MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INVESTIGAÇÃO DE CADEIAS DE MARKOV PARA TESTES DE SOFTWARE

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/INPE/CNPq)

Eduardo Ribeiro Moraes (UNIFEI, Bolsista PIBIC/CNPq)
E-mail: eduardo.rmoraes@outlook.com

Nandamudi Lankalapalli Vijaykumar (LAC/CTE/INPE, Orientador)
E-mail: vijay.nl@inpe.br

Julho de 2017

RESUMO

Este trabalho tem como objetivo a continuidade aos projetos de Iniciação Científica para melhorias em Testes de Software os quais cada vez mais são relevantes por garantir a qualidade no sistema desenvolvido. A iniciação científica descrita neste resumo aborda o uso de Cadeias de Markov, que são muito utilizadas para avaliar desempenho de sistemas, para priorizar sequências de teste. Os testes de software podem ser de caixa branca (dependem do código implementado) ou de caixa preta (não necessitam do código implementado). O foco será testes caixa preta. Neste caso, os testes são gerados a partir de modelagem da especificação de software sem ter a necessidade de se ter o código. O que ocorre é que os testes são gerados bem antes da implementação de software e quando o software estiver pronto, os testes gerados a partir da especificação são exercitados na implementação para conferir se a implementação está de conformidade com a especificação. Por este motivo, testes caixa preta também são conhecidos como testes de conformidade. No entanto, dependendo de como os testes são gerados, poderão haver centenas de milhares de casos de testes se a especificação for algo complexo. Então há uma necessidade de priorizar estes casos sem perder a sua qualidade, ou seja, de alguma forma, deve haver uma garantia que o software está validado. Para priorizar os casos de teste a ideia deste trabalho é explorar Cadeias de Markov. A especificação é modelada como uma Cadeia de Markov e a partir da qual se obtêm probabilidades limite que se referem à quantidade de tempo que o estado ficou ativo. Então, as probabilidades limite poderão dar uma visão ao testador sobre quais funções (estados) deverão ser testados com certa prioridade. Os resultados serão testados para especificações geradas aleatoriamente e depois serão testados em aplicações reais, em particular, aplicações espaciais de software embarcado em satélites e/ou em outras missões.

Sumário

Lista de Figuras

1. INTRODUÇÃO

A Qualidade de software garante que tanto o processo de desenvolvimento quanto o produto de software atinjam os níveis de qualidade especificados (Delamaro et al., 2007). Teste de um Software é um dos processos para se garantir a qualidade de um software que examina e valida o produto por meio de sua execução com o objetivo de revelar erros. Esta atividade, que corresponde à validação, aumenta a confiança de que o software desempenhe suas funções especificadas (BURNSTEIN, 2003).

Testes de Software dependem se o conjunto de testes realizado tem qualidade, assim evitando ao máximo falhas graves, falhas de fácil correção, defeitos e erros não revelados (Delamaro et al., 2007). Há algumas limitações em Testes de software que são a não existência de um algoritmo de teste de propósito geral para provar a corretitude de um programa, a indecisividade se dois caminhos de um mesmo programa ou de diferentes programas computam a mesma função e se existe um dado de entrada que leve à execução de um dado caminho de um programa (é indecidível se um dado caminho é executável ou não) (BURNSTEIN,2003). Testes também são custosos, logo procura-se soluções para maximizar a confiança de um software com baixos custos.

Há diferentes tipos de Testes de Software (BURNSTEIN, 2003) como o teste Caixa Branca onde se verifica a estrutura interna do programa e realiza testes dos procedimentos. Outro tipo é o teste Caixa Preta que aborda o software de um ponto de vista macroscópico, identificam-se as funções que o software deve realizar, suas especificações de requisitos, e cria casos de teste capazes de checar se essas funções estão sendo executadas corretamente. Neste tipo de teste existe uma dificuldade em quantificar a atividade de teste, logo não pode garantir se as partes essenciais ou críticas do software foram executadas e da automatização desses testes (BURNSTEIN, 2003). Também existe uma técnica baseada em erros o qual os requisitos de teste são derivados a partir dos erros mais frequentes cometidos durante o processo de desenvolvimento do software. Nesta pesquisa o tipo de teste utilizado foi o teste com a técnica funcional (Caixa Preta).

Todos os tipos de teste utilizam grafos para sua representação. Grafos são relações entre objetos de um determinado conjunto em sua representação um círculo (ou ponto) significa um objeto e linhas representam a relação dos objetos (ENDO T., 2010). Estas linhas podem ou não ter um único sentido. Os grafos se assemelham à Máquina Estados Finitos (MEF) (Lee & Yannakakis, 1996) onde há um número finito de estados e arcos de transição ligando estados entre eles. Ao longo do arco, há um rótulo que significa um evento ou entrada, ou seja, quando este evento é habilitado, há uma transição de um estado para um outro (ENDO T., 2010).

Antes de concluir esta introdução é necessário entender Cadeias de Markov. Estas cadeias são muito utilizadas para avaliar desempenho de um dado sistema. A sua propriedade fundamental é conhecida como sem memória, ou seja, a transição de um estado para um outro não depende do passou (ou em quais estados a Cadeia estava) (NOGUEIRA, 2009). As Cadeias de Markov também se assemelham às MEFs, pois também têm conjunto de estados e transições entre eles. No entanto, os rótulos nos arcos de transição possuem uma informação estocástica, em particular, estes rótulos (eventos) devem seguir uma distribuição exponencial (NOGUEIRA, 2009). As Cadeias de Markov podem ser discretos ou contínuos (NOGUEIRA, 2009). No caso de discretos, os rótulos de transição possuem valores de probabilidades, ou seja, há uma probabilidade determinada para fazer uma transição entre um estado e outro. No caso de contínuos, os rótulos (ou eventos) correspondem às taxas de transição, por exemplo, tempo entre chegada de clientes, tempo de atendimento a cada cliente, etc. Quando a Cadeia é resolvida, obtêm-se as probabilidades limite para cada estado que significam que o tempo que cada estado ficou ativo durante um certo tempo ou até em horizonte infinito (NOGUEIRA, 2009).

Concluindo esta introdução, utilizando a representação da especificação como uma MEF, há métodos (ou critérios) para percorrer uma MEF e gerar as sequências, ou seja, partindo de um estado inicial segue os arcos de transições onde há entradas/eventos e volta para o mesmo estado inicial (NOGUEIRA, 2009). Ao percorrer, esta sequência de entradas é listada. Dependendo do critério de teste, pode ser que tenha uma grande quantidade de sequências a ser exercitada na implementação. E às vezes não há tempo hábil para exercitar todas as sequências. Neste sentido, é necessário achar estratégias para dar prioridade a estas sequências, diminuir esse número de sequências, a serem exercitadas e validar se a implementação está conforme à sua especificação. A estratégia proposta neste trabalho de Iniciação Científica é considerar o modelo como uma Cadeia de Markov de Tempo Discreto e resolver esta cadeia. Uma vez resolvida a cadeia, e com as probabilidades limite pode-se ordenar as sequências. Em outras palavras, as sequências serão organizadas ou classificadas baseado em quais estados a Cadeia ficou mais tempo.

2. FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é apresentar conceitos breves sobre Teste de Software, Cadeias de Markov e Testes Baseado em Modelo. Ele está dividido nas seguintes seções com conteúdo do seu respectivo título: 3.1 Testes de Software; 3.2 Cadeias de Markov; 3.3 Teste Baseado em Modelo; 3.4 Máquina de Estado Finito; 3.5 Critério Switch Cover; 3.6 Considerações Finais.

2.1 TESTES DE SOFTWARE

Testes de Software têm como objetivo revelar a presença de erros (Myers et al, 2011), impactando na qualidade do software que depende se o conjunto de testes realizado são de qualidade assim evitando ao máximo falhas graves, falhas de fácil correção, defeitos e erros não revelados. Testes também são custosos, logo procura-se técnicas para maximizar a confiança de um software com baixos custos.

Algumas das técnicas para minimizar as várias falhas que podem ocorrer são o teste Caixa Branca o qual verifica a estrutura interna do programa, seu código, e realiza testes dos procedimentos. Já o teste Caixa Preta que foi utilizado nesse trabalho, é uma técnica que imagina o software como uma “caixa preta”, e utilizam-se apenas as saídas e compara com as que eram esperadas na especificação do software, por isso o teste também é conhecido como Teste de Entrada e Saída.

2.2 CADEIAS DE MARKOV

Processo Markoviano é um processo estocástico definido como o estado futuro depende apenas do estado presente e não dos estados passados, denominado como *memoryless property* (propriedade sem memória). Cadeias de Markov são um processo Markoviano com conjunto de estados discretos e suas transições são baseadas nas probabilidades de mudança de estado. Um exemplo é ilustrado na Figura 2.1 de um processo com quatro estados e suas transições de saída. Note que a soma das probabilidades de saída é igual a um.

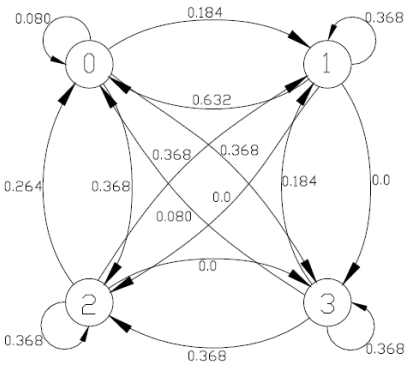

Figura 2.1: Exemplo de uma Cadeia de Markov
Fonte: Nogueira (2009)

2.3 TESTE BASEADO EM MODELO

O Teste Baseado em Modelo (TBM) é a criação de um modelo para representar o funcionamento de um software, modelo este que é desenvolvido a partir da documentação e especificação do sistema (Dalal, S.R., et al; 1999). Este modelo é então usado por alguns métodos de geração dos casos de testes, como ilustrado na Figura 2.2. Dessa forma, o código se torna dispensável na criação dos casos de teste, pois é a partir da documentação que será criado um modelo.

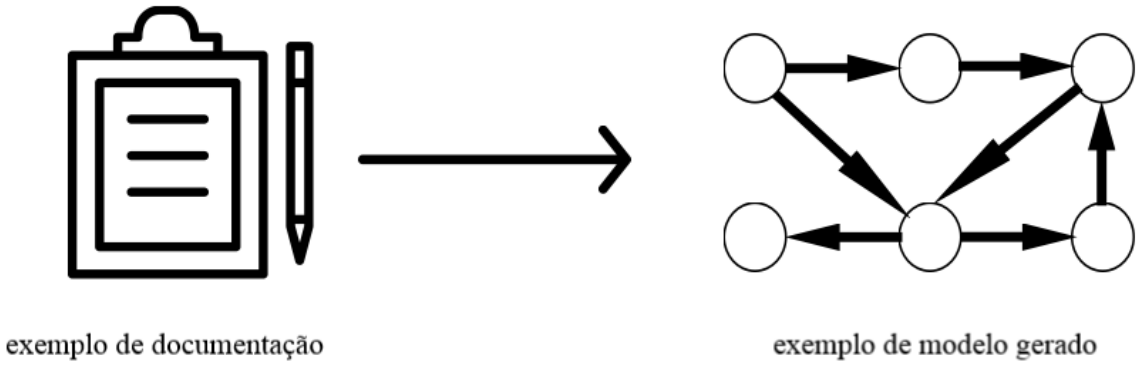

Figura 2.2: Teste baseado em Modelo a partir da Documentação

Como a abordagem TBM depende de um modelo, as Máquinas de Estados Finitos (MEF) (Lee & Yannakakis, 1996) se tornam candidatas para representar o comportamento de um software. Uma descrição resumida de uma MEF será apresentada na Seção 2.4.

2.4 MÁQUINA DE ESTADO FINITO

As Máquinas de Estado Finito (MEF) (Lee; Yannakakis, 1996) são um conjunto de estados e arcos de transição entre estes estados. Os arcos de transição possuem rótulos que são conhecidos como eventos ou entradas. A mudança de um estado para um outro ocorre quando este evento ou entrada é estimulada. Para este trabalho, o modelo de Diagrama de Transição de Estados é o que será utilizado para visualizar uma MEF. Os círculos são os estados e as linhas entre eles as transições, e as setas a indicação do destino destas transições de um estado para o outro. Neste trabalho as transições contêm o valor de uma probabilidade para mudar-se para o estado destino.

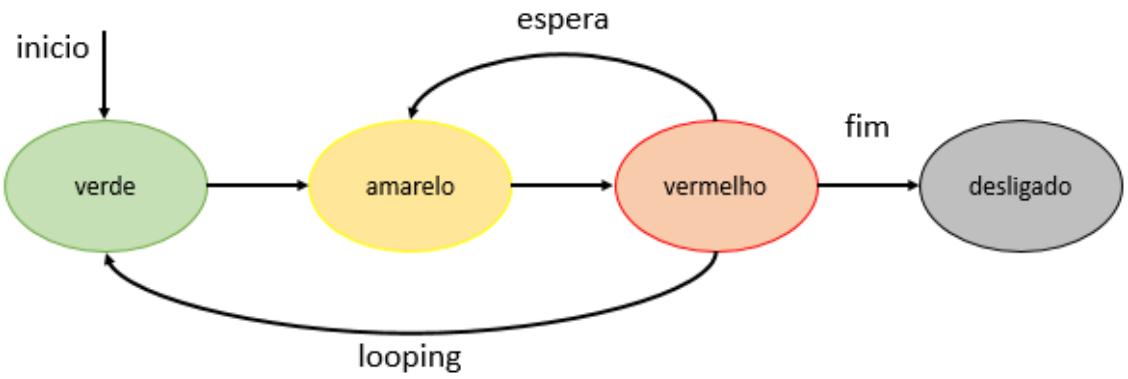

Figura 2.3: Representação de um sistema de semáforo a partir de uma MEF

Um dos exemplos mais clássicos de uma MEF é a de um sistema de semáforo. Neste exemplo mostrado na Figura 2.3, foram especificados 4 estados: o início do processo; a troca de cores, que irá disparar a mudança entre vermelho, amarelo e verde; o espera, que irá realizar um processo de espera no sistema; e o fim, quando é desligado. Esses processos possuem 5 transições, onde inicialmente o semáforo inicia o processo, realiza a troca de estado três vezes, com pausas neste meio tempo retornando à mudança de estado, e depois indo ao início novamente para reiniciar o processo. Isso será realizado em looping até que por algum motivo o sistema precise ser desligado, e quando isto ocorrer irá para o estado fim. A MEF, como tal, especifica apenas o caminho; como será realizado e a quantidade de vezes a ser passado entre as transições é a partir da interpretação do sistema.

Apesar de as Máquinas de Estado Finito trazerem uma informação completa ao testador, onde todas as possibilidades estão descritas através de seu diagrama, é difícil poder coletar esta informação sem se basear em algo que investigue os estados e transições. Então, mesmo que tenha em mãos o sistema, é inviável utilizar todas as entradas possíveis de uma máquina. Com isso, MEFs possuem diversos critérios que

foram investigados para auxiliar o testador a explorar a MEF, e é possível encontrar na literatura diversos trabalhos que demonstram critérios para percorrer uma MEF. Alguns exemplos de métodos são critérios Transition Tour, Switch Cover, DS e UIO que estão implementados na ferramenta WEB-PerformCharts (Alessandro et al, 2014). No subcapítulo 2.5 será detalhado o critério Switch Cover, que é o utilizado por este trabalho (Sidhu, D.P. & Leung, T., 1989).

2.5 CRITÉRIO SWITCH COVER

O critério Switch Cover, também chamado de "todas as combinações" (Pimont & Rault, 1976), é um dos mais complexos na literatura, pois especifica que todos os pares de transições adjacentes devem ser executadas pelo menos uma vez. Para implementá-lo, o primeiro passo é transformar todas as transições de uma MEF em vértices de um grafo, pois assim torna o algoritmo menos complexo de analisar e tem uma maior amplitude em todos os pares de transições (Pimont & Rault; 1976), criando assim o grafo dual. Para exemplificar, a Figura 2.4 representa uma MEF simples, e a Figura 2.5, representa os vértices criados a partir das transições deste primeiro grafo.

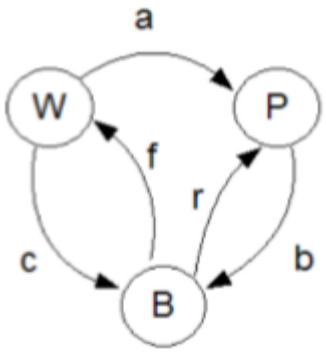

Figura 2.4: Representação de uma MEF em um modelo
Fonte: Amaral (2005)

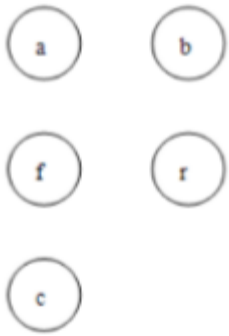

Figura 2.5: Grafo dual a partir da MEF original
Fonte: Amaral (2005)

Depois é necessário criar as arestas neste grafo dual. Para isto, é analisado o comportamento do grafo original, e verifica-se o caminho percorrido entre estes estados. Na Figura 2.6 mostra o grafo original onde destaca um caminho a partir do conjunto de um par de transição, enquanto que a Figura 2.7 exibe o grafo dual já preenchida de arestas, mas, em destaque, a aresta que foi resultado da transição destacada do grafo original.

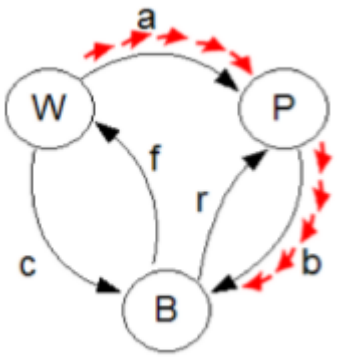

Figura 2.6: Nova transição no Grafo dual pela MEF original
Fonte: Amaral (2005)

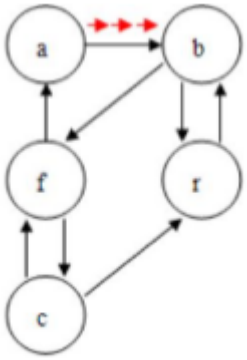

Figura 2.7 Criando uma nova transição
Fonte: Amaral (2005)

Agora deve-se balanceá-lo, analisando as polaridades dos vértices. Balanceamento consiste em duplicar as arestas de tal forma que o número de arestas que chegam deve ser igual ao número de arestas que deixam o vértice. Na prática, é fazer com que cada vértice possua o mesmo número de entradas e saídas, o que no final permite ter um caminho euleriano, onde cada aresta no grafo é visitada apenas uma vez. Um exemplo desta mudança está na Figura 2.8 no vértice r, onde possui duas arestas de entrada, e uma de saída.

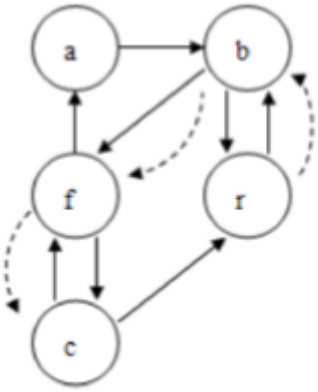

Figura 2.8: Grafo dual balanceado Fonte: Amaral (2005)

Por fim, basta apenas gerar as sequências de teste percorrendo o grafo dual, iniciando de algum vértice inicial e voltando para ele próprio. Como o estado W na MEF original é o estado inicial, então tanto o vértice a quanto c do grafo dual serão iniciais. As sequências geradas para este grafo são: abf, abrbf, cf & crbf.

2.6 CONSIDERAÇÕES FINAIS

Neste trabalho utilizou-se do critério do *Switch Cover* por sua complexidade e questão de integração com a ferramenta de outros trabalhos. Propomos realizar um estudo sobre a utilização de Cadeias de Markov na priorização de testes nesta ferramenta.

3. USO DE CADEIAS DE MARKOV NA PRIORIZAÇÃO DE TESTES

Testes são realizados com objetivo de diminuir os erros assim podemos dizer também que reduzem os riscos, portanto procura-se por alternativas pragmáticas, acessíveis, rápidas e que forneçam resultados. Dependendo de como os testes são gerados, podem haver centenas de milhares de casos de testes se a especificação for algo complexo. Então há uma necessidade de priorizar estes casos sem perder a sua qualidade, neste sentido, seria interessante achar alguma estratégia para diminuir este número de sequências tendo uma garantia que o software está validado. A estratégia proposta neste trabalho de Iniciação Científica é considerar o modelo como uma Cadeia de Markov de Tempo Discreto e resolver esta cadeia. Uma vez resolvida a cadeia, e com as probabilidades limite pode-se ordenar as sequências. Em outras palavras, as sequências serão organizadas ou classificadas baseado em quais estados a Cadeia ficou mais tempo. Ou seja, as sequências que passem por estados que ficaram mais tempo ativos terão prioridade de serem exercitadas.

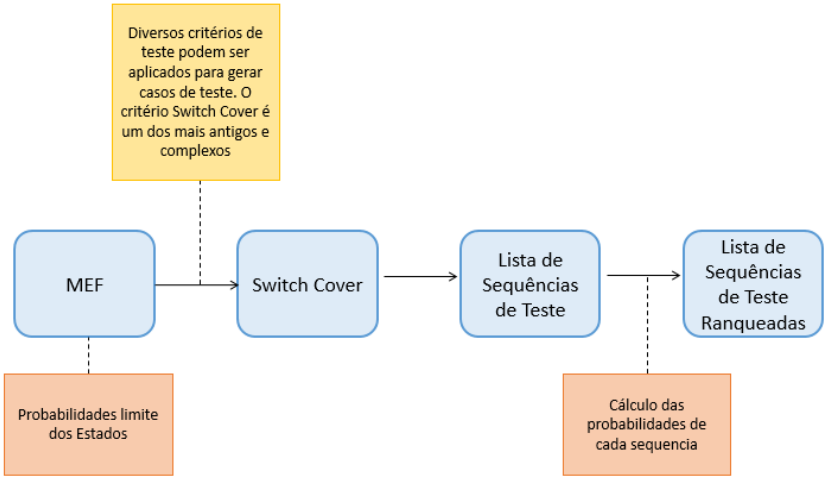

Figura 3.1: Visão Geral do Projeto

Desta forma, a proposta deste trabalho começa na especificação onde se modela uma Cadeia de Markov e obtém as probabilidades limite. Em seguida o *Switch Cover* gera uma lista de Sequências de Teste da qual é realizado o cálculo da probabilidade de cada sequência e por fim são ranqueadas as sequencias a partir do cálculo. A Figura 3.1 representa de forma simplificada a visão geral do projeto, enfatizando em passos as áreas mais relevantes.

De forma geral, na Figura 3.1, primeiramente o sistema identifica uma MEF em um diretório especificado, está já modelada como uma Cadeia de Markov e com as

probabilidades limite dos estados, isto é, a quantidade de tempo que o estado fica ativo. Assume-se que a Cadeia de Markov já foi calculada previamente gerando as probabilidades limite que serão utilizadas por esta abordagem para ranquear as sequências. A MEF deve estar representada no formato XML. Com a MEF inserida em memória (s*tep 1*), será aplicado o critério Switch Cover e criará o grafo dual (*step 2*). Com isso, serão geradas sequências de caso de teste, a partir dessas sequencias é realizado o cálculo de cada uma das sequências de caso de teste, o cálculo é a multiplicação da probabilidade limite dos estados de cada estado desta sequência (*step 3*). O último passo é ordenação dessas sequências do menor para o maior valor em seguida é salvo as sequencias ranqueadas em um arquivo TXT (*step 4*). Por fim é feito uma análise desses resultados.

4. RESULTADOS

Inicialmente foi feito um estudo prévio com uma MEF com 17 estados que gera 32 sequências de casos de teste para conferir se o algoritmo estava sendo executado corretamente, assim foi realizado os cálculos manualmente e comparados com o resultado do algoritmo resultando em nenhuma divergência.

Para as MEFs reais, foi utilizado neste trabalho o modelo do software do projeto Software of the Payload Data Handling Computer (SWPDC). O SWPDC é um software que gerencia a aquisição de dados científicos, de diagnóstico, a partir de câmeras de raio X (Event Pre-Processor (EPP) H1 e EPP H2) e um Sistema de Computação denominado Central Electronics Unit (CEU). O software SWPDC é composto de 20 (vinte) modelos para teste, ou seja, foram utilizados 20 (vinte) cenários representados por MEFs para a geração dos casos de teste (Santiago et al., 2010).

Observando o modelo número 9, após todos passos explicados no tópico anterior gerou-se as três seguintes sequências de caso de teste, a primeira com 8 estados, a segunda com 5 estados e a terceira com 12 estados, os nomes dos estados é a junção do nome de 2 transições e estão separados por uma vírgula:

switchPDCOn/start60s,POSTOk/null,endtime/null,VER_OP_MODE/INFO_OP_MODE,ACT_HW_EPP1On/CMD_REC,ACT_HW_EPP2On/CMD_REC,start30s/null,endtime/null

switchPDCOn/start60s,POSTOk/null,endtime/null,CH_OP_MODE_Nominal/CMD_REC,VER_OP_MODE/INFO_OP_MODE

switchPDCOn/start60s,POSTOk/null,endtime/null,VER_OP_MODE/INFO_OP_MODE,P_DMP_Prg_0000H_7FFFH/CMD_REC,TX_DATA_Dmp_29_1/DMP_DATA_RSC29_1_1111,TX_DATA_Dmp_0/DMP_DATA_RSC0_549,TX_DATA_Sci/NO_DATA,ACT_HW_EPP2Off/CMD_REC,ACT_HW_EPP1Off/CMD_REC,switchPDCOff/null,switchPDCOn/start60s

Com estas sequências e tendo os valores das probabilidades dos estados o algoritmo realiza o cálculo que é a multiplicação da probabilidade de cada estado. O cálculo da segunda sequência pode ser visto na Figura 4.1.

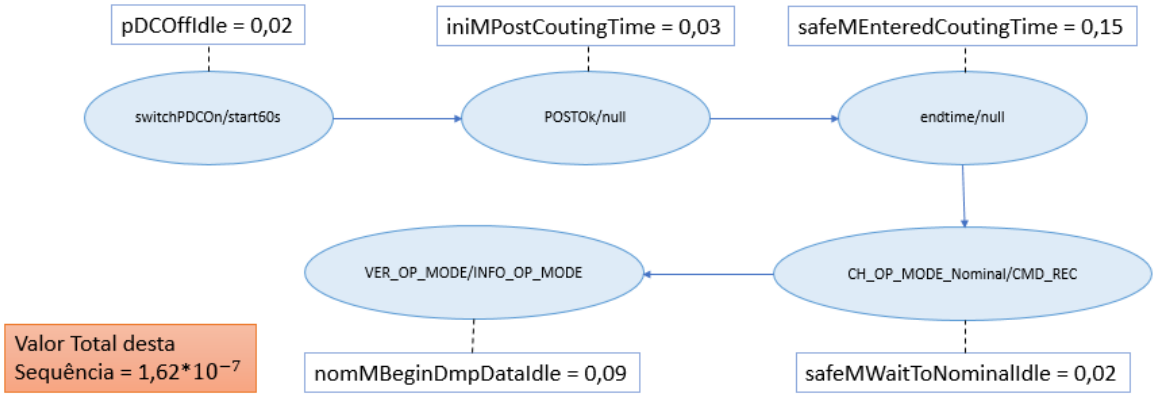

Figura 4.1: Demonstração do cálculo da segunda sequência

Este cálculo é realizado para todas sequências, neste caso a primeira sequência o valor total é 7,20 * 10^ (-10), e a terceira é 2,62* 10^ (-19). A partir desses valores conseguimos ranquear as sequências mais importantes do menor para o maior seguindo nosso método, o algoritmo realiza esse ranqueamento e gera um arquivo ordenado das sequencias apresentadas.

Analisando nosso método quanto mais estados a sequência de teste tiver e com estados com probabilidades parecidas maior será sua importância. E quanto menos estados a sequência de teste tiver e com estados com probabilidades parecidas menor será sua importância. Porem se tivermos sequências com o mesmo número de estados e com probabilidades de estados parecidas, o valor dos estados que irão influencias mais, sendo que quanto maior forem os números menos ranqueados ficarão, é inversamente proporcional. A criação de outros métodos para calcular as sequências e sua ordenação fica a critério e sua efetividade deve ser comparada.

5. CONCLUSÃO E TRABALHOS FUTUROS

Testes de software é fundamental no processo de desenvolvimento de um software e está diretamente relacionado à qualidade do produto, o custo por trás dos testes podem influenciar na viabilidade de projetos. Existe uma demanda no sentido de melhorar o desenvolvimento e aplicação de técnicas de teste proporcionando o desenvolvimento das ferramentas de testes.

Após o avanço na programação e resultados obtidos com a ferramenta de testes desenvolvida vistos no tópico anterior, Resultados Preliminares, a criação de um método ideal que determine o nível de importâncias de sequências de casos de teste deve ser útil para a priorização dos testes.

REFERÊNCIAS BIBLIOGRÁFICAS

Amaral, A. S. M. S. Geração de casos de testes para sistemas especificados em Statecharts. 162 p. Dissertação (Mestrado) — Instituto Nacional de Pesquisas Espaciais, (INPE-14215TDI/1116). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005. Disponível em: <http://urlib.net/rep/sid.inpe.br/MTC-m13@80/2006/02.14.19.24?languagebutton=pt-BR>.

BURNSTEIN, ILENE. Practical Software Testing.Chicago: Ed. Springer 2003

Dalal, S.R., et al. "Model-based testing in practice." Proceedings of the 21st international conference on Software engineering. ACM, 1999

DELAMARO, E.; MANDONADO, J. C.; JINO, M. Introdução ao Teste de Software. Rio de Janeiro: Ed. Elsevier 2007.

ENDO T., ANDRÉ. Teste baseado em máquinas de estados finitos para aplicações orientadas a serviço. São Paulo: Serviço de Pós-Graduação do ICMC-USP, 2010.

Lee, D, & Yannakakis, M. Principles and methods of testing finite state machines: a survey. Proceedings of the IEEE 84(8):1090–1123, (1996).

Myers, G.J.; Sandler, C., and Badgett, T. The art of software testing. John Wiley & Sons, 2011.

NOGUEIRA, FERNANDO. Notas de Aula de Cadeias de Markov, Universidade Federal de Juiz de Fora, MG: 2009.

Pimont, S., & RAULT, C. J. A Software Raliability Assessment Based on a Structural and Bheaviiral Analysis of Programs. THOMSON - CSF, 1976.

RAMOS, RICARDO A. Introdução ao Teste de software.

Santiago, V.A.; Cristini, A.M.; Vijaykumar, N.L. Model-based test case generation using Statecharts and Z: A comparison and a combined approach. São José dos Campos, 2010. 72 p. (INPE-16677-RPQ/850). Disponível em: <http://urlib.net/sid.inpe.br/mtcm19@80/2010/02.26.14.05>.

Sidhu, D. P., & Leung, T. (1989). Formal Methods for Protocol Testing: A Detailed Study. Transactions on Software Engineering. IEEE Computer Society. Dept. of Comput. Sci., Maryland Univ., Baltimore, MD. 13(4), 413-426.

SOUZA F., ÉRICA. GERAÇÃO DE CASOS DE TESTE PARA SISTEMAS DA ÁREA ESPECIAL USANDO CRITÉRIOS DE TESTE PARA MÁQUINAS DE ESTADOS FINITOS. INPE-16682-TDI/1627, 2010.