Universidade Federal
do Rio de Janeiro
Escola Politécnica

APLICAÇÃO DE UM SISTEMA ESPECIALISTA PARA O DIAGNÓSTICO EM TEMPO REAL DAS CONDIÇÕES LIMITE DE OPERAÇÃO EM USINAS NUCLEARES

Gustavo Varanda Paiva

Projeto de Graduação apresentado ao Curso de Engenharia Nuclear da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Roberto Schirru

Rio de Janeiro
Março de 2015

APLICAÇÃO DE UM SISTEMA ESPECIALISTA PARA O DIAGNÓSTICO EM TEMPO REAL DAS CONDIÇÕES LIMITE DE OPERAÇÃO EM USINAS NUCLEARES

Gustavo Varanda Paiva

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA NUCLEAR DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO NUCLEAR.

Examinado por:

_____

Prof. Roberto Schirru

_____

Prof. Paulo Fernando Ferreira Frutuoso e Melo

_____

Prof. Claudio Marcio de Nascimento Abreu Pereira

RIO DE JANEIRO, RJ - BRASIL

MARÇO de 2015

Paiva, Gustavo Varanda

Aplicação de Sistema Especialista para Diagnóstico em Tempo Real das Condições Limite de Operação em Usinas Nucleares / Gustavo Varanda Paiva – Rio de Janeiro: UFRJ/ESCOLA POLITÉCNICA, 2015.

X, 55 p.: il.; 29,7 cm.

Orientador: Roberto Schirru

Projeto de Graduação – UFRJ/POLI/ Engenharia Nuclear, 2015.

Referencias Bibliográficas: p. 32-33

1. Sistema Especialista. 2. Condição Limite de Operação. 3. Python. 4. Sistema de Tempo Real. 5. Inteligência Artificial. 6. Árvore de Falha. I. Roberto Schirru. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia Nuclear. III. Aplicação de Sistema Especialista para Diagnóstico em Tempo Real das Condições Limite de Operação em Usinas Nucleares.

A toda minha família

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 do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Nuclear.

APLICAÇÃO DE UM SISTEMA ESPECIALISTA PARA O DIAGNÓSTICO EM TEMPO REAL DAS CONDIÇÕES LIMITE DE OPERAÇÃO EM USINAS NUCLEARES

Gustavo Varanda Paiva

Março/2015

Orientador: Roberto Schirru

Curso: Engenharia Nuclear

Na história da operação de usinas nuclear a segurança é um importante fator a ser considerado e para isso, o uso de matérias resistentes e a aplicação de sistemas redundantes são usados para que a confiabilidade da planta seja elevada. Através da aquisição de experiência com o tempo e com acidentes ocorridos na área, observou-se que a importância de criar métodos que simplifiquem o trabalho do operador na tomada de decisões em cenários de acidente é um fator importante para garantir a segurança das usinas nucleares. Este trabalho tem como objetivo a criação de um programa, usando a linguagem Python, que com o uso de um Sistema Especialista aplique, em tempo real, as regras contidas nas Condições Limite de Operação (CLOs) e informe ao operador a ocorrência de violação das condições limite e a ocorrência de falha ao executar as ações requeridas no tempo para conclusão. A estrutura genérica utilizada para representar o conhecimento do Sistema Especialista foi uma árvore de falha onde os eventos desta árvore são objetos no programa. Para testar a veracidade do programa foi utilizado um modelo simplificado de uma árvore de falha que representa as CLOs da Central Nuclear Almirante Álvaro Alberto 1. Com os resultados obtidos na análise feita no modelo simplificado, foi observada uma significativa redução do tempo para identificação das CLOs e o sucesso da análise em tempo real das CLOs, demonstrando que a aplicação deste programa para modelos mais complexos de árvore de falha seria viável.

Palavras-chave: Inteligência Artificial, Sistema Especialista, Condição Limite de Operação, Árvore de Falha, Python, Sistema de Tempo Real.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Nuclear Engineer.

APPLICATION OF AN EXPERT SYSTEM FOR REAL TIME DIAGNOSIS OF THE LIMITING CONDITIONS FOR OPERATION IN NUCLEAR POWER PLANTS

Gustavo Varanda Paiva

March /2015

Advisor: Roberto Schirru

Course: Nuclear Engineering

In the history of nuclear power plants operation safety is an important factor to be considered and for this, the use of resistant materials and the application of redundant systems are used to make a plant with high reliability. Through the acquisition of experience with time and accidents that happened in the area, it was observed that the importance of creating methods that simplify the operator work in making decisions in accidents scenarios is an important factor in ensuring the safety of nuclear power plants. This work aims to create a program made with the Python language, which with the use of an expert system be able to apply, in real time, the rules contained in the Limiting Conditions for Operation (LCO) and tell to the operator the occurrence of any limiting conditions and the occurrence of failure to perform the required actions in the time to completion. The generic structure used to represent the knowledge of the expert system was a fault tree where the events of this tree are objects in the program. To test the accuracy of the program a simplified model of a fault tree was used that represents the LCO of the nuclear power station named Central Nuclear Almirante Álvaro Alberto 1. With the results obtained in the analysis of the simplified model it was observed a significant reduction in the time to identify the LCO and the success of real-time analysis of LCO, showing that the implementation of this program to more complex models of fault tree would be practicable.

Keywords: Artificial Intelligence, Expert System, Limiting Conditions for Operation, Fault Tree, Python, Real-time analysis

Sumário

1 INTRODUÇÃO 1

2 FUNDAMENTAÇÃO TEÓRICA 6

2.1 Inteligência Artificial 6

2.2 Sistema Especialista 7

2.2.1 Representação do Conhecimento 8

2.2.2 Motor de Inferência 10

2.3 Árvore de Falha 12

2.4 Python 15

3 APRESENTAÇÃO DO PROBLEMA 17

3.1 Relatório de Análise de Segurança 17

3.1.1 Especificações Técnicas 18

3.2 Condições Limite de Operação 19

4 MÉTODOLOGIA 23

5 PROTÓTIPO E RESULTADOS 25

5.1 Protótipo do Programa 25

5.2 Resultados 29

6 CONCLUSÃO E RECOMENDAÇÕES PARA TRABALHOS FUTUROS 31

Referências 32

Apêndice I 34

Apêndice II 50

Apêndice III 52

Lista de Figuras

Figura 1 - Estrutura de um Sistema Especialista. 8
Figura 2 - Exemplificação de uma rede semântica. (Emmanuel Lopes, 1989) 9
Figura 3 - Estrutura Processo de Inferência do Encadeamento para Frente. (SOUTO, 2001) 10
Figura 4 - Busca em Profundidade. (SOUTO, 2001) 12
Figura 5 - Busca em Amplitude. (SOUTO, 2001) 12
Figura 6 - Exemplo de Árvore de Falha. 13
Figura 7 - Simbologia, nomenclatura e função dos portões lógicos (NUREG 0492, 1981). 14
Figura 8 - Características dos modos de operação segundo as Especificações Técnicas (ELETRONUCLEAR, 2012). 20
Figura 9 - Exemplo de uma CLO (ELETRONUCLEAR, 2012). 22
Figura 10 - Diagrama do Sistema Especialista. 25
Figura 11 - Determinação dos atributos designados a cada evento da árvore de falha. 26
Figura 12 - Exemplo do arquivo txt de entrada. 27
Figura 13 - Exemplo de saída do programa. 29

Lista de Siglas

APS – Análise Probabilística de Segurança

CLO – Condição Limite de Operação

CNEN – Comissão Nacional de Energia Nuclear

IAEA – International Atomic Energy Agency

LCO – Limiting Conditions for Operation

RFAS – Relatório Final de Análise de Segurança

RPAS – Relatório Preliminar de Análise de Segurança

SICA – Sistema Integrado de Computadores de Angra

STR – Sistema de Tempo Real

1 INTRODUÇÃO

A geração elétrica por meio de usinas nucleares possui um papel importante na matriz energética mundial onde, segundo a Agência Internacional de Energia Atômica (IAEA), existem 439 reatores em operação em 30 países, totalizando um potencial elétrico de aproximadamente 377 GWe. No Brasil, a geração nucleoelétrica é dada por 2 usinas, que fornecem cerca de 2,3% e o pais possui sua terceira usina nuclear em processo de construção.

Desde o inicio do uso da energia nuclear para fins comerciais, o fator segurança tem sido de extrema importância. Sistemas de alta confiabilidade são usados para garantir sua segurança, porém, com o tempo, observou-se que junto com o uso de sistemas de segurança era necessário possuir uma equipe qualificada, assim como formas eficazes de interação homem-máquina.

Nos acidentes ocorridos em Chernobyl e Three Mile Island foram identificadas diversas falhas devido a fatores humanos, como exemplo, a falha da capacidade cognitiva dos operadores quando um grande número de estímulos e tarefas são apresentados em um curto período de tempo. Com a ocorrência destes acidentes, modificações foram feitas nos projetos das usinas nucleares de forma a simplificar o trabalho dos operadores e aumentar a segurança da usina.

Este trabalho tem como objetivo a criação de um programa que usa Inteligência Artificial para a automatização do processo de identificação de violações nas condições limite de operação (CLO) e manter, em tempo real, os operadores informados em relação às quais ações devem ser realizadas.

As CLOs apresentam diversas condições limitantes para garantir a segurança na usina em seus diferentes modos de operação. Para cada sistema da usina existem diversas CLOs que são identificadas pelo seu nome e sistema de atuação. O Relatório Final de Análise de Segurança apresenta todas as CLOs em sua seção de especificações técnicas. Ao acionar uma CLO, os órgãos fiscalizadores devem ser informados da situação da usina.

Ao violar a condição limitante de uma CLO, esta é dada como acionada e, com isso, devem-se analisar suas condições. Estas condições são organizadas em uma ordem onde se devem executar primeiramente as condições que sejam atendidas e que possuam menor índice nesta ordem. Sendo executada uma condição, deve-se ir à seção das ações requeridas a ela vinculadas. Na presença de mais de uma ação, existirá um operador lógico, podendo este ser “E” ou “OU” que definirá se mais de uma ação deverá ser executada ou se cabe ao operador decidir qual executar.

Ao iniciar uma ação requerida, esta deve ser executada dentro de um tempo limite determinado como tempo para conclusão, caso contrario, a CLO não será atendida ou uma nova condição será executada. A não execução de uma CLO pode fazer com que a usina altere seu modo de operação para que as normas de segurança sejam compridas.

Para a solução deste problema proposto neste trabalho, foi usado um Sistema Especialista, sendo este, uma técnica de Inteligência Artificial que possui como objetivo obter uma solução por meio de inferências. Sua estrutura é composta por uma base de fatos onde são armazenadas as informações do sistema analisado, uma base de regras que armazena as regras do sistema e um motor de inferência que examina os fatos existentes e executa as regras que se aplicam a este sistema.

Para representar este Sistema Especialista foi usada a estrutura de uma árvore de falha. Uma árvore de falha é uma estrutura composta por eventos onde estes são interligados por portões lógicos que possuem operações lógicas como “E” e “OU”. A árvore de falha utilizada neste trabalho foi obtida por meio do programa CAFTA Fault Tree Analysis criado pelo Electric Power Research Institute (EPRI).

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.

Ao se criar o programa, foi executado um caso exemplo onde, para a sua execução, foram realizadas algumas simplificações em relação às regras presentes nas CLOs e estas simplificações serão mencionadas nos próximos capítulos. Ao concluir o trabalho, é apresentado que o exemplo usado é viável e devido à característica do Sistema Especialista, sua aplicabilidade para árvores de falha mais complexas se torna viável também, sendo necessárias algumas alterações para que o programa englobe mais regras presente nas CLOs.

A execução de diagnósticos tem se tornado um dos maiores usos de Sistemas Especialistas devido a sua capacidade de simular o pensamento humano ao resolver os problemas com o uso de heurística (Angeli, 2010). O uso do Sistema Especialista junto a técnicas de análise em tempo real o torna capaz de detectar e prever tarefas. Outro fator que o torna atraente para esta função é a possibilidade do uso de Sistemas Especialista junto a outras técnicas de Inteligência Artificial.

Ao procurar por outros trabalhos cujo foco fosse um Sistema Especialista aplicado ao diagnóstico, não foi encontrado um que tenha como objetivo o uso para resolução das CLOs de usinas nucleares, porem a necessidade de automação ao se diagnosticar alguma falha em um sistema não é uma necessidade única para o problema das CLOs. A seguir serão apresentados outros trabalhos que usam o Sistema Especialista na execução de diagnóstico.

Angeli e Atherton (2001) desenvolveram um Sistema Especialista online para detectar falhas em sistemas hidroelétrico usando conexão online com sensores, métodos de análise de sinais, estratégias baseadas em modelo e técnicas de raciocínio profundo. O conhecimento especialista se encontra principalmente em um modelo de domínio especialista. As conclusões finais do diagnóstico são realizadas após a interação entre as diversas fontes de informação (Angeli, 2010).

Saludes *et al* (2003) apresentaram um modelo para isolamento e detecção de falhas em usinas hidroelétricas baseados em uma rede neural e um subsistema de Sistema Especialista. O Sistema Especialista armazena conhecimento adquirido pelo operador para as conclusões do diagnóstico enquanto a rede neural foi treinada com dados coletados automaticamente durante um ano a fim de decidir entre estado normal ou anormal (Angeli, 2010).

Yu Quian *et al* (2003) apresentaram o desenvolvimento e implantação de um Sistema Especialista para diagnóstico em tempo real de falhas em processos químicos que fornecem sugestão ao operador quando situações anormais ocorrem (Angeli, 2010).

Nabeshima *et at* (2003) apresentaram um Sistema Especialista online para centrais nucleares que usam uma combinação de rede neural e Sistema Especialista, de modo a monitorar e diagnosticar o estado do sistema. O Sistema Especialista usa as saídas da rede neural geradas pela medição dos sinais da planta assim como uma base de conhecimento prévio sobre os reatores de água pressurizada. O coeficiente de energia elétrica é constantemente monitorado pelos sinais de potência ativa e reativa medidos (Angeli, 2010).

De forma a apresentar a ideia proposta neste trabalho, este foi organizado em cinco capítulos que serão descritos a seguir.

No Capítulo 2 são apresentados os fundamentos teóricos necessários para a compreensão do trabalho, onde é introduzido o conceito de Inteligência Artificial e do método de Sistemas Especialista, explicando suas características quanto a sua estrutura e atuação. Como próximo tópico é apresentado o conceito de árvore de falha e é explicado como funciona a sua estrutura. Por fim, apresenta-se a linguagem computacional usada no trabalho, demonstrando suas características.

O Capítulo 3 possui como objetivo apresentar o problema proposto no trabalho e explicar quais ferramentais são necessárias para a sua resolução. Neste capítulo, serão apresentadas as especificações técnicas da usina e as CLOs. Ambos os documentos citados anteriormente estão presentes nos Relatórios de Análise de Segurança, portanto inicialmente serão comentados os objetivos e aplicações deste relatório. Em seguida serão mencionadas as especificações técnicas, que estão presentes em um capítulo do Relatório de Análise de Segurança e seu objetivo é definir os limites de segurança para garantir a integridade da barreira de contenção em caso de liberação de material radioativo. Um dos tópicos presentes nas especificações técnicas são as CLOs que, ao final do capítulo, serão apresentadas. Serão discutidas suas funções e em seguida para que seja possível a compreensão de sua aplicabilidade serão definidos os 6 possíveis modos de operação da usina. Em seguida é informado sobre sua estrutura onde estão presente os dados que a identificam, os fatores limitantes que levam a sua execução, as ações a serem executadas onde é citada a presença dos operadores lógicos nas ações requeridas e a existência de um tempo para a conclusão e seus requisitos de inspeção. Por fim, é apresentado um exemplo de CLO e explica-se como interpretá-lo.

No Capítulo 4 é apresentada a metodologia usada para a resolução do problema proposto. Inicialmente, é analisada a árvore de falha com o intuito de identificar quais são as características que deverão ser transmitidas à estrutura genérica a ser usada no

programa. Em seguida, são apresentadas as simplificações efetuadas para a criação da estrutura genérica. Devido à necessidade de aquisição de dados em tempo real, é apresentado o que é um sistema em tempo real, suas características e como este foi aplicado no trabalho. Por fim é apresentado como é feita a aquisição de dados e quais são as etapas da operação do programa criado em Python para solucionar este problema.

O Capítulo 5 é dividido em demonstração do protótipo e dos resultados obtidos. Em sua primeira parte é apresentado um diagrama esquemático de um Sistema Especialista onde este demonstra as partes constituintes do sistema assim como seus dados de entrada e saída. Também são apresentadas as etapas do processamento do programa e em seguida cada etapa é detalhada. É explicado qual foi o método usado para representar a árvore de falha em uma estrutura que o Python possa interpretar. Por fim, são apresentados a configuração dos dados de entrada e saída do programa. Na seção de resultados são feitos comentários sobre os resultados obtidos com os testes e são propostas ideias para dar continuidade às pesquisas realizadas neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Devido à alta complexidade do problema apresentado, a aplicação da Inteligência Artificial é um método para resolvê-lo de forma rápida e eficiente. Com esta necessidade, neste capitulo será apresentada uma introdução à Inteligência Artificial e ao conceito de Sistema Especialista, que será o método usado para a resolução do problema deste trabalho.

Na resolução do problema é usada a estrutura de árvore de falha o que torna necessário entender quais são as regras aplicadas a esta estrutura e como esta foi criada. Com isso, neste capítulo é esclarecido o que é uma árvore de falha e como funcionam seus operadores lógicos.

Por fim são demonstradas as características da linguagem computacional que será usada para criação dos códigos computacionais do programa.

2.1 Inteligência Artificial

Faz tempo que a inteligência dos seres vivos desperta curiosidade no ser humano. Um dos interesses do homem era a ideia de se criar inteligência de forma artificial. Começou-se a usar o termo Inteligência Artificial no ano de 1956 (STUART *et al*, 1995), porem já existiam pesquisas voltadas para esta área. Com a criação do computador próximo ao ano de 1950, possibilitou-se um maior avanço nesta área devido a sua capacidade de processamento. O computador forneceu um veiculo para a criação da Inteligência Artificial onde este fornecia formas de testar as teorias e ver se estas eram válidas.

O estudo da inteligência não se limita a problemas relacionados com lógica, suas ideias podem ser usadas na resolução de equações matemáticas, resolução de jogos, reconhecimento de imagem e som, operações financeiras e diagnóstico de doenças.

Existem diversas formas de se definir o que é Inteligência Artificial, onde essas ideias se dividem em 4 grupos: Sistemas que pensam como os humanos, sistemas que agem como os humanos, sistemas que pensam racionalmente e sistemas que agem racionalmente (STUART *et al*, 1995). Um exemplo de definição com a ideia de sistemas que agem de forma racional é dada por Schalkoff como “Um campo de estudo que explica e simula o comportamento inteligente em termos de processamento computacional”.

No inicio do conceito de Inteligência Artificial, Alan Turning propôs uma forma de definir se algo possui Inteligência Artificial por meio de um teste nomeado de Teste Turing. Neste teste, o objeto em teste deveria ser interrogado por um humano que desconhecia a natureza de quem ele interrogava. Caso o humano não conseguisse distinguir se o interrogado era um humano ou uma maquina observando apenas as respostas dadas, considerava-se que o objeto possuía Inteligência Artificial.

Como exemplos de aplicações de Inteligência Artificial existem: robótica, visão por computador e processamento de linguagem natural. Destes exemplos, este trabalho se focará nos sistemas especialistas.

2.2 Sistema Especialista

Sistema Especialista é um método de solução vinculado à área de Inteligência Artificial que usa uma base de conhecimento fornecida por um grupo de especialistas no assunto e um modo de inferir este conhecimento de forma a simular a forma de raciocínio usada pelos especialistas ao solucionar problemas.

O que difere um Sistema Especialista é o uso de heurística em sua resolução enquanto nos sistemas convencionais, usa-se um modelo algorítmico. Segundo Emmanuel Lopes Passos em seu livro “Inteligência Artificial e Sistemas Especialistas Ao Alcance de Todos” (PASSOS, 1989), a aplicação de sistemas especialistas se torna vantajosa quando comparadas aos convencionais, nos seguintes casos:

1- Casos com um grande número de combinações que necessitam de muito tempo para que todas as opções sejam avaliadas;
2- Casos onde os processos requerem grandes quantidades de dados ou que haja a necessidade de uso e recuperação da informação com rapidez

A estrutura de um Sistema Especialista é composta por uma interface para o usuário e os seus dois principais componentes que são: A Base de Conhecimento que armazena o conhecimento na forma de regras do problema, que foram fornecidas pelo especialista, e o Motor de Inferência que determina quais regras serão aplicadas, em qual ordem serão aplicadas e obtém os novos fatos e regras com a solução das regras anteriores. A Figura 1 demonstra um modelo da estrutura de um Sistema Especialista.

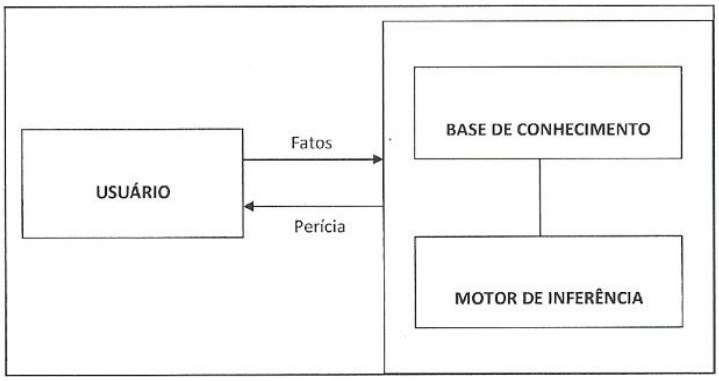


Figura 1 - Estrutura de um Sistema Especialista.

A principal característica que um Sistema Especialista apresenta é a independência entre a Base de Conhecimento e o Motor de Inferência. Esta independência facilita a alteração das informações contidas na Base de Conhecimento, sem que haja a necessidade de alteração no Motor de Inferência. Outras características importantes são a velocidade de resolução e a interação com o usuário devido à possibilidade do programa explicar o raciocínio que o levou à solução.

2.2.1 Representação do Conhecimento

Dentro da área de sistemas especialistas, existem diversas formas de representar o conhecimento que será contido na Base de Conhecimento onde destas formas, se destacam: regras, redes semânticas, frames e orientação a objetos.

O método por regra se assemelha a como os especialistas aplicam seus conhecimentos para resolver um problema. Este método se baseia em uma série de condições que quando são atendidas apresentam uma consequência que também será atendida. Este formato é exemplificado por:

If (Se) <Condição> -> Then (Então) <Consequencia>

Exemplo: If A = True -> Then B = False

O exemplo citado acima é um exemplo de Modus Ponens o que significa que sendo a condição atendida, então se pode obter a consequência, porem, para o caso contrário onde a consequência é conhecida, não é correto afirmar que se conhece o valor da condição.

Em redes semânticas, o conhecimento é apresentado de forma semelhante à criação de frases na linguagem natural. Cada dado é um nó e estes nós se conectam através de arcos que os relacionam. A Figura 2 exemplifica uma rede semântica.

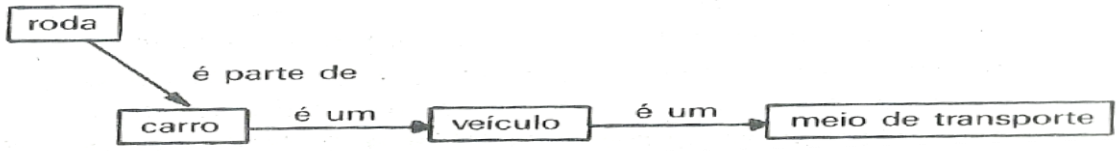


Figura 2 - Exemplificação de uma rede semântica. (Emmanuel Lopes, 1989)

A rede semântica na Figura 2 esta representa a seguinte ideia: “Todo carro tem rodas, um carro em um veículo e um veículo é um meio de transporte.”. Esta rede tem 4 nós que são ligados por 2 tipos de arcos. O arco “é parte de” indica que o nó “roda” é um constituinte do carro e os arcos “é um” indica que o nó de onde ele iniciou é um elemento do conjunto denominado pelo nó onde o arco chega. Um importante conceito de rede semântica é a ideia de herança, onde um nó de nível inferior herda as características vinculadas aos nós superiores a ele.

Como evolução das redes semânticas, surgem os métodos por frames e por orientação a objetos onde ambos surgem no mesmo período e possuem características semelhantes como seu nível estrutural devido a possibilidade de representar o conhecimento por classes hierárquicas. Porem, o destaque da orientação a objeto é o fato de possuir uma clara separação entre os procedimentos e as informações disponíveis (SOUTO, 2001).

O método por frames usa estruturas chamadas Frame (Quadro) e Script (Roteiro) para agrupar toda a informação relacionada a um determinado objeto ou ação.

Objetos consistem de dados e métodos e estes estão estruturados através de classes, instâncias e atributos. As classes definem a estrutura e o comportamento dos objetos, onde todos os objetos pertencentes a uma determinada classe, possuem a mesma estrutura e comportamento. Uma classe pode ser sub-classe de outra classe e com isso herdar características de sua classe superior. Instâncias são objetos pertencentes a uma classe e assim como os objetos, possuem um atributo identificador que se caracteriza por ser uma propriedade inerente.

O modelo de orientação por objeto possui características como o fato de possuir uma total separação entre os dados e os métodos, o que o torna vantajoso em relação ao frame, devido à maior facilidade em alterar o conhecimento, já que, devido à esta separação, encontrar o dado ou método que se deseja alterar se torna mais fácil.

Os modelos de regras e redes semânticas possuem grande desvantagem em relação ao orientado por objeto quando se trata de problemas complexos, onde o número

de nós e regra se torna muito grande, devido à falta de estruturação destes modelos em relação ao orientado por objeto, que apresenta estrutura modulada.

Com o uso da linguagem Python e seu potencial ao trabalhar com listas, foi usado um modelo híbrido para representar as regras no Sistema Especialista. Com o uso das listas presente no Python, este modelo trata os valores relacionados às regras (nome do evento, seu tempo para conclusão e seu tempo em execução) como objetos ao usar de artifícios como o uso de uma lista dentro de outra lista.

2.2.2 Motor de Inferência

O motor de inferência é onde se encontra o "raciocínio" do Sistema Especialista. Ele é responsável por pegar os conhecimentos contidos na base de conhecimento e processá-los. Existem as duas seguintes formas de processar este conhecimento:

1- Encadeamento para a frente (*foward chaining*)
2- Encadeamento para trás (*backward chaining*)

No modelo de encadeamento para a frente, o motor de inferência busca as soluções aplicando as regras aos fatos, partindo dos fatos iniciais. O modo pelo qual o motor de inferência executa este processo se dá com a repetição dos passos demonstrados na Figura 3.

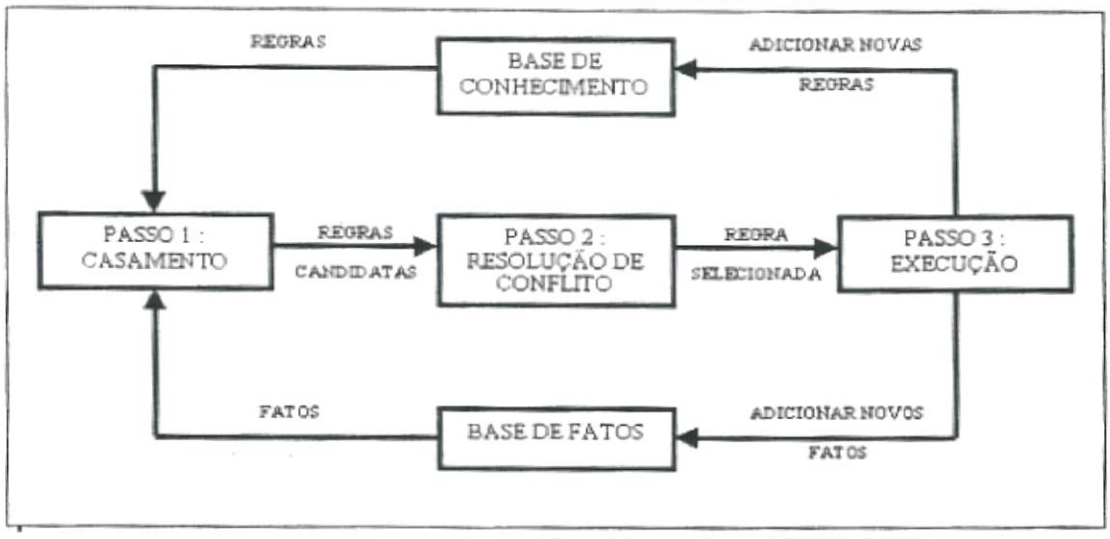


Figura 3 - Estrutura Processo de Inferência do Encadeamento para Frente. (SOUTO, 2001)

Como demonstrado na Figura 3, o método de encadeamento para a frente é constituído de 3 passos, que serão explicados a seguir.

1- Casamento: Neste passo, com a base de fatos atuais, verificam-se quais regras possuem todas as suas condições atendidas e estas regras se tornam candidatas para a fase seguinte.
2- Resolução de conflitos: Após selecionar as regras candidatas, é possível que exista mais de uma candidata e, com isso, pode ser necessária uma forma de decidir qual será a regra aplicada. Este passo é importante para determinar a eficiência do Sistema Especialista, já que o programador pode definir quais serão os critérios a serem consideradas para decidir qual candidata será executada antes. Alguns dos critérios a ser usados são: Não duplicata (Não executar uma regra em um mesmo argumento duas vezes), 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) (STUART *et al*, 1995).
3- Execução: Por fim, a regra selecionada é executada e, com o seu resultado, podem ser formadas novas regras ou fatos.

No encadeamento para trás, o processo parte de uma solução selecionada e de uma base de fatos vazia. Com a solução escolhida, aplicam-se as regras e geram-se novos fatos. Com os novos fatos obtidos, é verificado se existem regras que possam ser aplicadas a estes fatos. Este processo se repete até o ponto onde não existem mais regras que se apliquem aos fatos obtidos.

Junto aos modos de encadeamento, podem-se aplicar duas formas de busca no espaço do problema. Estas são:

1- Busca em profundidade
2- Busca em amplitude

Uma busca em profundidade ocorre quando: “Dentre todos os nodos marcados (nós com chances de ser 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.” (SZWARCFITER L. J. 1984 *apud* KELLING SOUTO, 2001). A Figura 4 demonstra um exemplo para este modo.

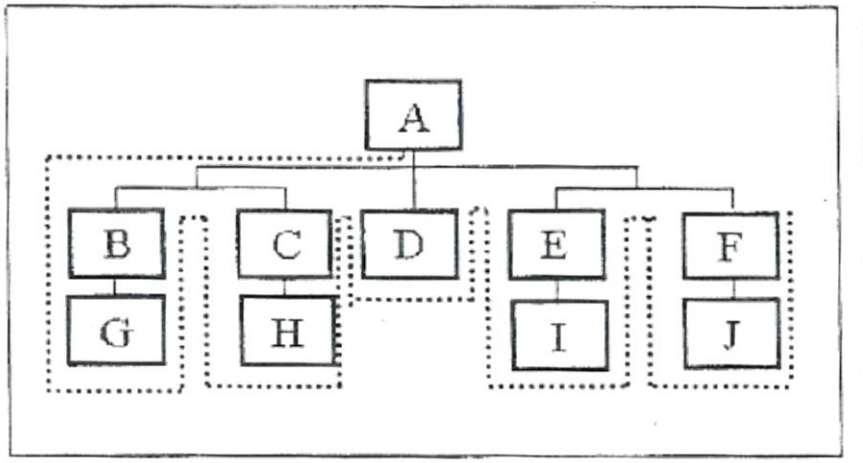


Figura 4 - Busca em Profundidade. (SOUTO, 2001)

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.” (SZWARCFITER L. J. 1984 *apud* KELLING SOUTO, 2001). A Figura 5 demonstra um exemplo para este modo.

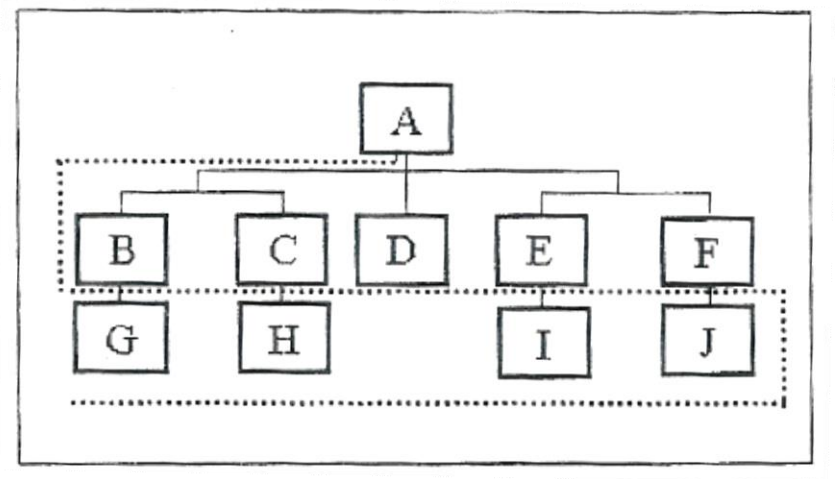


Figura 5 - Busca em Amplitude. (SOUTO, 2001)

2.3 Árvore de Falha

Árvores de falha são representações gráficas que podem representar a relação entre um evento e suas consequências (modelo indutivo) ou de um evento e suas causas (modelo dedutivo). Com essa relação, cria-se um caminho que inicia no evento de topo (Top Event) interligado por portões lógicos, eventos intermediários e eventos iniciadores. Em usinas nucleares ao realizar a Análise Probabilística de Segurança (APS) da instalação, é possível obter a árvore de falha da usina.

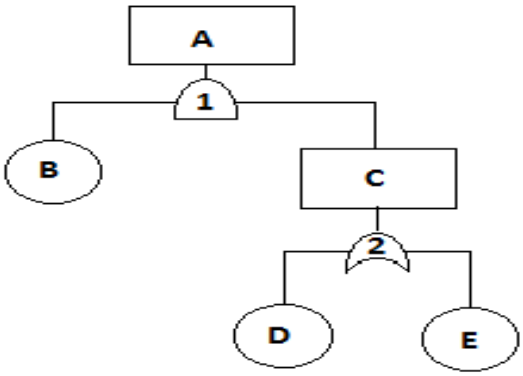


Figura 6 - Exemplo de Árvore de Falha.

No uso de uma árvore de falha, é comum usar alguns termos para caracterizar os eventos. Algumas destes termos são: “folha” ou “leef” para caracterizar o evento iniciador, “nó” para qualquer evento, “pai” para o evento que deu origem ao evento estudado, “filho” para o evento que foi originado pelo evento estudado, “irmãos” para eventos que compartilham do mesmo “pai”. Também será usado o termo “ancestrais” para caracterizar o conjunto de elementos contendo o evento de topo e todos os demais eventos que ligam estes evento com o evento sobre estudo.

Na Figura 6 é apresentado um exemplo da estrutura de uma árvore de falha, onde o elemento A é o evento topo, o elemento C é um evento intermediário, os elementos B, D e E são eventos iniciadores e os elementos 1 e 2 são portões lógicos cujo operações são “E” e “OU”, respectivamente. Este exemplo também serve para demonstrar os termos comentados no parágrafo anterior, por exemplo: o elemento A é pai do elemento B e C, os elementos B e C são irmãos, o elemento C é filho do elemento A, os ancestrais do elemento E são os elementos C e A.

Árvores de falha são bem aplicadas em situações de alto risco, onde o número de componentes é elevado e sistemas que possuem característica complexa. Ao se aplicar uma representação por árvore de falha, se obtêm os seguintes benefícios:

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

Ao se avaliar uma árvore de falha, se obtêm duas formas de resultado: 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, obtenção da análise de importância dos cortes mínimos (NUREG 0492, 1981).

Como já mencionado anteriormente, a estrutura de uma árvore de falha contém eventos e portões. Nos eventos, além de informações como sua identificação e seu estado (verdadeiro, falso ou indefinido), pode-se atribuir outras informações referentes ao evento.

O portão lógico tem como função executar uma operação booleana entre os eventos ao quais se conectam. Os tipos de portões com suas operações e simbologia são demonstrados no texto a seguir e na Figura 7.

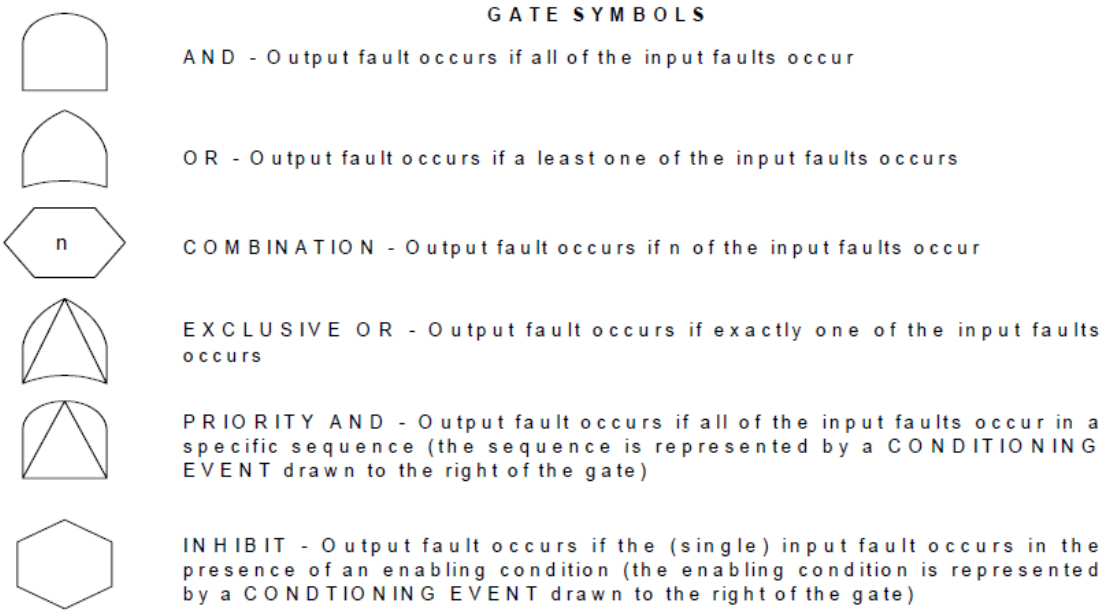


Figura 7 - Simbologia, nomenclatura e função dos portões lógicos (NUREG 0492, 1981).

- 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 “OU” por votação (COMBINATION): O evento de saída é atendido se pelo menos n eventos de entrada forem atendidos.
- Portão “OU” exclusivo (EXCLUSIVE OR): O evento de saída é atendido se apenas um evento de entrada for atendido
- Portão “E” por prioridade (PRIORITY AND): O evento de saída é atendido se todos os eventos de entrada forem atendidos em uma sequencia pré-definida.

- Portão de inibição (INHIBIT): O evento de saída é atendido se o evento de entrada ocorrer conforme uma condição previamente definida. Neste caso, o evento de entrada é um evento único.

É vinculada aos eventos da árvore de falha uma probabilidade do evento ocorrer e com o uso dos operadores lógicos, pode-se calcular a probabilidade para os demais eventos. No problema apresentado neste trabalho, os eventos da árvore de falha também possuem uma probabilidade de ocorrer, porém seus valores são 0 ou 1, indicando se o evento ocorreu ou não. Desta forma, ao se calcular os valores dos demais eventos é possível identificar quais CLOs foram executadas.

2.4 Python

Para a realização de programa é necessário decidir qual linguagem computacional será usada. Neste trabalho, a linguagem escolhida foi Python e a seguir serão mencionados os motivos que levaram a essa decisão.

Um dos motivos para a escolha da linguagem Python foi sua semelhança com a linguagem LISP. LISP é uma linguagem de programação de alto nível que foi criada em 1958 e durante sua existência esta foi constantemente utilizada na solução de problemas envolvendo Inteligência Artificial.

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. Como outras características que o tornam competitivo, pode-se citar o fato de ser uma linguagem gratuita, facilidade de estendê-lo com funções feitas em outras linguagens, o agrupamento das declarações é feito por indentação, que é feita de forma automática e o fato de não ser necessária a declaração de variáveis.

Um recurso que será muito usado neste trabalho é o modulo lista. Uma lista é um vetor que possui seus elementos ordenados, começando por 0 a partir do lado esquerdo. Como elementos de uma lista, podem-se adicionar inteiros, strings ou mesmo outra lista e outros tipos pertencentes ao Python. Com o uso de operadores específicos de lista, pode-se manipulá-las de forma a armazenar um grande número de informações. Outro modulo que será usando neste trabalho é o modulo string. Um string é um conjunto de caracteres apresentado entre um par de aspas e sua utilidade é possibilitar a criação de

textos sem que esse seja um comentário ou uma variável. A seguir, é apresentado um exemplo de lista e de string.

- Lista = [1,[],"texto", False]
- String = "texto !@#$ 1234"

Outro fator importante, usado com a linguagem Python, é a recursividade. Este recurso permite simplificar o programa, porem, em certas condições, pode limitar o espaço de atuação do programa devido ao limite computacional.

Devido à alta capacidade de se usar lista e à presença da recursividade, a linguagem Python possui significativo potencial na área da Inteligência Artificial.

3 APRESENTAÇÃO DO PROBLEMA

É papel do operador obter informação quanto à operabilidade dos componentes e em caso de mau funcionamento, verificar se alguma condição limite de operação está sendo violada para que possam ser realizadas as ações requeridas para normalizar a operação.

Para a segurança da usina, é necessária a avaliação em tempo real de seus componentes, para identificar qualquer violação das condições limite de operação. Esta tarefa é algo de grande complexidade devido à possibilidade de ocorrência de múltiplas falhas e ao grande número de componentes. Com essa complexidade, algo que possa reduzir esse estresse sobre os operadores é de grande utilidade para a operação segura das instalações.

Nos itens seguintes deste capítulo serão apresentadas as funções dos documentos que os operadores usam para verificar se as condições da usina violam alguma condição limite de operação.

3.1 Relatório de Análise de Segurança

Os relatórios de análise de segurança são documentos requeridos pela agência reguladora no processo de licenciamento da usina nuclear. Existem 2 fases neste relatório que são: Relatório Preliminar de Análise de Segurança (RPAS) e Relatório Final de Análise de Segurança (RFAS). Nestes relatórios, são considerados fatores técnicos e humanos relacionados à planta assim como a dependência entre os dois fatores.

No RPAS são dadas informações relacionadas à construção da usina e ao local de construção. Com o relatório, o órgão regulador concede a licença de construção da planta.

Como etapa final, é realizado o RFAS. O RFAS deve conter informações que descrevam a instalação, apresentem as bases de projeto, os limiteis de operação e uma análise de segurança da instalação como um todo. Ao receber este relatório e avaliá-lo, o órgão regulador concede a autorização de operação inicial e permanente (CNEN NE 1.04, 2002).

Outras informações que são apresentadas no RPAS e RFAS são citadas na norma NE 1.04 da CNEN nos itens 6.4 e 8.4 respectivamente.

Após o inicio da operação da usina, o RFAS deve ser atualizado periodicamente com as possíveis alterações feitas no tempo de vida útil da usina.

A seguir é apresentada a estruturação dos capítulos do RFAS de Angra 2 obtidos da tradução da versão em inglês do RFAS

1. Introdução e descrição geral
2. Características do sitio
3. Design das estruturas, componentes, equipamentos e sistemas
4. Reator
5. Sistema de refrigeração do reator e sistemas conectados
6. Recursos de engenharia de segurança
7. Instrumentação e controle
8. Energia elétrica
9. Sistemas auxiliares
10. Sistema de conversão de vapor e potência
11. Gerenciamento de rejeitos radioativos
12. Proteção radiológica
13. Condutas de operação
14. Programa de testes iniciais
15. Análise de Acidente
16. Especificações técnicas
17. Sistema de garantia de qualidade
18. Engenharia de fatores humanos

Para este trabalho, os principais componentes do RFAS são a Análise Probabilística de Segurança (Capítulo 15) para obter a árvore de falha e as Especificações Técnicas (Capítulo 16) que será comentada no tópico seguinte.

3.1.1 Especificações Técnicas

Um dos requisitos para o licenciamento de uma usina nuclear é a apresentação das Especificações Técnicas. Neste documento, são definidos os limites de segurança para os parâmetros estabelecidos na proteção da integridade das barreiras de contenção em caso de liberação de material radioativo.

As Especificações Técnicas são constituídas de 2 volumes. O primeiro, denominado como Especificações, tem como objetivo definir o número mínimo de

equipamentos de segurança necessários para as várias condições de funcionamento e os requisitos de desempenho para estes equipamentos. As especificações técnicas também definem as ações que precisam ser executadas dentro do intervalo de tempo em que o equipamento de segurança não está disponível ou operando com baixo desempenho. O segundo, é denominado de Bases Técnicas e, neste, é explicado o papel dos equipamentos de segurança durante os acidentes de base e explica motivos dos parâmetros usados no que foi estabelecido nas especificações (LOCKBAUM, 2013).

3.2 Condições Limite de Operação

Uma das informações apresentadas nas especificações técnicas são as Condições Limite de Operação. Segundo a CNEN, sua função é estabelecer os níveis mínimos de desempenho ou de capacidade de funcionamento de sistemas ou componentes exigidos para a operação segura da instalação.

Para saber se uma CLO é aplicável a uma situação, é necessário saber em qual modo de operação a usina está. Existem 6 modos de operação que são: 1) Operação à Potência, 2) Partida, 3) Prontidão Quente, 4) Desligado Quente, 5) Desligado Frio, 6) Recarregamento. Na Figura 8 são apresentadas as características dos modos de operação segundo as Especificações Técnicas.

MODO	TÍTULO	CONDIÇÃO DE REATIVIDADE (k_{eff})	% POTÊNCIA TÉRMICA NOMINAL[a]	TEMPERATURA MÉDIA DO REFRIGERANTE DO REATOR °C (°F)
1	Operação à Potência	> 0,99	> 2%	NA
2	Partida	> 0,984	≤ 2%	NA
3	Prontidão Quente	≤ 0,984	NA	≥ 177°C (350°F)
4	Desligado Quente[b]	≤ 0,984	NA	93°C (200°F) < T_{med} < 177°C (350°F)
5	Desligado Frio[b]	≤ 0,99	NA	≤ 93°C (200°F)
6	Recarrega-mento[c]	≤ 0,95	NA	≤ 60°C (140°F)

(a) Excluindo o calor de decaimento.

(b) Todos os parafusos de fechamento da cabeça do vaso do reator totalmente tensionados.

(c) Um ou mais parafusos de fechamento da cabeça do vaso do reator não totalmente tensionado(s), todas as barras inseridas e concentração de boro mantida dentro do limite especificado no Relatório de Projeto Nuclear e Termo-hidráulico RPNT.

Figura 8 - Características dos modos de operação segundo as Especificações Técnicas (ELETRONUCLEAR, 2012).

A estrutura de uma CLO apresenta o sistema à qual ela pertence, seu nome, suas condições e aplicabilidade, suas ações e seus requisitos de inspeção. Dentro das ações, é apresentada uma tabela dividida em três colunas onde essas colunas são nomeadas de Condição, Ação Requerida e Tempo para Conclusão.

Na coluna Condição são apresentadas as condições presente na CLO. Estas condições são indexadas com letras em ordem alfabética para facilitar sua identificação. Uma vez entrando-se na condição, trens, subsistemas, componentes ou variáveis subsequentes expressas na condição, descobertos inoperáveis ou fora dos limites, não resultarão em entradas separadas na condição, a menos que especificamente estabelecido. As ações requeridas da condição continuam aplicáveis para cada falha adicional, onde o tempo para conclusão desta condição é iniciado na ocorrência da primeira falha e não é alterado devido a falhas consecutivas (ELETRONUCLEAR, 2012).

A coluna de Ação Requerida contém as ações a serem feitas caso a condição à qual elas se relacionam seja atendida. Assim como na coluna Condição, as ações requeridas são indexadas com a mesma letra de sua condição, junto a um número em ordem crescente. Esta coluna pode apresentar os operadores lógicos “E” e “OU” conectando uma ou mais ações requeridas.

Na coluna Tempo para Conclusão são apresentados os limites de tempo em que as ações requeridas devem ser cumpridas. Assim que a condição é atendida, o tempo para cada ação requerida é iniciado independente dos outros. Caso o tempo limite seja atingido, a condição relacionada a este tempo é dada como falha e isto pode fazer com que outras condições sejam iniciadas. Outras operações como acréscimo do tempo limite, conectores lógicos e notas pré-definidas também podem ser aplicadas a esta coluna.

Os requisitos de inspeção são apresentados em uma tabela onde é apresentado o que deve ser inspecionado e com qual frequência isto deve ser realizado. Assim como no tempo para conclusão, existem fatores que podem alterar a frequência de inspeção e também existe a possibilidade do uso de operadores lógicos nesta coluna.

Na Figura 9 é apresentado um exemplo de CLO e, em seguida, será explicado como interpretar este exemplo.

16.3.1 SISTEMAS DE CONTROLE DE REATIVIDADE

16.3.1.3 Reatividade do Núcleo

CLO 16.3.1.3 A reatividade do núcleo medida deverá estar dentro de ± 1% Δ k/k dos valores previstos no RPNT.

APLICABILIDADE: MODOS 1 e 2.

AÇÕES

CONDIÇÃO		AÇÃO REQUERIDA		TEMPO PARA CONCLUSÃO
A.	Reatividade do núcleo medida fora dos limites.	A.1	Reavalie o projeto do núcleo e as análises de segurança e determine a aceitabilidade do núcleo do reator para prosseguimento da operação.	72 horas
		E		
		A.2	Estabeleça restrições operacionais e RIs, apropriados.	72 horas
B.	A Ação Requerida e o Tempo para Conclusão associado não cumpridos.	B.1	Esteja em MODO 3.	6 horas

Figura 9 - Exemplo de uma CLO (ELETRONUCLEAR, 2012).

A CLO deste exemplo está relacionada ao sistema de controle de reatividade, mais precisamente à reatividade do núcleo. Como sinalizado pelo número contido em seu nome, esta CLO pode ser encontrada na seção 16.3.1.3 do Relatório Final de Análise de Segurança.

Esta CLO será acionada caso a usina esteja nos modos de operação 1 ou 2 e a reatividade do núcleo não se mantenha dentro dos limites de $\pm 1\%$ do valor previsto no RPNT. Com a CLO sendo acionada, deve-se partir para a condição A e realizar as ações requeridas A.1 e A.2 em um tempo de 72 horas para cada uma, começando a ser contado simultaneamente no instante que a condição A foi atendida. Caso alguma das ações requeridas da condição A não seja realizada no prazo, a condição B será iniciada onde sua ação requerida é de alterar o modo de operação da usina para o modo 3 em um prazo de 6 horas.

4 MÉTODOLOGIA

Como primeira etapa deste trabalho, foi realizado um estudo sobre o conjunto de regras que são apresentadas nas CLOs. Com este estudo observou-se quais características foram necessárias para a criação de uma estrutura genérica que satisfaça as operações necessárias para solucionar o problema estudado.

Com a informação do estudo, foi usado um modelo simplificado como estrutura genérica, onde está presente a capacidade de resolver as operações lógicas “E” e “OU” que podem existir na seção Ação Requerida das CLOs, assim como a manipulação dos cronômetros usados na contagem do tempo usado na seção Tempo para Conclusão. Esta estrutura genérica se apresenta em forma de uma árvore de falha, onde seus elementos são os eventos das CLOs. A árvore de falha usada neste trabalho foi obtida por meio do programa CAFTA Fault Tree Analysis criado pelo Electric Power Research Institute (EPRI).

Devido à característica do Sistema Especialista em separar a base de conhecimento do motor de inferências, o uso de um fragmento da estrutura genérica é capaz de representá-la por inteiro onde a diferença se dá na adição de novos dados à base de conhecimento sem alterar o motor de inferências.

O programa para resolução das CLOs possui aquisição de dados da operação da usina em tempo real. Com a captação de dados em tempo real e o processamento das regras pelo Sistema Especialista, o trabalho efetuado pelos operadores da usina é simplificado pelo fato de não existir mais a necessidade do operador ter que verificar os documentos de segurança em busca das condições da CLO para verificar se o estado atual da usina viola alguma dessas condições.

Um Sistema de Tempo Real (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos (FARINES, FRAGA, OLIVEIRA, 2000). O objetivo de um STR é de 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; tarefas periódicas, onde o processo é executado ciclicamente com um período de tempo por ciclo.

Neste trabalho é usado o modelo de tarefas periódicas onde foi selecionado um período de 10 segundos para cada tomada de dado. Foi considerado o uso de 10

segundos por este valor ser bem superior ao tempo computacional exigido e devido ao fato de, 10 segundos ser um valor aceitável ao se considerar que os eventos ocorridos no início do ciclo são mantidos como verdade durante todo o período.

Os dados coletados no início de cada ciclo são fornecidos pelo Sistema Integrado de Computadores de Angra – SICA (SCHIRRU e PEREIRA, 2004), onde este é responsável pelo monitoramento em tempo real dos parâmetros essenciais para a determinação do estado de segurança da usina no caso de uma situação de emergência, bem como no acompanhamento do funcionamento da mesma durante sua operação normal (NICOLAU, 2014).

Com o uso do Python 2.7.6, foi criado um programa que aplica as regras contidas na árvore de falha. Como dados de entrada iniciais do programa são fornecidos a estrutura da árvore de falha e o estado dos eventos na condição inicial. Com a estrutura informada, a próxima etapa é o processamento dos dados pelo motor de inferência resolvendo a árvore de falha e iniciando os cronômetros caso seja atendida uma condição de uma CLO. Em seguida, é informado ao usuário quais CLOs foram iniciadas ou finalizadas junto do cronometro das condições em aberto. O procedimento citado anteriormente conclui um ciclo e, como inicio do ciclo seguinte, novos dados serão coletados do SICA.

5 PROTÓTIPO E RESULTADOS

5.1 Protótipo do Programa

Na criação do programa deste trabalho foi usada a estrutura apresentada na Figura 10, onde esta representa os dados de entrada do sistema e o Sistema Especialista que é constituído da base de regras, base de fatos e do motor de inferência.

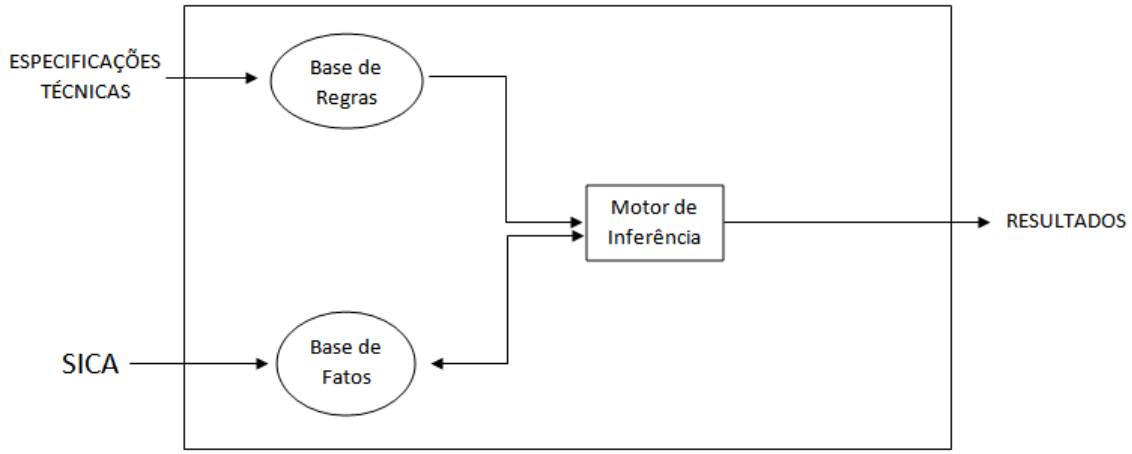


Figura 10 - Diagrama do Sistema Especialista.

Este protótipo é constituído das seguintes etapas que são executadas a cada ciclo: 1) Tomada de dados e criação das variáveis relacionadas aos nodos da árvore de falha; 2) Atuação do motor de inferência na resolução das operações lógicas presente na árvore de falha; 3) Identificação das CLOs que foram executadas, acompanhamento do tempo decorrido após a entrada em cada CLO e verificar se algum tempo decorrido é maior que o tempo para conclusão da mesma; 4) Informar ao usuário as informações analisadas.

Como mencionado acima, a primeira atuação do programa é na entrada de dados referentes à estrutura da árvore de falha e quanto à condição inicial dos eventos. Na estrutura da árvore de falha, cada evento é classificado como um objeto onde este possui atributos para identificá-lo. A Figura 11 é mostra que atributos são estes e nos parágrafos seguintes serão comentados sobre como estes atributos foram adicionados.

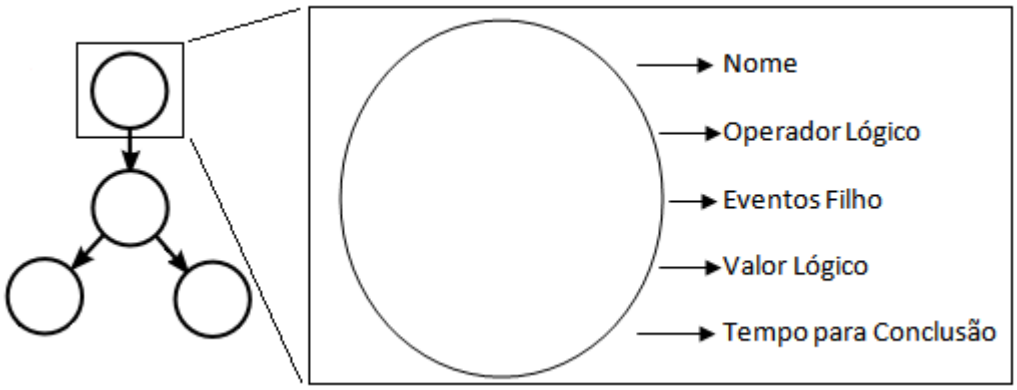


Figura 11 - Determinação dos atributos designados a cada evento da árvore de falha.

Para a formação da árvore de falha é necessária uma forma de identificar os eventos, saber com quais outros eventos este interage (eventos filhos) e qual o operador lógico que governa essa interação. Com o uso da estruturação em lista do Python, foi decidido o seguinte formato para os eventos da árvore de falha.

- Evento = [“identificação do evento”, “operador lógico”, “eventos filhos”]
- Exemplo: Evento = [“A”, “and”, “B,C”]

Na ausência de um operador lógico para o evento, no espaço “operador lógico” é colocado “leef” o identificando como um evento iniciador e no espaço “eventos filhos” usa-se “” para demonstrar que este evento não possui filhos. Caso o evento analisado possua apenas um evento filho, no espaço “operador lógico” usa-se “null”.

A mesma estruturação foi feita para criar uma lista que armazene as informações relacionadas ao valor lógico do evento, podendo este ser verdadeiro “V” ou falso “F” e um valor para armazenar o tempo para conclusão do evento apresentado, em segundos.

- Evento = [“identificação do evento”, “valor lógico”, “tempo para conclusão”]
- Exemplo: Evento = [“A”, “F”, “60”]

No caso do evento analisado não possuir um tempo para conclusão, no espaço “tempo para conclusão” é colocado o valor 0.

Feita a estruturação em lista para todos os eventos, é feita em seguida, uma lista chamada Tree_input para armazenar todas as listas de eventos relacionadas à estrutura da árvore de falha e uma lista Valor_input para armazenar as listas relacionadas aos valores dos eventos. Abaixo é demonstrado um exemplo para ambas.

- Tree_input = [[“A”, “and”, “B,C”], [“B”, “leef”, “”], [“C”, “leef”, “”]]
- Valor_input = [[“A”, “F”, “60”], [“B”, “V”, “0”], [“C”, “F”, “0”]]

Os dados de entrada citados acima são fornecidos por meio de um arquivo txt. Ao ler este arquivo o programa cria as listas Tree_input e Valor_input. A Figura 12 possui um exemplo de um arquivo txt usado como entrada. Este exemplo foi criado usando a árvore de falha citada na Figura 6.

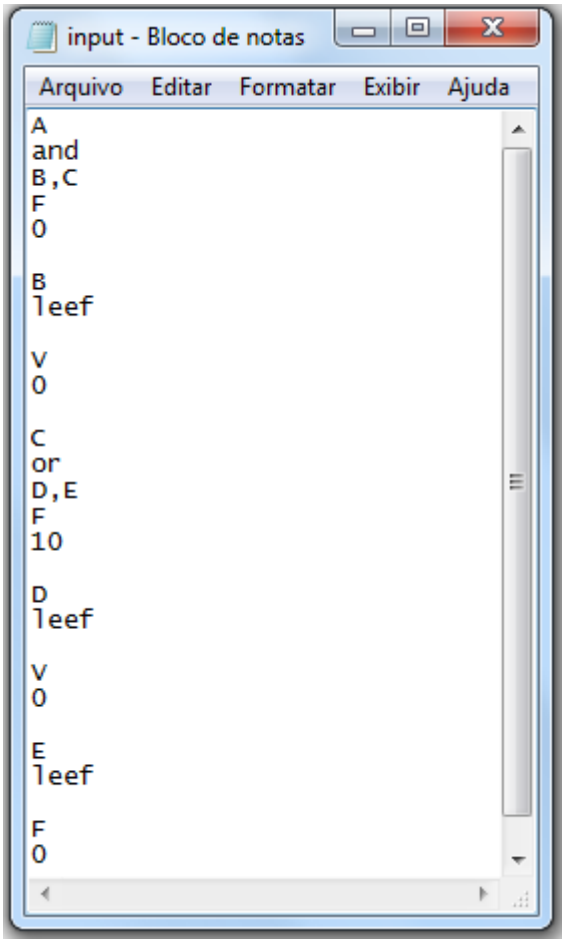


Figura 12 - Exemplo do arquivo txt de entrada.

Com os dados de entradas fornecidos são feitas algumas alterações nas listas Tree_input e Valor_input para a operação interna do programa. Uma destas alterações é a adição de um cronômetro para cada nó da árvore de falha. Este cronômetro só é utilizado caso o nó possua tempo de conclusão diferente de zero e só é iniciado quando os eventos filhos do nó estudado fornecem condições para que o seu valor seja verdadeiro. O cronômetro de um nó é zerado no momento em que os valores de seus filhos param de fornecer as condições para que o seu valor seja verdadeiro. Enquanto o cronômetro de um nó possui valor menor que o limite descrito pelo tempo para conclusão, este nó permanecerá tendo valor falso.

Após serem feitas as alterações nas listas de entrada, são executadas as operações lógicas, onde é realizada uma busca em amplitude na árvore de falha

começando pelo evento de topo. Estas operações lógicas são efetuadas no inicio de cada ciclo e quando o tempo para conclusão de um evento é alcançado.

Em cada ciclo, são apresentados os atuais casos de maior gravidade, que são os eventos que possuem valor verdadeiro e que todos os eventos ancestrais a ele são falsos. Junto com os eventos de maior gravidade, é apresentado o tempo decorrido desde o inicio do primeiro ciclo, quais CLOs foram abertas neste ciclo, quais CLOs tiveram o tempo para conclusão esgotado e o valor de tempo indicado nos cronômetros dos eventos que possuem tempo para conclusão Na Figura 13 é apresentado um exemplo da tela de saída do programa.

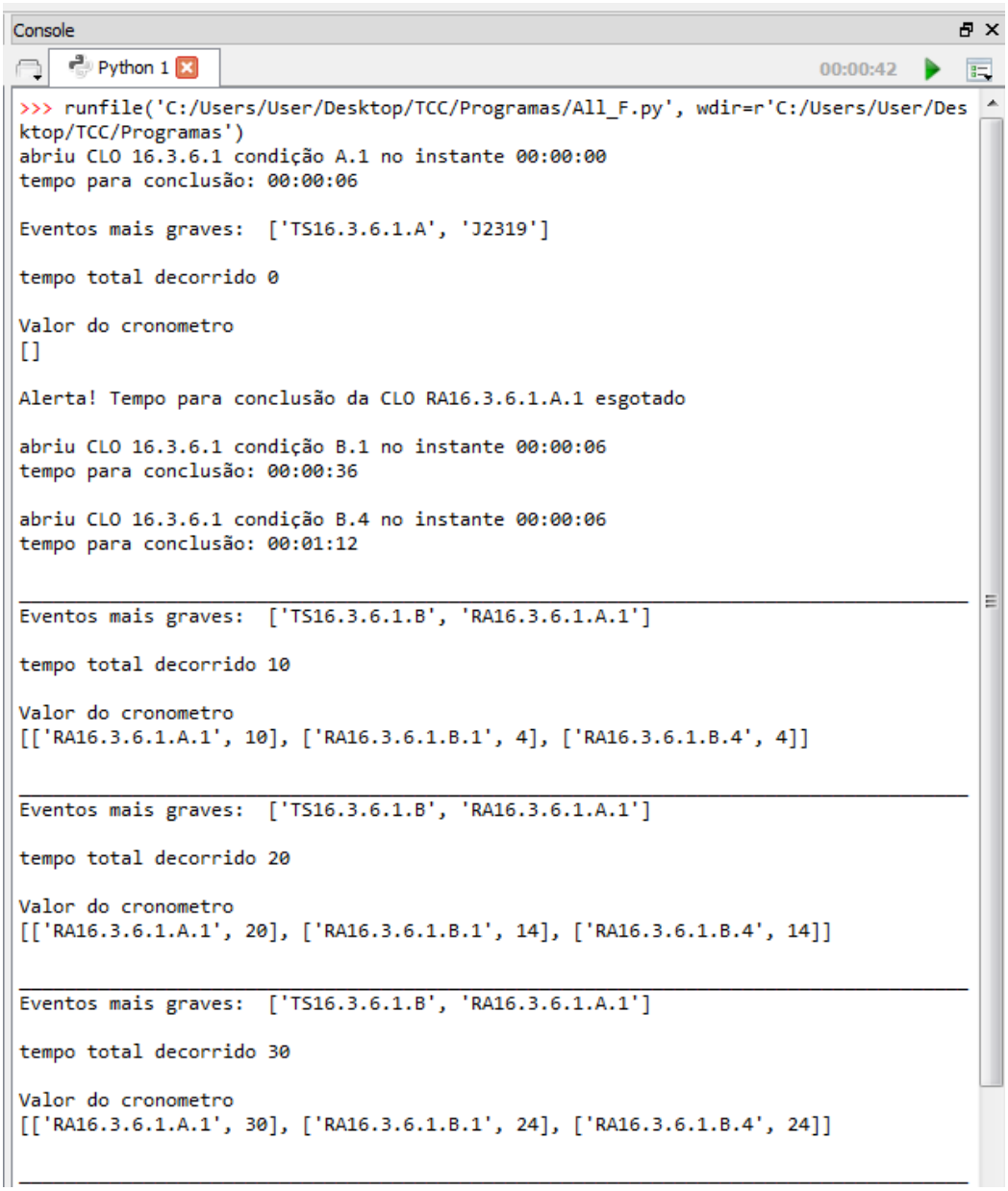


Figura 13 - Exemplo de saída do programa.

No fim do ciclo, é verificado novamente o arquivo txt e, caso este tenha sido alterado, uma nova Valor_input é criada para o próximo ciclo.

5.2 Resultados

O objetivo deste trabalho foi a criação de um programa que possibilitasse a simplificação do trabalho dos operadores da usina ao analisar se as condições de operação da usina violam alguma das CLOs determinadas no projeto. Com essa simplificação, o stress sobre o operador é reduzido, reduzindo também a possibilidade de falhas humanas e, consequentemente, aumentando a segurança da usina.

Para a criação do programa foi usado o modelo de Sistema Especialista onde sua estrutura de conhecimento é constituída de uma árvore de falha onde estão presentes as regras contidas nas CLOs e de uma base de fatos obtida, em tempo real, por meio do sistema SICA da usina nuclear Angra 1.

Com a modelagem da estrutura das regras, foi criado um protótipo do programa que possui como objetivo executar corretamente as operações lógicas contidas nas CLOs, gerenciar o tempo de execução das ações requeridas e informar ao operador quanto a entrada em uma CLO ou na falha ao cumpri-la.

Uma característica do Sistema Especialista é que a adição de novas regras ou fatos ao sistema não implica em alterações no código do programa. Isto ocorre pelo fato da base de fatos e a base de regras estarem separadas do motor de inferência, sendo este ultimo, o responsável pela execução das regras. Devido à característica citada anteriormente, os testes executados para a análise da validade do programa foram realizados usando um fragmento da árvore de falha. Para executar os testes foi selecionado o fragmento da árvore de falha relacionado à CLO 16.3.6.1 e este fragmento é apresentado no Apêndice 3.

Ao realizar os testes, verificou-se exatidão na resolução das operações lógicas, correto funcionamento da aquisição de dados, do gerenciamento do tempo de execução, nas informações transmitidas ao usuário quanto à execução ou falha das CLOs e na execução do acompanhamento das CLOs em tempo real. Com os testes, observou-se uma significativa redução no tempo para avaliar as CLOs, o que conclui o objetivo de reduzir o estresse dos operadores ao realizar estas análises.

6 CONCLUSÃO E RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Com o sucesso dos testes realizados para o fragmento de árvore, pode-se considerar que o programa terá o mesmo desempenho para a árvore de falha completa de usinas nucleares, já que apenas haverá adição de novas regras e fatos.

A aplicação de Sistemas Especialista para diagnóstico tem sido uma ferramenta de sucesso vastamente utilizada. Por meio deste trabalho é proposto pela primeira vez o uso de Sistemas Especialista para o diagnóstico das CLOs de uma usina nuclear.

Como forma de dar continuidade às pesquisas realizadas neste trabalho, pode-se adicionar as regras presentes nas CLOs que foram simplificadas neste trabalho e testar sua validade ao usar os dados gerados pelo simulador de Angra 1. Deve-se também realizar testes para garantir a confiabilidade do software usado. Este procedimento é constantemente usado como forma de garantir que o software é confiável e que possui baixa probabilidade de por a segurança do sistema em risco. Outra sugestão para trabalho futuro é realizar uma simulação com operadores com o intuito de obter uma estimativa do tempo levado para realizar manualmente o acompanhamento das CLOs e comparar esta estimativa ao tempo de processamento do programa realizado neste trabalho.

Referências

ANGELI, C., **Diagnostic Expert Systems: From Expert's Knowledge to Real-Time Systems**, 2010. Disponível em: http://www.tmrfindia.org/eseries/ebookv1-c4.pdf. Acesso em: março de 2015.

CAFTA Fault Tree Analysis, Electric Power Research Institute. Disponível em: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001015514. Acesso em: março de 2015

CNEN NE 1.04, **Licenciamento de Instalações Nucleares, Comissão Nacional de Energia Nuclear**, Brasil, 2002.

FARINES, J., FRAGA, J., OLIVEIRA, R., **Sistemas de Tempo Real**, Disponível em: http://www.das.ufsc.br/~romulo/livro-tr.pdf. Acesso em: março de 2015.

LOCHBAUM, Dave, **Nuclear Energy Activist Toolkit #5: Technical Specifications**. Disponível em: http://allthingsnuclear.org/nuclear-energy-activist-toolkit-5-technical-specifications/. Acesso em: março de 2015.

NICOLAU, A. S., ***Algoritmo Evolucionário de Inspiração Quântica Aplicado na Otimização de Problemas da Engenharia Nuclear***. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2014.

NUREG-0492, **Fault Tree Handbook**, US Nuclear Regulatory Commission, Washington, DC, 1981.

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

Relatório Final de Análise de Segurança, Central Nuclear Almirante Álvaro Alberto unidade 2, Eletrobrás Termonuclear S.A. – Eletronuclear, Rev.: 10, Maio de 2007.

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

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.

Especificações Técnicas, Central Nuclear Almirante Álvaro Alberto unidade 1, Eletrobrás Termonuclear S.A. – Eletronuclear, Rev.0, Rio de Janeiro, Brasil, 2012.

VESELY, Bill, **Fault Tree Analysis (FTA): Concepts and Applications**. Disponível em: http://www.hq.nasa.gov/office/codeq/risk/docs/ftacourse.pdf. Acesso em: março de 2015.

Apêndice I

Programa de Execução do Sistema Especialista

(em linguagem Python 2.7.6)

```
# -*- coding: cp1252 -*-
"""
Created on Mon Aug 18 19:42:35 2014
@author: User
"""
from time import *
from random import *
import copy

#Tree: [nome,operador,[filhos],nº de repetições,profundidade]
#Valor: [nome,valor logico,tempo maximo(s),determidador de leitura,tempo rodado(s)]
f = open("input.txt","r")

# leitor do arquivo txt
fim = 0
Tree_input = []
Valor_input = []
while fim == 0:
    lista1 = []
    lista2 = []
    for i in range(0,5):
        text = f.readline()
        if text[-1:] == "\n":
            text = text[:-1]
        if text == "" and i == 0:
            fim = 1
            break
        if i in [0,1,2]:
            if i <> 2:
                lista1.append(text)
            if i == 2:
                lista1.append([text])
        if i in [0,3,4]:
            lista2.append(text)
```

```
    if lista1 == [] or lista2 == []:
        pass
    else:
        Tree_input.append(lista1)
        lista2[2] = int(lista2[2])
        lista2.append(0)
        lista2.append(0)
        Valor_input.append(lista2)
    text = f.readline()
f.close()

Tree_i = copy.deepcopy(Tree_input)
Valor_i = copy.deepcopy(Valor_input)

time_list = []
c = 0

def add(ind,num):
    for i in range(len(Tree)):
        if ind == Tree[i][0]:
            if Tree[i][2] <> [""]:
                familha[num][1] = familha[num][1] + Tree[i][2]
                for j in Tree[i][2]:
                    add(j,num)

def filhos(ind):
    for i in range(len(Tree)):
        if Tree[i][0] == ind:
            if Tree[i][2] <> [""]:
                for j in Tree[i][2]:
                    filho.append(j)
                for j in Tree[i][2]:
                    filhos(j)
```

```
def profundidade(Termo):
    for z in range(len(Tree)):
        if Tree[z][0] == Termo:
            filhos = Tree[z][2]
            if filhos <> [""]:
                Tree[z][4] = Tree[z][4] + 1
                for i in filhos:
                    profundidade(i)
            else:
                Tree[z][4] = Tree[z][4] + 1

def Ancestral(i):
    for j in range(len(Valor)):
        if Tree[i][0] in Tree[j][2]:
            ancestral.append(Tree[j][0])
            Ancestral(j)

def logica(l_list):
    # função de execução das operaçoes logicas
    prof = 1
    cont = -1
    while cont <> 0:
        cont = 0
        while prof <= max_prof - 1:
            for i in range(len(Tree)-contador):
                if i in l_list or Valor[i][4] >= Valor[i][2]:
                    sin = 1
                else:
                    sin = 0
                if Tree[i][4] == prof:
                    o_ind = Tree[i][1]
                    f_ind = Tree[i][2]
                    p = len(f_ind)
                    if Valor[i][2] <> 0 and sin == 0:
```

```
                Valor[i][1] = "F"
                Valor[i][3] = 1
                if Valor[i][3] == 1:
                    for a in range(len(Valor)):
                        if str.find(Valor[a][0],"-:-") <> -1 and
Valor[a][0][:str.find(Valor[a][0],"-:-")] == Valor[i][0]:
                            Valor[a][1] = Valor[i][1]
                            Valor[a][3] = Valor[i][3]
            else:
                for j in range(len(Tree)):
                    if Tree[j][0] in f_ind and Valor[i][3] == 0:
                        if o_ind == "and":
                            if Valor[j][1] == "F" and Valor[j][3] == 1:
                                Valor[i][1] = "F"
                                Valor[i][3] = 1
                            if Valor[j][1] == "V":
                                p = p - 1
                                if p == 0:
                                    Valor[i][1] = "V"
                                    Valor[i][3] = 1
                            if Valor[i][3] == 1:
                                for a in range(len(Valor)):
                                    if str.find(Valor[a][0],"-:-") <> -1 and
Valor[a][0][:str.find(Valor[a][0],"-:-")] == Valor[i][0]:
                                        Valor[a][1] = Valor[i][1]
                                        Valor[a][3] = Valor[i][3]
                        if o_ind == "or":
                            if Valor[j][1] == "V":
                                Valor[i][1] = "V"
                                Valor[i][3] = 1
                            if Valor[j][1] == "F" and Valor[j][3] == 1:
                                p = p - 1
                                if p == 0:
                                    Valor[i][1] = "F"
```

```
                                Valor[i][3] = 1
                            if Valor[i][3] == 1:
                                for a in range(len(Valor)):
                                    if str.find(Valor[a][0],"-:-") <> -1 and
Valor[a][0][:str.find(Valor[a][0],"-:-")] == Valor[i][0]:
                                        Valor[a][1] = Valor[i][1]
                                        Valor[a][3] = Valor[i][3]
                        if o_ind == "null":
                            if Valor[j][1] == "V":
                                Valor[i][1] = "V"
                                Valor[i][3] = 1
                            if Valor[j][1] == "F" and Valor[j][3] == 1:
                                Valor[i][1] = "F"
                                Valor[i][3] = 1
                            if Valor[i][3] == 1:
                                for a in range(len(Valor)):
                                    if str.find(Valor[a][0],"-:-") <> -1 and
Valor[a][0][:str.find(Valor[a][0],"-:-")] == Valor[i][0]:
                                        Valor[a][1] = Valor[i][1]
                                        Valor[a][3] = Valor[i][3]
            prof = prof + 1
        for a in range(len(Valor)):
            if Valor[a][3] == 0:
                cont = 1
        prof = 1
    for i in range(len(Valor)):
        if Valor[i][2] <> 0:
            foi = 0
            for j in range(len(Valor)):
                if Valor[j][0] in Tree[i][2]:
                    if Valor[j][1] == "V":
                        foi = foi +1
            if (Tree[i][1] == "and" and foi == len(Tree[i][2])) or ((Tree[i][1] == "or" or
Tree[i][1] == "null") and foi > 0):
```

```
            if Valor[i][4] == 0:
                nome = Tree[i][0][2:]
                clo = ""
                for z in range(4):
                    p = str.find(nome,".")
                    clo = clo + nome[:p+1]
                    nome = nome[p+1:]
                mm, ss = divmod(t + c,60)
                hh, mm = divmod(mm,60)
                print "abriu CLO", clo[:-1], "condição", nome, "no instante " +
"%02d:%02d:%02d" % (hh, mm, ss)
                mm, ss = divmod(Valor[i][2],60)
                hh, mm = divmod(mm,60)
                print "tempo para conclusão: " + "%02d:%02d:%02d" % (hh, mm, ss), "\n"
T_exp = 100
# T_exp é o tempo que o esperimento ira rodar em segundos.
tempo = clock()
t = int(clock() - tempo)
while int(clock() - tempo) <= T_exp:
    Tree = Tree_input
    Error = False

#########################################################################
#########
    # adiciona em Tree o termo que determina quantos individuos iguais existem
    for i in range(len(Tree_input)):
        Tree[i].append(0)

#########################################################################
#########
    # parte que faz o programa aceitar qualquer ordem de imput no Tree e Valor
    Valor = []
    for i in range(len(Tree_input)):
        for j in range(len(Valor_input)):
```

```
            if Tree_input[i][0] == Valor_input[j][0]:
                Valor.append(Valor_input[j])
                break

#########################################################################
#########
    # separa os individuos da lista de filhos
    # adiciona em Tree o termo que determina a profundidade
    for i in range(len(Tree)):
        for j in range(len(Tree[i][2])):
            Tree[i][2][j] = Tree[i][2][j].split(",")
        Tree[i].append(0)

    for i in range(len(Tree)):
        Tree[i][2] = Tree[i][2][0]

#########################################################################
#########
    # determina quantos termos iguais existem
    for i in range(len(Tree)):
        for j in range(len(Tree)):
            if Tree[i][0] in Tree[j][2]:
                Tree[i][3] = Tree[i][3] + 1
    for i in range(len(Tree)):
        if Tree[i][3] == 0:
            Tree[i][3] = 1

    quant = []
    for i in range(len(Tree)):
        quant.append(Tree[i][3])

    for i in range(len(Tree)):
        if Tree[i][3] > 1:
            filho = []
```

```
            filhos(Tree[i][0])
            for j in range(len(Tree)):
                if Tree[j][0] in filho:
                    quant[j] = quant[j] + filho.count(Tree[j][0])

    for i in range(len(quant)):
        Tree[i][3] = quant[i]

#########################################################################
#########
    # adiciona uma replica dos termos que se repetem
    contador = 0
    for i in range(len(Tree)):
        q = Tree[i][3]
        v = 1
        while q > 1:
            Tree.append([Tree[i][0]                    +                    '-:-'                    +
str(v),Tree[i][1],Tree[i][2],[0,Tree[i][0]],Tree[i][4]])
            Valor.append([Valor[i][0]                    +                    '-:-'                    +
str(v),Valor[i][1],Valor[i][2],Valor[i][3],Valor[i][4]])
            q = q - 1
            v = v + 1
            Tree[i][3] = 1
            contador = contador + 1
    for i in range(len(Tree)):
        Tree[i] = Tree[i][:]
        Tree[i][2] = Tree[i][2][:]
        try:
            Tree[i][3] = Tree[i][3][:]
        except (TypeError):
            pass
    for i in range(len(Tree)):
        cont = 0
        cont2 = 0
```

```
        try:
            if Tree[i][3][0] == 0:
                for j in range(len(Tree)):
                    if Tree[i][3][1] in Tree[j][2] and cont2 == 0:
                        if cont <> 0:
                            for z in range(len(Tree[j][2])):
                                if Tree[j][2][z] == Tree[i][3][1]:
                                    Tree[j][2][z] = Tree[i][0]
                                    cont2 = 1
                        cont = cont + 1
                    if cont2 == 1:
                        break
        except (SyntaxError,TypeError):
            pass
    for i in range(len(Tree)):
        Tree[i][3] = 1

##########################################################################
#########
    # define o conceito de familha e de profundidade e os aplicam
    familha = []
    for i in range(len(Tree)):
        familha.append([Tree[i][0],[]])

    for i in range(len(Tree)):
        try:
            add(Tree[i][0],i)
        except (RuntimeError):
            Error = True
    for i in range(len(Tree)):
        for j in range(len(Tree)):
            while familha[i][1].count(Tree[j][0]) > 1:
                familha[i][1].remove(Tree[j][0])
```

```
    for a in Tree:
        try:
            profundidade(a[0])
        except (RuntimeError):
            Error = True

#########################################################################
#########
    # determina a profundidade maxima e determina que todos os termos "V" já forao
lidos
    if Error == False:
        max_prof = 1
        for a in range(len(Tree)):
            if Tree[a][4] > max_prof:
                max_prof = max_prof + 1

        for i in range(len(Tree)):
            if Valor[i][1] == "V" or Tree[i][1] == "leef":
                if Valor[i][3] == 0:
                    Valor[i][3] = 1
                if Valor[i][3] == 2:
                    Valor[i][3] = 0
            if Valor[i][1] == "F" and Valor[i][3] == 2:
                Valor[i][3] = 0

#########################################################################
#########
        ancestrais = []
        for i in range(len(Valor)):
            ancestral = []
            Ancestral(i)
            ancestrais.append([Valor[i][0],ancestral])
    # parte logica
        logica([])
```

```
#########################################################################
#########
   # identifica os eventos mais graves
      eventos = []
      for z in range(max_prof, 0,-1):
         for i in range(len(Valor)):
            if Tree[i][4] == z:
               ind = Tree[i][0]
               if Valor[i][1] == "V":
                  for j in familha[i][1]:
                     if j in eventos:
                        eventos.remove(j)
                  eventos.append(ind)

      for i in range(len(Tree)):
         for j in range(len(Tree[i][2])):
            if str.find(Tree[i][2][j],"-:-") <> -1:
               Tree[i][2][j] = Tree[i][2][j][:str.find(Tree[i][2][j],"-:-")]
      tirar = []
      for i in range(len(eventos)):
         try:
            if str.find(eventos[i],"-:-") <> -1:
               if eventos[i][:str.find(eventos[i],"-:-")] not in eventos:
                  eventos.append(eventos[i][:str.find(eventos[i],"-:-")])
               tirar.append(eventos[i])
         except (IndexError):
            pass
      for i in tirar:
         eventos.remove(i)
      print "Eventos mais graves: ", eventos, "\n"
   if Error == True:
      print "Error! Arvore invalida. Causa: um individuo filho de si mesmo"
      break
```

```
#########################################################################
#########
   Tree_input = copy.deepcopy(Tree_i)
   t = int(clock() - tempo)
   print "tempo total decorrido", t,"\n"

   for i in range(len(Valor)):
      for j in range(len(time_list)):
         if Valor[i][0] == time_list[j][0]:
            conta = 0
            for k in range(len(Valor)):
               if Valor[k][0] in Tree[i][2] and Valor[k][1] == "F":
                  conta = conta + 1
            if (Tree[i][1] == "and" and conta > 0) or ((Tree[i][1] == "or" or Tree[i][1] ==
"null") and conta == len(Tree[i][2])):
               time_list[j][1] = 0

   print "[CLO, Valor em seu cronometro, tempo para conclusão]"
   print time_list, "\n"

   if t < T_exp:
      c = 0
      while c < 10:
         passa = []
         sleep(1)
         c = c + 1
         for i in range(len(Valor)):
            if Valor[i][0] in passa:
               pass
            else:
               for f in range(len(time_list)):
                  if Valor[i][0] == time_list[f][0]:
                     conta = 0
```

```
                    for j in range(len(Valor)):
                        if Valor[j][0] in Tree[i][2] and Valor[j][1] == "V":
                            conta = conta + 1
                    if (Tree[i][1] == "and" and conta == len(Tree[i][2])) or ((Tree[i][1] ==
"or" or Tree[i][1] == "null") and conta > 0):
                        time_list[f][1] = time_list[f][1] + 1
                if Valor[i][2] <> 0 and t == 0:
                    conta = 0
                    for j in range(len(Valor)):
                        if Valor[j][0] in Tree[i][2] and Valor[j][1] == "V":
                            conta = conta + 1
                    if (Tree[i][1] == "and" and conta == len(Tree[i][2])) or ((Tree[i][1] ==
"or" or Tree[i][1] == "null") and conta > 0):
                        Valor[i][4] = Valor[i][4] + 1
                        if c == 1:
                            time_list.append([Valor[i][0],1,Valor[i][2]])
                    else:
                        if c == 1:
                            time_list.append([Valor[i][0],0,Valor[i][2]])
                if Valor[i][2] <> 0 and t <> 0:
                    for q in range(len(time_list)):
                        if Valor[i][0] == time_list[q][0]:
                            Valor[i][4] = time_list[q][1]
                if Valor[i][2] <> 0 and Valor[i][4] == Valor[i][2]:
                    print "Alerta! Tempo para conclusão da CLO", Valor[i][0], "esgotado",
"\n"
                    for ind in ancestrais[i][1]:
                        passa.append(ind)
                        for j in range(len(Valor)):
                            if Valor[j][0] == ind:
                                Valor[j][3] = 0
                    Valor[i][3] = 0
                    logica([i])
                    if Valor[i][0] == "RA16.3.6.1.A.1":
```

```
                trocar = [i]
                for j in range(len(Valor)):
                    if str.find(Valor[j][0],"RA16.3.6.1.A.1X") <> -1:
                        trocar.append(j)
                        for ind in ancestrais[j][1]:
                            passa.append(ind)
                            for k in range(len(Valor)):
                                if Valor[k][0] == ind:
                                    Valor[k][3] = 0
                        Valor[j][3] = 1
                        Valor[j][1] = "V"
                logica(trocar)

while contador >= 1:
    Tree.pop()
    Valor.pop()
    contador = contador - 1

for i in range(len(Valor_input)):
    Valor_input[i][3] = 0
f = open("input.txt","r")
fim = 0
Valor_futuro = []
while fim == 0:
    lista2 = []
    for i in range(0,5):
        text = f.readline()
        if text[-1:] == "\n":
            text = text[:-1]
        if text == "" and i == 0:
            fim = 1
            break
        if i in [0,3,4]:
            lista2.append(text)
```

```
            if lista2 == []:
                pass
            else:
                lista2[2] = int(lista2[2])
                lista2.append(0)
                lista2.append(0)
                Valor_futuro.append(lista2)
            text = f.readline()
        f.close()

        if Valor_futuro <> Valor_i:
            Valor_input = Valor_futuro
            Valor_i = copy.deepcopy(Valor_futuro)
            for i in range(len(time_list)):
                time_list[i][1] = 0

        print
"_____
_____"
    else:
        break
```

Apêndice II

Fragmento da Condição Limite de Operação 16.3.6.1.

(Especificações Técnicas Angra 1. Rev. 0)

16.3.6 SISTEMAS DA CONTENÇÃO

16.3.6.1 Contenção

CLO 16.3.6.1 A contenção deve estar OPERÁVEL.

APLICABILIDADE: MODOS 1, 2, 3 e 4.

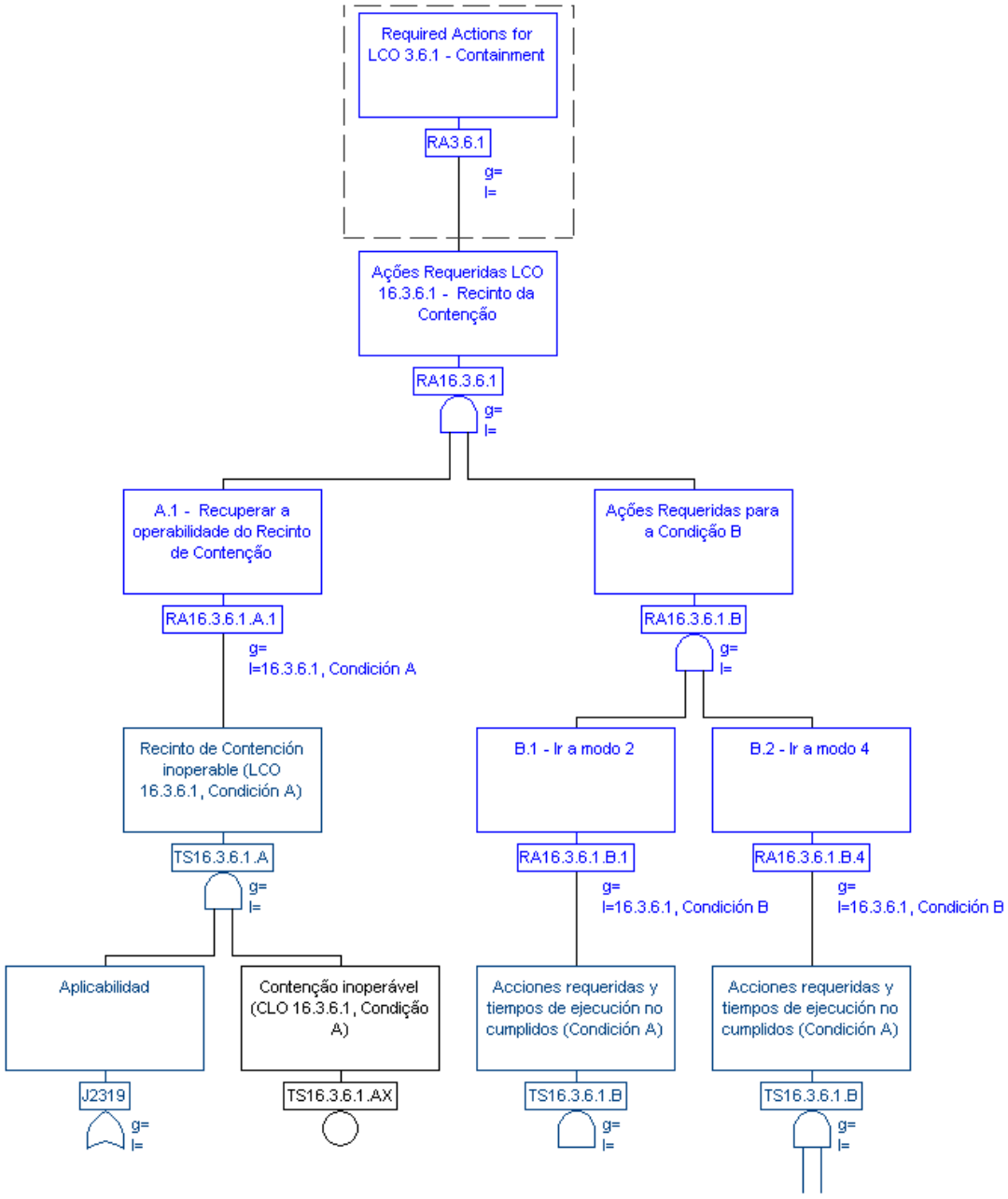
AÇÕES

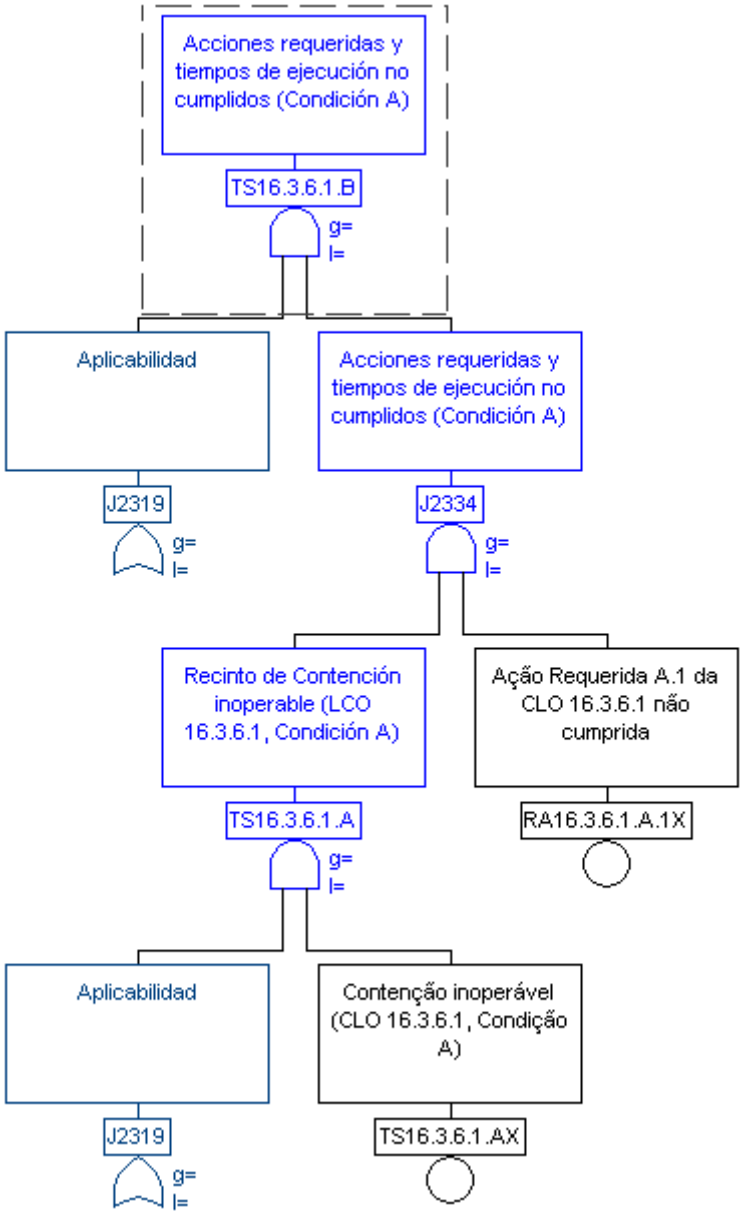
CONDIÇÃO		AÇÃO REQUERIDA		TEMPO PARA CONCLUSÃO
A.	Contenção inoperável.	A.1	Restaure a contenção para a condição OPERÁVEL.	1 hora
B.	A Ação Requerida e o Tempo para Conclusão associados não cumpridos.	B.1	Esteja em MODO 3.	6 horas
		E		
		B.2	Esteja em MODO 5.	36 horas

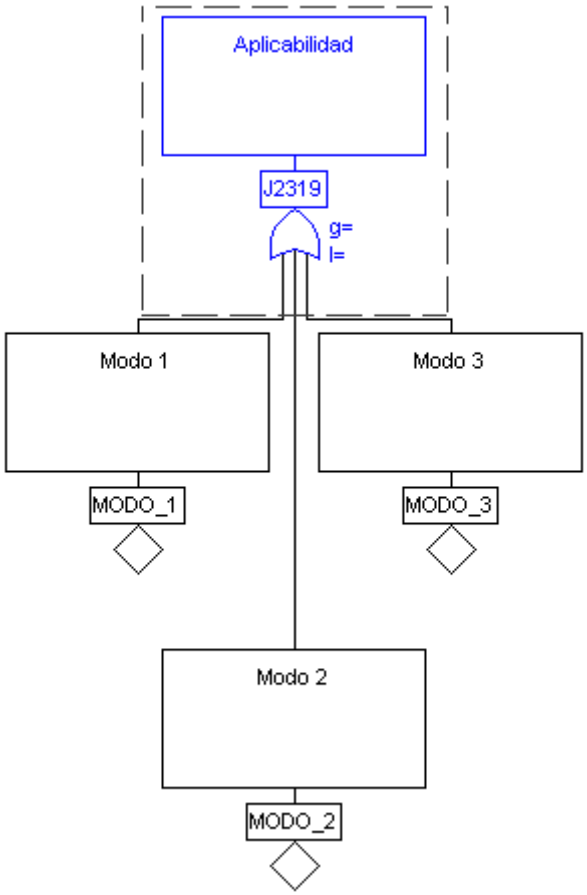
Apêndice III

Árvore de Falha realizada para CLO 16.3.6.1

(Obtido pelo CAFTA Fault Tree Analysis criado pelo Electric Power Research Institute (EPRI))


Required Actions for LCO 3.6.1 - Containment
RA3.6.1
g=
l=
Ações Requeridas LCO 16.3.6.1 - Recinto da Contenção
RA16.3.6.1
g=
l=
A.1 - Recuperar a operabilidade do Recinto de Contenção
Ações Requeridas para a Condição B
RA16.3.6.1.A.1
g=
l=16.3.6.1, Condición A
RA16.3.6.1.B
g=
l=
Recinto de Contención inoperable (LCO 16.3.6.1, Condición A)
B.1 - Ir a modo 2
B.2 - Ir a modo 4
TS16.3.6.1.A
g=
l=
RA16.3.6.1.B.1
g=
l=16.3.6.1, Condición B
RA16.3.6.1.B.4
g=
l=16.3.6.1, Condición B
Aplicabilidad
Contenção inoperável (CLO 16.3.6.1, Condição A)
Acciones requeridas y tiempos de ejecución no cumplidos (Condición A)
Acciones requeridas y tiempos de ejecución no cumplidos (Condición A)
J2319
g=
l=
TS16.3.6.1.AX
TS16.3.6.1.B
g=
l=
TS16.3.6.1.B
g=
l=


Acciones requeridas y tiempos de ejecución no cumplidos (Condición A)
TS16.3.6.1.B
g=
l=
Aplicabilidad
Acciones requeridas y tiempos de ejecución no cumplidos (Condición A)
J2319
g=
l=
J2334
g=
l=
Recinto de Contención inoperable (LCO 16.3.6.1, Condición A)
Ação Requerida A.1 da CLO 16.3.6.1 não cumprida
TS16.3.6.1.A
g=
l=
RA16.3.6.1.A.1X
Aplicabilidad
Contenção inoperável (CLO 16.3.6.1, Condição A)
J2319
g=
l=
TS16.3.6.1.AX


Aplicabilidad
J2319
g=
l=
Modo 1
Modo 3
MODO_1
MODO_3
Modo 2
MODO_2