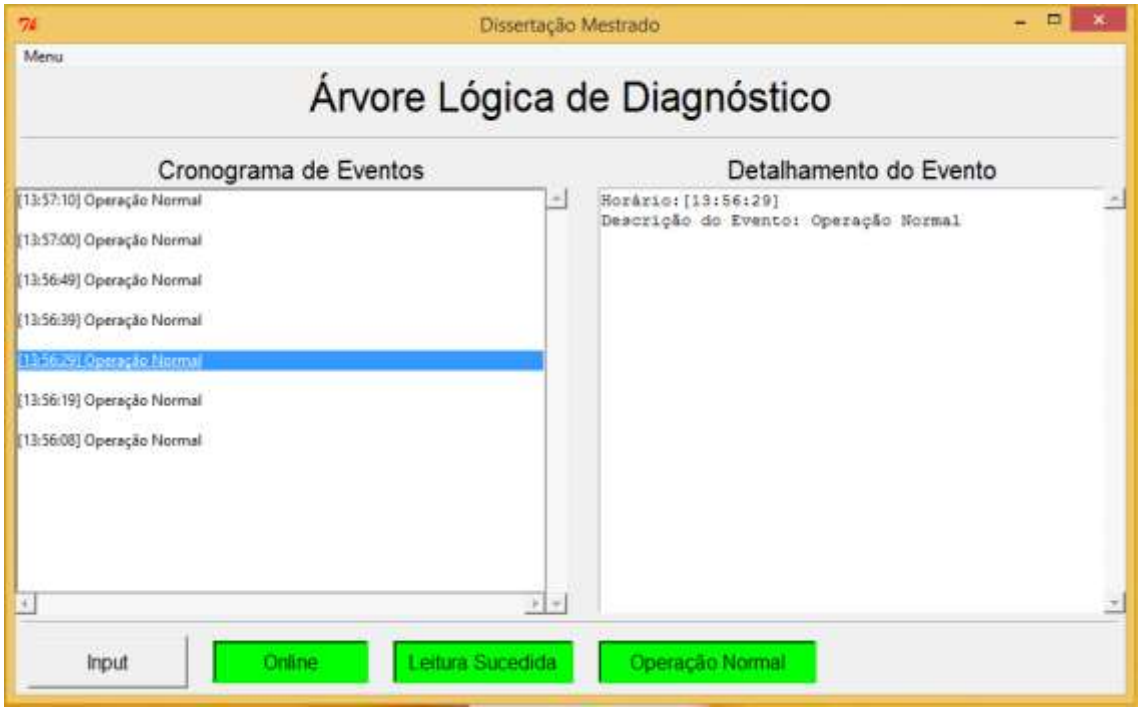


Figura 237: Tela principal do PIG após 60 segundos de simulação. Simulação de operação normal.

O segundo teste simula a ocorrência de uma falha crítica na leitura de dados. Isso ocorre quando não é possível definir se houve ou não, um *Trip* na usina. Nessa situação é impossível dizer se a usina opera normalmente ou se esta ocorrendo um acidente/transiente postulado ou não postulado. A Figura 28 demonstra a tela principal do PIG no momento em que ocorre o alerta de falha crítica na leitura.

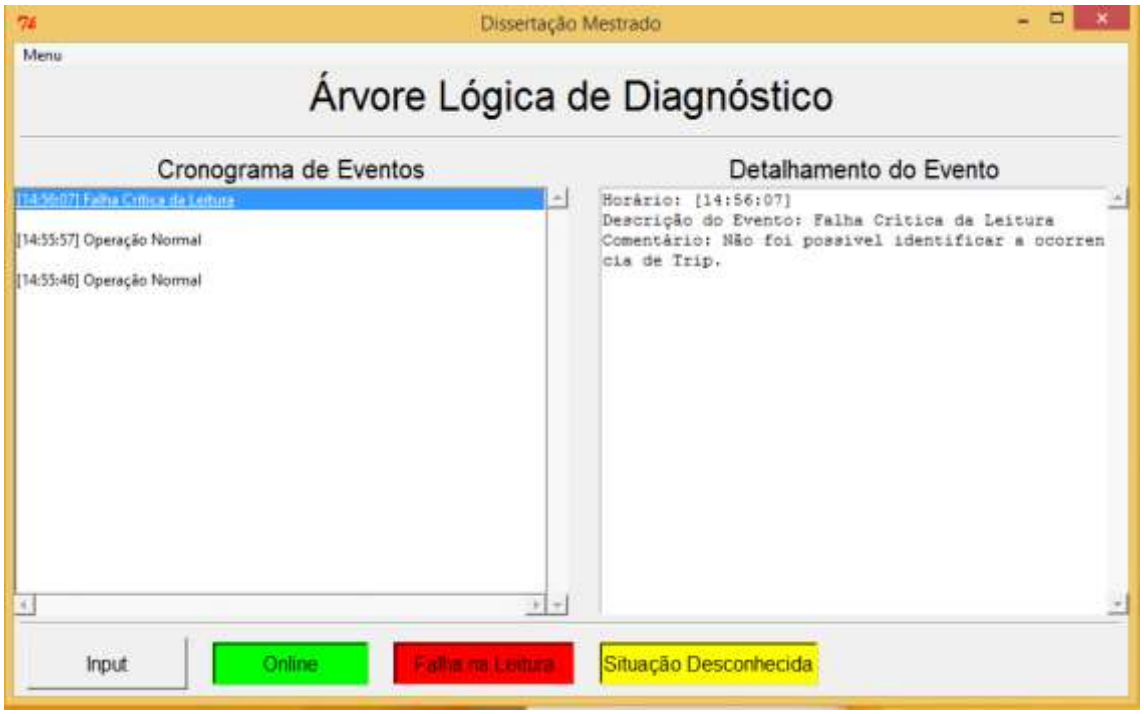


Figura 28: Tela principal do PIG no momento em que ocorre a falha crítica na segunda simulação.

No terceiro cenário simulado, a usina inicia em operação normal, evolui para uma situação de “Pequena quebra do secundário dentro da contenção” que mais adiante se torna o evento “Quebra linha aalim secundário na contenção mont última vv retenção”. Pode-se se observar que na sessão “Detalhamento do Evento” é sinalizado qual capítulo do MO o usuário deve consultar para o evento selecionado. A Figura 29 mostra a tela principal do PIG após os 60 segundos simulando o cenário descrito anteriormente.

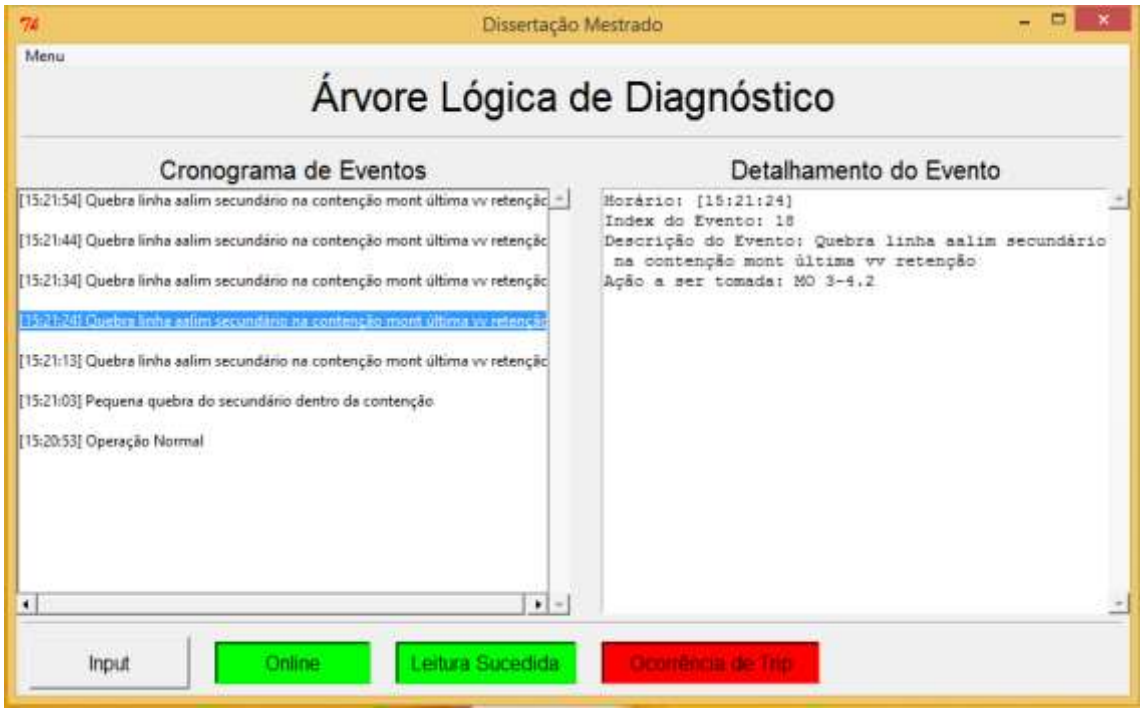


Figura 29: Tela principal do PIG após 60 segundos de simulação. Simulação com evolução de operação normal para um acidente postulado.

No quarto cenário a usina se encontrava inicialmente em operação normal e após 10 segundos ocorreu o *Trip* do reator, porém não foi possível relacionar o padrão dos dados obtidos com nenhum transiente/acidente postulado no projeto. Após 40 segundos de simulação foi identificada a ocorrência de um transiente/acidente que continuou até o fim da simulação.

O ultimo cenário de teste possui como objetivo testar o caso de impossibilidade em determinar a ocorrência de um acidente/transiente devido a falha na leitura de algumas variáveis. Neste caso uma variável não possui valor e com isso o programa não consegue definir qual dos dois eventos que ocorreu. A Figura 30 demonstra a tela principal do PIG para este cenário onde se pode ver que o programa apesar de saber que o evento 20 ou 21 ocorreu, ele não tem certeza de qual foi, sendo esta a forma do

programa dizer “não sei”. Na figura também se pode ver que o programa indica qual foi o evento que não possui valor.

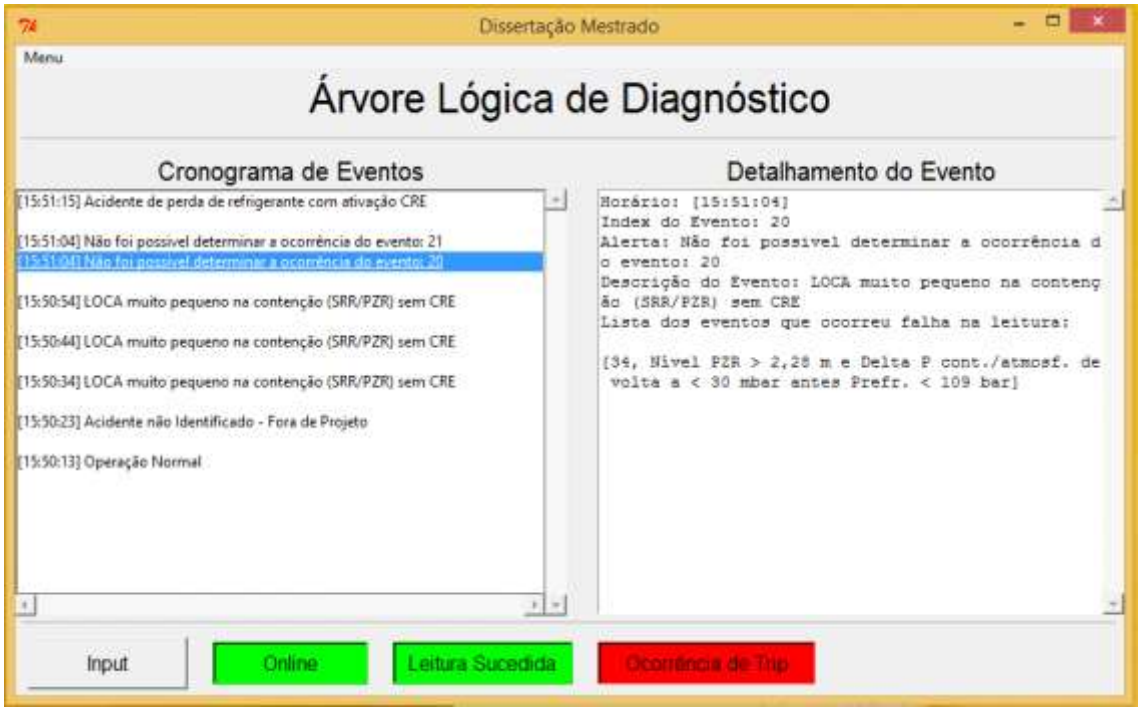


Figura 240: Tela principal do PIG após 60 segundos de simulação. Simulação com falha de leitura de dados.

Juntamente com os 5 cenários testados, também foram testados casos onde o usuário necessita interagir com a interface gráfica para alterar uma variável. Nestes testes foi verificado o correto funcionamento da sessão de *input* do programa.

CAPÍTULO 6

6. Conclusão

Como citado anteriormente, o objetivo deste trabalho foi criar um programa que com o uso de técnicas de IA possibilite a simplificação do trabalho dos operadores de centrais nucleares ao automatizar o sistema de identificação de acidente e informe ao operador quais sessões do MO deverão ser consultadas para normalizar o estado da usina. Ao automatizar este sistema, o estresse sobre o operador é reduzido e com isso o risco de erros devido a fatores humanos é reduzido.

Com os resultados citados no capítulo 5 observa-se que o programa cumpriu com o objetivo de realizar o diagnóstico da usina onde a cada ciclo de 10 segundos foi realizado um diagnóstico. Enquanto o programa leva 10 segundos para realizar o diagnóstico o método manual leva questão de minutos ou horas o que demonstra a eficiência do programa em reduzir o tempo de execução.

Uma das características relevantes deste programa esta na inclusão do valor “none” na resolução da árvore de falha, que possibilitou que o programa sinalize quando não é possível definir o estado da usina.

Outro fator diferencial é a capacidade do programa em obter dados do sistema da usina, onde no caso do trabalho foi o SICA, ou obter dados de forma manual, onde o usuário define os valores das variáveis para o ciclo seguinte.

Com o uso do sistema especialista o programa possui a característica de não ser limitado apenas ao sistema onde foi testado inicialmente, podendo se usado em diversos modelos de ALD sem que sejam necessárias grandes alterações nos códigos do programa.

Para a área acadêmica, este trabalho proporciona a ampliação do conhecimento nas áreas de IA, análise de diagnóstico e segurança em centrais nucleares. Outro fator é a ampliação nas pesquisas relacionada à aplicabilidade da IA, onde, neste trabalho, foi utilizado SE e Lógica Nebulosa no diagnóstico em tempo real de uma usina nuclear.

A relevância deste trabalho na área industrial é o fato do programa proposto ter o intuito de aumentar a segurança no ambiente de trabalho com foco em usinas nucleares. Outro objetivo deste trabalho é colaborar com o aumento do uso de técnicas que reduzam o estresse sob os operadores da sala de controle de uma usina nuclear e, com isso, proporcionar um ambiente de trabalho mais seguro.

Como propostas para trabalhos futuros no tema pode-se ser feita a adição da sessão da ALD sem IDR. Outra sugestão para trabalhos futuro é a simulação utilizando mais dados da usina para que possa ser visualizado os demais acidentes/transientes listados na ALD.

REFERÊNCIAS

[1] Agência Nacional de Energia Atômica. Disponível em: https://www.iaea.org/pris/. Acesso em: Fevereiro de 2016.

[2] Eletrobras Eletronuclear. Disponível em: http://www.eletronuclear.gov.br/Saibamais/Perguntasfrequentes/EnergiaNuclearnoBrasil.aspx. Acesso em: Fevereiro de 2016.

[3] Manual de Operação da Usina – Identificação de Acidente – Árvore Lógica de Diagnósticos, Eletrobras Eletronuclear, 2010

[4] RUSSELL, S. e NORVIG, P., 1995, Artificial Intelligence A Modern Approach. New Jersey, Prentice Hall.

[5] BARTLETT, E. B., "Detecting Faults in a Nuclear Power Plant by Using a Dinamic Node Architecture Artificial Neural Network", **Nuclear Science and Engineering**, v. 116, pp. 313-325, 1994.

[6] BARTAL, Y., LIN, J., UHRIG, R. E., "Nuclear Power Plant Transient Diagnostics Using Artificial Neural Network that Allow "Don't Know" Classifications", **Nuclear Tecnology**, Vol. 110, pp. 346-449, June 1995.

[7] ALVARENGA, M.A.B., MARTINEZ, A.S., SCHIRRU, R., "Adaptive Vector Quantization Optimized By Genetic Algorithm For Real-Time Diagnosis Through Fuzzy Sets". **Nuclear Tecnology**, v. 120, pp. 188-197, Dec. 1997.

[8] COSTA, R. G., Sistema de Auxílio para o Direcionamento da Atenção no Diagnóstico de Acidentes em Usinas Nucleares Baseado em Inteligência Artificial. Dissertação de M.Sc., CNEN/IEN, Rio de Janeiro, RJ, Brasil, 2009

[9] SALMON, D. R., Sistema Especialista Baseado em Níveis Progressivos de Diagnóstico para Identificação de Acidentes em Usina Nuclear PWR. Dissertação de M.Sc., UFRJ/COPPE, Rio de Janeiro, RJ, Brasil, 2013

[10] ALLALOU, A. N., TADJINE, M., BOUCHERIT, M. S., "Online monitoring and accident diagnosis aid system for the Nur Nuclear Research Reactor". Mar. 2016. Disponível em: http://journals.tubitak.gov.tr/elektrik/issues/elk-16-24-3/elk-24-3-65-1401-272.pdf. Acessado em: Janeiro de 2017.

[11] ROBERT E. UHRIG, LEFTERI H. TSOUKALAS, "Multi-agent-based anticipatory control for enhancing the safety and performance of Generation-IV nuclear power plants during long-term semi-autonomous operation". Disponível em: http://ac.els-cdn.com/S0149197003000039/1-s2.0-S0149197003000039-main.pdf?_tid=c97ba942-05a4-11e7-8e64-00000aab0f26&acdnat=1489159277_e04cac17026e3a2e3946074345e1a61a. Acessado em: Janeiro de 2017

[12] PASSOS, Emmanuel Lopes, 1989, Inteligência Artificial e Sistemas Especialistas Ao Alcance de Todos. Rio de Janeiro, Livros Técnicos e Científicos Editora LTDA.

[13] KELLER, Robert. Tecnologia de Sistemas Especialistas: Desenvolvimento e Aplicação. Tradução: Reinaldo Castello. São Paulo: Makron Books, 1991.

[14] NUREG-0492, Fault Tree Handbook, US Nuclear Regulatory Commission, Washington, DC, 1981.

[15] SOUTO, K., Base para uma Arquitetura Cognitiva Destinada à Supervisão de Segurança na Operação de Usinas Nucleares. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2001.

[16] Stanford Encyclopedia of Philosophy, Jan Lukasiewick. Disponível em: https://plato.stanford.edu/entries/lukasiewicz/. Acesso em: Janeiro de 2017.

[17] RIGNEL, D. G. S., CHENCI, G. P., LUCAS, C. A., "Uma Introdução a Lógica Fuzzy". **Revista Eletônica de Sistemas de Informação e Gestão Tecnológica**, v. 01, 2011.

[18] Alessandro Assi Marro et al. Lógica Fuzzy: Conceitos e aplicações. DIMAp/UFRN. Disponível em: http://aquilesburlamaqui.wdfiles.com/local--files/logica-aplicada-a-computacao/texto_fuzzy.pdf. Acesso em: fevereiro de 2016.

[19] PASSINO, K. M, YURKOVICH, S., 1998, Fuzzy Control, Addison-wesley.

Apêndice I

Lista com a Nomenclatura dos Eventos da Árvore de Falha

Index	Nome do Evento	*Index*	Nome do Evento
1	Identificação de Acidentes	47	
2	Acidentes Detectáveis	48	
3	Acidente não Identificado com ocorrência de *TRIP*	49	identif. acidente/perturbação não possível
4	Impactos Externos	50	Acidente não identificado devido à falta de dados
5	LOCA primário e quebra secundária dentro contenção	51	
6	Ruptura tubos GV c/s limites radioact. VP excedidos	52	Quebra linha VP fora da contenção
7	Desarme do reator iniciado	53	Normalização da usina após iniciação do DAF1
8		54	Transiente de sobreresfriamento causado por falha no secundário
9	Impacto externo durante operação a potência	55	Quebra de linha de VP a jusante da válvula de isolamento do VP
10	Impacto externo com inventário do SRR reduzido ou durante recarregamento	56	Válvula de segurança ou de controle de alívio de VP bloqueada aberta após atuação
11	Terremoto	57	Grad. pressão VP > max. excedida (*JR 65* atuado)
12		58	
13	Sala de controle principal não disponível ou terremoto	59	Pressão VP caiu antes do grad pressão VP > max. atingido (DAF 1)
14	Ponte nível PZR ativada	60	
15		61	
16	Sinais de Terremoto	62	Sinal var. pressão VP (DAF 1) foi 1º critério disparo do reator
17	Pequena quebra do secundário dentro da contenção	63	1 de 4 VBL VSE VP fechada ou 1 de 4 VBL VCP VP e VBL VSE VP assoc. fechada
18	Quebra linha aalim secundário na contenção mont última vv retenção	64	Eventos de perda de água de alimentação
19	Quebra linha aalim secundário na contenção jus última vv retenção	65	Perda de suprimento de água de alimentação
20	LOCA muito pequeno na contenção (SRR/PZR) sem CRE	66	Quebra linha de água de alimentação do secundário fora da contenção
21	Acidente de perda de refrigerante com ativação CRE	67	Falha alim. GV ou níveis GV caindo
22	Mudança na atmosf. cont. Delta P aumentando ou atua SMV	68	
23		69	Nível baixo tanque de água de alimentação < 1,6 m
24		70	Falha suprimento potência externa
25	Baixo nível em um ou mais GV's ou Supr água alimentação emergencial atuado	71	Perda de potência externa
26		72	Alimentação elétrica de emergência

27	Baixo nível no GV e comparador pressão GVs atuado	73	
28	Atividade na contenção aumentando ou alta	74	Falha transferência suprimento potência externa
29		75	Falha na remoção de calor
30	Resfriamento a 100 k/h atuado	76	Alimentação elétrica de emergência
31		77	Desarme da turbina (com falha da estação de bypass de VP)
32		78	Fechamento inadvertido de uma válvula de isolamento de VP
33		79	Resfriamento parcial iniciado
34	Nível PZR > 2,28 m e Delta P cont./atmosf. de volta a < 30 mbar antes Prefr. < 109 bar	80	
35	Ruptura tubos de gerador de vapor sem violação dos limites de atividade do VP sem iniciação dos CREN	81	
36	Ruptura tubos de gerador de vapor sem violação dos limites de atividade do VP com CREN atingido	82	Suprimento potência auxiliar disponível
37	Ruptura tubos de gerador de vapor com violação dos limites de atividade VP	83	Apenas 1 de 4 VBL linha VP fechada ou resfriamento parcial iniciado em apenas 1 de 4 trens
38	Iniciação espúria dos limites de atividade do VP	84	Falha reposição perda refrigerante
39	Radiat. no secundário aumentando ou leitura ating.	85	Falhas no sistema de controle de volume - falhas na linha de extração com vazão extração muito alta ou falhas na linha de injeção
40		86	Falhas no sistema de controle de volume - falhas na linha de extração com vazão extração muito baixa ou linha bloqueada
41		87	Nível PZR fora do normal sem mudança atmosf. cont.
42	Critérios de refrigeração de emergência núcleo atingidos	88	Nível de água do PZR diminuindo
43	Limite SPR de atividade vapor principal excedido	89	Nível de água do PZR aumentando
44		90	Falha no controle de pressão do SRR
45	Limites SPR atividade VP excedidos espuriamente	91	Não fechamento das válvulas de spray do pressurizador
46	Operação Normal	92	Pressão SRR muito baixa ou vv spray PZR mal posi.
		93	Transferência suprimento potência externa iniciado

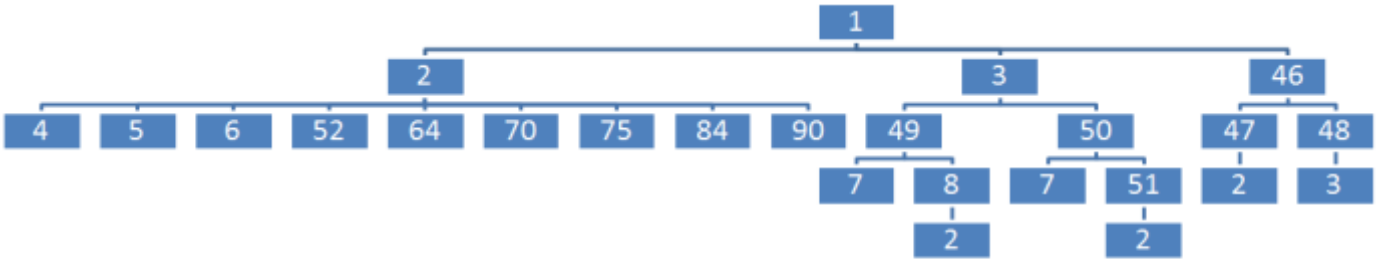
Apêndice II

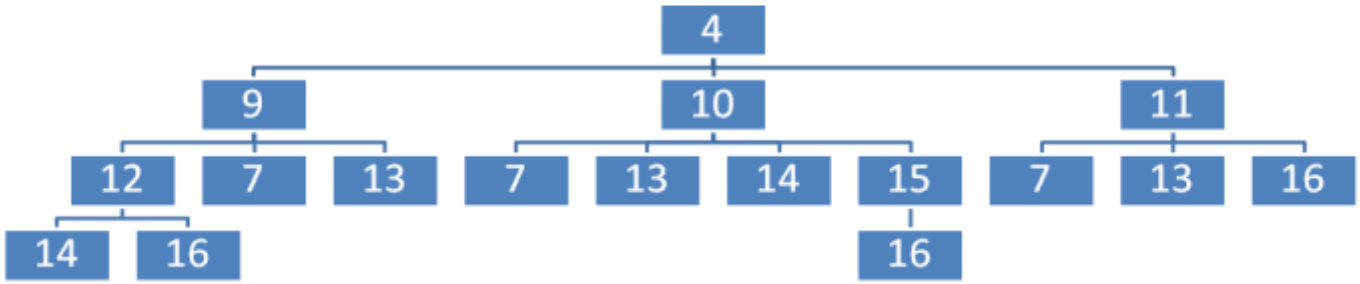
Lista com as Classificações dos Portões Lógicos da Árvore de Falha

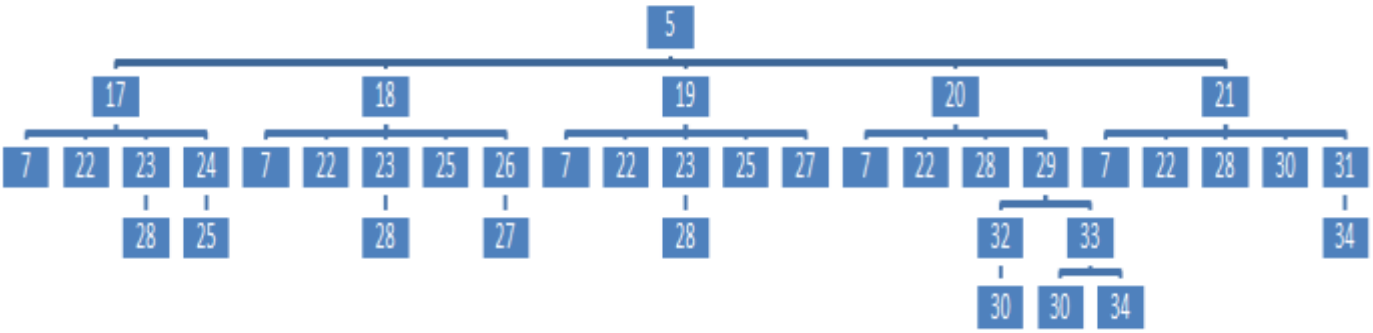
Index do Gate	*Input*	Operador	*Output*	*Index* do Gate	*Input*	Operador	*Output*
1	1	*or*	[2,3,46]	33	47	*not*	[2]
2	2	*or*	[4,5,6,52,64,70,75,84,90]	34	48	*not*	[3]
3	3	*or*	[49,50]	35	49	*and*	[7,8]
4	8	*not*	[2]	36	50	*and*	[7,51]
5	4	*or*	[9,10,11]	37	51	*inhibit-none*	[2]
6	9	*and*	[12,7,13]	38	52	*or*	[53,54,55,56]
7	12	*nor*	[14,16]	39	53	*and*	[7,57,58]
8	10	*and*	[7,13,14,15]	40	58	*not*	[59]
9	15	*not*	[16]	41	54	*and*	[7,57,59,60,61]
10	11	*and*	[7,13,16]	42	60	*not*	[63]
11	17	*and*	[7,22,23,24]	43	61	*not*	[62]
12	23	*not*	[28]	44	55	*and*	[7,57,59,60,62]
13	24	*not*	[25]	45	56	*and*	[7,57,59,63]
14	18	*and*	[7,22,23,25,26]	46	64	*or*	[65,66]
15	26	*not*	[27]	47	65	*and*	[7,67,68]
16	19	*and*	[7,22,23,25,27]	48	68	*not*	[69]
17	20	*and*	[7,22,28,29]	49	66	*and*	[7,67,69]
18	29	*or*	[32,33]	50	70	*or*	[71,72]
19	32	*not*	[30]	51	71	*and*	[7,93,73]
20	33	*and*	[30,34]	52	73	*not*	[74]
21	21	*and*	[7,22,28,30,31]	53	72	*and*	[7,93,74]
22	31	*not*	[34]	54	75	*or*	[76,77,78]
23	5	*or*	[17,18,19,20,21]	55	76	*and*	[7,79,80,81]
24	6	*or*	[35,36,37,38]	56	80	*not*	[83]
25	35	*and*	[7,39,40,41]	57	81	*not*	[82]
26	40	*not*	[43]	58	77	*and*	[7,79,80,82]
27	41	*not*	[42]	59	78	*and*	[7,79,83]
28	36	*and*	[7,39,40,42]	60	84	*or*	[85,86]
29	37	*and*	[7,39,43,44]	61	85	*and*	[7,87,88]
30	44	*not*	[45]	62	86	*and*	[7,87,89]
31	38	*and*	[7,39,43,45]	63	90	*or*	[91]
32	46	*and*	[47,48]	64	91	*and*	[7,92]

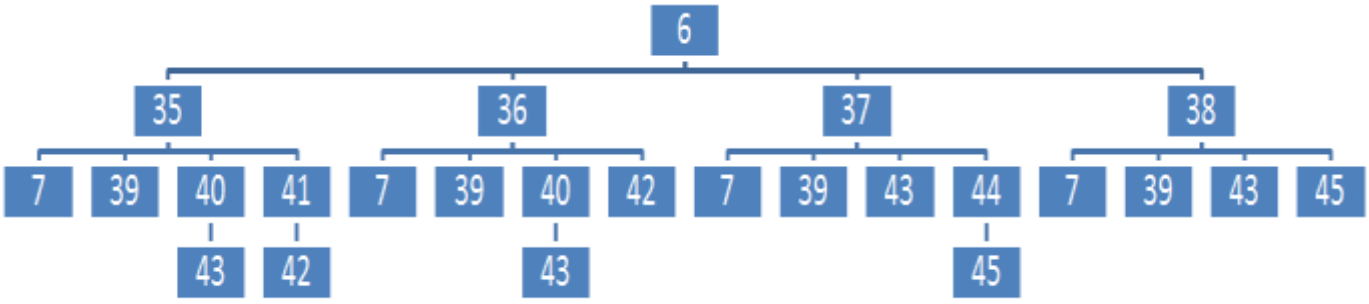
Apêndice III

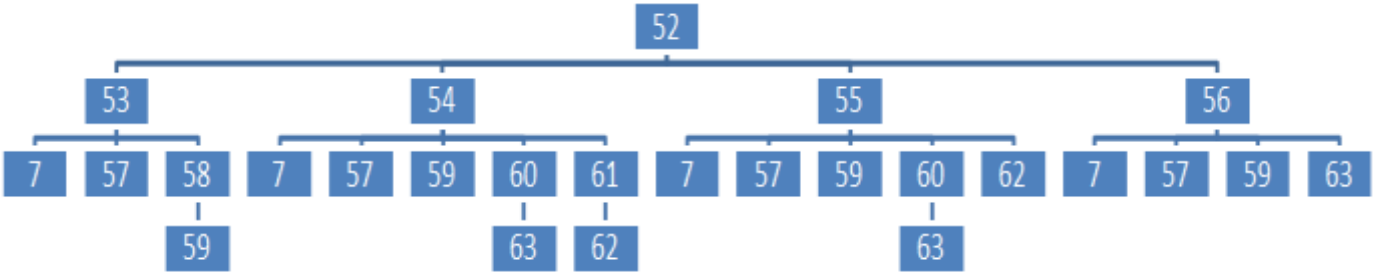
Representação da Árvore de Falha utilizada no Trabalho

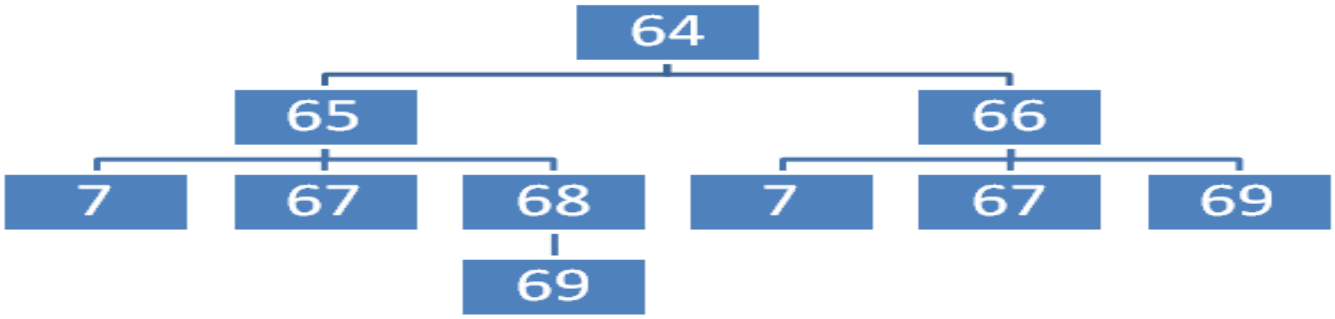

1
2
3
46
4
5
6
52
64
70
75
84
90
49
50
47
48
7
8
7
51
2
3
2
2

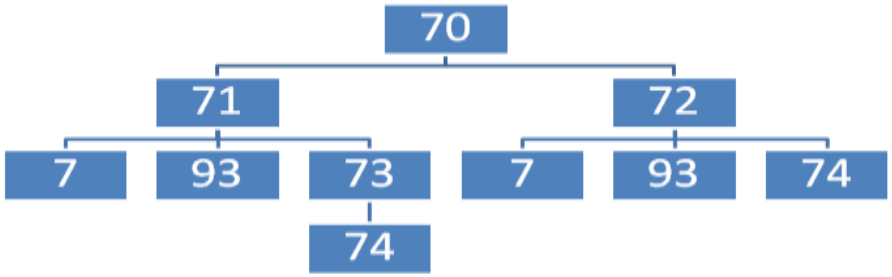

4
9
10
11
12
7
13
7
13
14
15
7
13
16
14
16
16

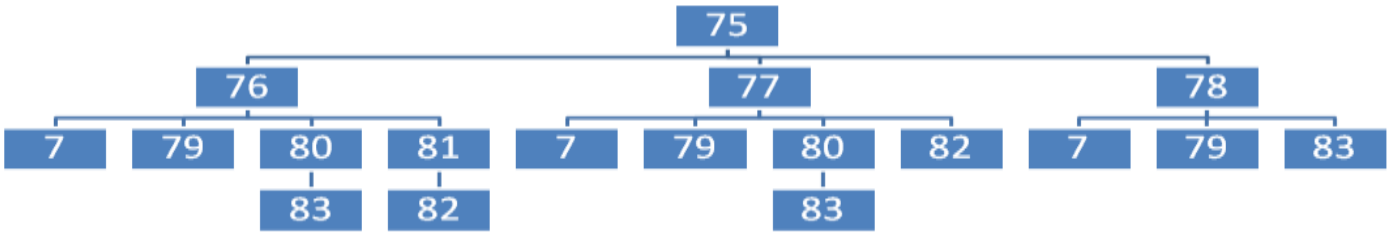

5
17
18
19
20
21
7
22
23
24
7
22
23
25
26
7
22
23
25
27
7
22
28
29
7
22
28
30
31
28
25
28
27
28
32
33
34
30
30
34

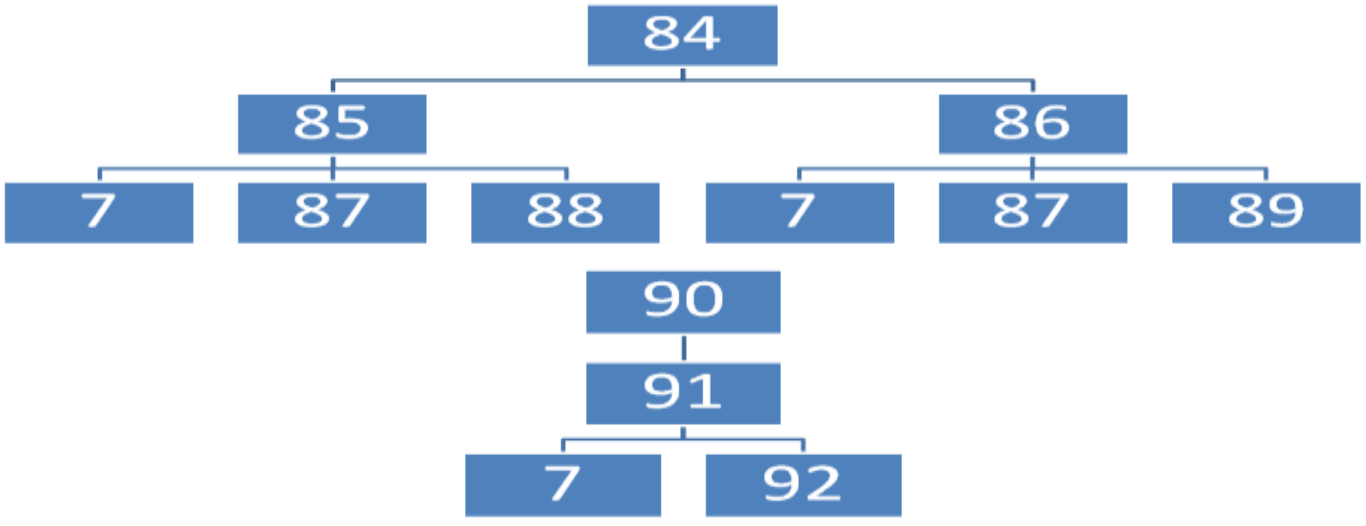

6
35
36
37
38
7
39
40
41
7
39
40
42
7
39
43
44
7
39
43
45
43
42
43
45


52
53
54
55
56
7
57
58
7
57
59
60
61
7
57
59
60
62
7
57
59
63
59
63
62
63


64
65
66
7
67
68
7
67
69
69


70
71
72
7
93
73
7
93
74
74


75
76
77
78
7
79
80
81
7
79
80
82
7
79
83
83
82
83


84
85
86
7
87
88
7
87
89
90
91
7
92