Universidade Federal de Santa Catarina - UFSC
Centro Tecnológico
Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Model Checking no NuSMVS

Érique Moser

Disciplina: DAS410099 - Verificação Formal
Orientação: Prof. Dr. Max Hering de Queiroz

Florianópolis
2022

Lista de Figuras

Sumário

1 Introdução

As ferramentas de verificação formal contém muitos tópicos de estudos que estão constantemente recebendo o ingresso de novos pesquisadores. Métodos de verificação formal surge no contexto de buscar novas técnicas e ferramentas para auxiliar no desenvolvimento de sistemas mais robustos e seguros. A ideia é construir representações matemáticas a partir do modelo e a partir os requisitos do sistema (especificações), de forma a provar matematicamente que o modelo atende as especificações. Os algoritmos CTL e PLTL são as principais lógicas de representar as linguagens verificadoras, cada uma com seus algoritmos correspondentes e desenvolvidos de forma independente. O NuSMV é uma ferramenta de software para a verificação formal de sistemas de estados finitos.

Este trabalho tem o objetivo de apresentar o verificador de modelos SMV e sua evolução NuSMV, através de exemplos e fundamentações acerca do seu funcionamento. A ideia é apresentar a ferramenta com detalhes acerca do formalismo da linguagem e da execução dos códigos.

Este documento está estruturado na seguinte formal:

No capítulo 2 são apresentadas as características do NuSMV, estabelecendo seus aspectos gerais, origem e estrutura de código.

No capítulo 3 é apresentado um tutorial de instalação e execução do programa.

No capítulo 4 é apresentado um exemplo teórico de um elevador extraído do livro de referência.

No capítulo 5 é apresentado um problema prático extraído da pesquisa na área de petróleo e gás.

No capítulo 6 é apresentada uma conclusão com os principais aprendizados.

2 Características do NuSMV

2.1 Aspectos gerais

A linguagem de verificação formal SMV foi desenvolvido por K. L. McMillan, sob orientação de E. M. Clarke na universidade Carnegie-Mellon University (Pittsburg, pA, USA) em meados de 1998 [1]. Essa linguagem realiza a verificação do modelo simbólico (baseado em BDD) de fórmulas CTL em redes de autômatos com variáveis compartilhadas [2]. A ferramenta é disponibilizada de forma livre no site da Universidade. Entretanto, após a falta de atualizações da ferramenta, um novo grupo de estudos começou a trabalhar em uma versão aprimorada do SMV, que é o NuSMV, e que terá um maior enfoque neste trabalho. Com relação ao SMV, o NuSMV fornece os seguintes recursos adicionais: interação, análise de invariantes, métodos de particionamento, verificação do modelo LTL, verificação do modelo PSL, verificação de modelo limitado baseado em SAT.

A linguagem de entrada do NuSMV é projetada para permitir a descrição de sistemas de estados finitos que variam de completamente síncronos a completamente assíncronos. A linguagem NuSMV (como a linguagem do SMV) fornece descrições hierárquicas modulares e a definição de componentes reutilizáveis. O propósito básico da linguagem NuSMV é descrever (usando expressões em cálculo proposicional) a relação de transição de uma estrutura finita de Kripke. Isso proporciona uma grande flexibilidade, mas ao mesmo tempo pode introduzir o perigo de inconsistência (para usuários não especialistas) [3] . Alguns exemplos resolvidos de códigos podem ser encontrados em [4].

Devido à sua rapidez e fluidez de verificação, algumas características dessa linguagem, a tornam uma ferramenta muito utilizada para verificação de Model Checking. A seguir serão listadas algumas dessas características:

- Ela utiliza composições hierárquicas e permite composição de autômatos, inclusive a utilização de variáveis compartilhadas.
- Os autômatos são descritos textualmente, como será visto adiante.
- Uma característica importante do SMV são as suposições de justiça que podem ser incluídas, permitindo a elaboração de sistemas concorrentes que realizam execução paralela, ou até mesmo limitar a verificação para atender alguma condição.
- Como todas outras ferramentas de verificação estudadas, o NuSMV fornece um contraexemplo quando a propriedade solicitada não é satisfeita. Caso contrário, é retornado como 'TRUE' quando é satisfeita.
- O SMV tem uma semântica muito declarativa e entende qualquer fórmula booleana ligando os valores atuais e futuros das variáveis.

2.1.1 Vantagens e Desvantagens

Como pontos positivos, o NuSMV é uma das ferramentas com maior probabilidade de verificar completamente um sistema complexo. Além disso, ele oferece muitas opções permitindo que seu usuário experimente várias estratégias diferentes para ordenar variáveis booleanas, para a representação do gráfico de estado, etc.

Alguns pontos negativos apontam uma falta de facilidade para simulação, já que o NuSMV não oferece uma interface gráfica para verificação dos modelos, e apenas é executado em linhas de comando. Além disso, existe alguma critica em relação a linguagem escolhida para descrever os autômatos, entretanto, essa característica varia da experiência de cada pessoa.

2.2 Estrutura de código

O programa possui uma estrutura a fim de facilitar a organização e o seu entendimento. Alguns dos elementos que fazem parte dessa estrutura são: Módulos, Variável local, Atribuições, DEFINE, FAIRNESS, SPEC. O manual completo do programa com os recursos disponíveis pode ser consultado em [5]. A seguir será analisado cada um desses elementos para entender como que eles auxiliam a maneira como o programa executa o código.

Módulos: Um programa SMV pode consistir em vários módulos. Cada módulo é uma descrição encapsulada que pode ser instanciada várias vezes dentro do modelo. Um módulo também pode conter instâncias de outros módulos, permitindo que uma hierarquia estrutural seja construída. Guardadas as devidas proporções, essa característica lembra 'função' em comparação à outra linguagem de programação, como C, onde permite o recebimento de parâmetros e pode realizar alguma ação repetidas vezes. Por fim, cada descrição NuSMV deve conter um módulo chamado main, sem parâmetros funcionais, já que esse módulo forma a raiz da hierarquia do modelo e o ponto de partida da execução.

Variável local: No código é abreviada por VAR, lista todos os estados (na forma de conjuntos) que podem ser acessados pelo autômato. Como exemplo, uma variável pode ser declarada na forma booleana, tipo de enumeração ou subintervalo inteiro.

Atribuições: abreviado por ASSIGN, as atribuições tem sua concepção orientada para descrever uma relação de possível "próximo estado". Portanto, ela tem a função de informar para qual estado o autômato pode evoluir através da sintaxe 'init' e 'next', que representam o estado inicial e o próximo estado. A notação 'next' se aplica para variáveis e não aos estados completos, o que permite decompor alterações em várias partes.

DEFINE: nem sempre é utilizado, sua função é de atribuir o valor de uma expressão à um símbolo. Poderia ser feito utilizando VAR e ASSING, mas símbolos dessa forma não aumentam o espaço de estado. Além disso, para DEFINE, o tipo de variável não precisa ser declarada. Um ponto negativo, por outro lado, é que símbolos não aceitam valor não determinístico.

FAIRNESSS: são suposições de justiça como foi comentado anteriormente. Implica algum comportamento que o sistema deve ter.

SPEC: é a parte do código utilizada para especificar as fórmulas (em nosso caso, de sintaxe CTL). O verificador se baseia nas fórmulas especificadas para realizar a verificação e retornar o resultado da busca.

3 Instalação e execução do programa verificador

3.1 Instalação

O programa de verificação do NuSMV é fácil e rápido de ser instalado, mas merece um pouco de atenção. Ele é leve e ocupa em disco aproximadamente 30MB. Para instalar o programa, deve ser seguido os seguintes passos:

1. Primeiramente deve ser feito o download da sua versão mais recente em: `https://nusmv.fbk.eu/NuSMV/download/getting-v2.html`.
2. Em seguida, deve-se extrair o arquivo *.tar.gz* e colar a pasta em `C:Files`.
3. Por fim deve-se colocar a pasta `C:Files-2.6.0-win64` em **Path** nas variáveis de ambiente do sistema operacional.

3.2 Execução

Os códigos podem ser escritos em algum bloco de notas e salvos com a extensão *.smv*. Além disso eles devem ser salvos na pasta */bin* mencionada anteriormente. Para execução, o prompt deve ser aberto nessa mesma pasta que foi colocado o código *.smv*. Por fim, um arquivo teste *test.smv* pode ser aberto utilizando o comando: **NuSMV test.smv**, como mostra a Figura 1. Uma vez feito isso, o model Checker retornará com o resultado e uma nova verificação estará pronta para ser realizada.

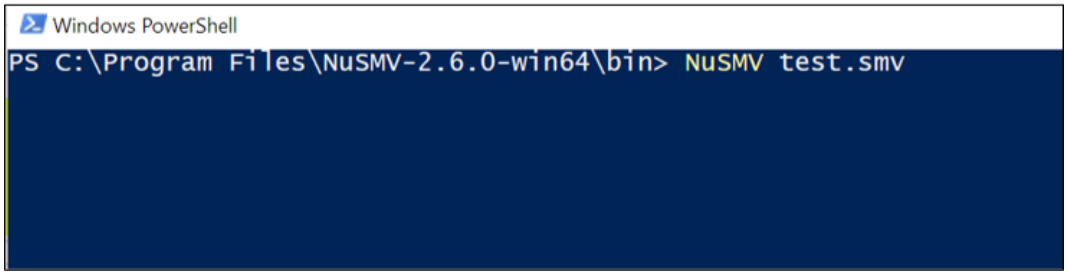

Figura 1: Abertura do programa através do PowerShell do Windows.

4 Exemplo do elevador

Para entender um pouco melhor o funcionamento do NuSMV, é apresentado um exemplo extraído do Capítulo 12 do livro Systems and Software Verification [2] que compreende à verificação de um modelo de autômato de um elevador de 4 andares, Figura 2. O elevador pode ser requisitado em cada andar, mas para simplificar, a cabine sobre e desce de acordo com a sequência 0, 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, etc. Em seguida serão explicadas as partes do código, que pode ser encontrado por completo no Apêndice A.

Figura 2: Problema do Elevador [6].

4.1 Declaração das variáveis

Inicia-se o código explicitando quais são as variáveis que fazem parte do modelo. Para este modelo existem 3 variáveis, cada uma com seus conjuntos de estados:

- Cabine: que representam os andares de 0 a 3;
- Direção: que representa os estados de sobe e desce;
- Pedidos: que indica se há pedido pendente no piso, através de um vetor de booleanos de posição 0 a 3.

A sintaxe para a declaração das variáveis pode ser vista na Figura 3.

```
MODULE main
VAR
    cabin : 0 .. 3;
    dir :  {up,down};
    request: array 0 .. 3 of boolean;
```

Figura 3: Variáveis do problema.

4.2 Relações de transição

As relações de transição corresponde as transições dos autômatos da cabine de da direção. Para o primeiro, é necessário indicar em quais momentos a cabine pode subir, descer ou ficar parada. Já para o segundo, a direção indica quando a cabine deve alterar o sentido da direção. Vale ressaltar que nessa parte do código, o NuSMV permite que a definição da direção ocorra antes da definição do próximo valor da cabine. A construção "case ... esac"é uma seleção de casos em que o autômato evolui conforme explícito na direita, sob determinadas condições da esquerda. As relações de transição da cabine e da direção podem ser vistas na Figura 4.

```
ASSIGN
    next(cabin):=   case
                        dir=up & cabin < 3 : cabin + 1;     --moves up
                        dir=down & cabin > 0 : cabin - 1;   --moves down
                        TRUE : cabin;                           --stuck
                    esac;

                            --sempre que chega em uma das extremidades, muda direção
    next(dir) :=    case
                        dir=up & next(cabin) = 3 : down;    --switch dir
                        dir=down & next(cabin) = 0 : up;    -- switch dir
                        TRUE: dir;                              --keep on
                    esac;
```

Figura 4: Transições da cabine e direção.

As relações de transição da evolução dos pedidos segue a mesma construção. Ela indica qual momento um pedido pode aparecer e desaparecer. O modelo permite que a solicitação ocorra a qualquer momento, exceto se a cabine estiver no andar. As relações de transição da evolução dos pedidos pode ser vista na Figura 5.

```
next(request[0]) := case
                        next(cabin) = 0 : FALSE;    --disappears
                        request[0] : TRUE;          --remains
                        TRUE : {FALSE,TRUE};
                    esac;
next(request[1]) := case
                        next(cabin) = 1 : FALSE;    --disappears
                        request[1] : TRUE;          --remains
                        TRUE : {FALSE,TRUE};
                    esac;
next(request[2]) := case
                        next(cabin) = 2 : FALSE;    --disappears
                        request[2] : TRUE;          --remains
                        TRUE : {FALSE,TRUE};
                    esac;
next(request[3]) := case
                        next(cabin) = 3 : FALSE;    --disappears
                        request[3] : TRUE;          --remains
                        TRUE : {FALSE,TRUE};
                    esac;
```

Figura 5: Transições dos pedidos.

4.3 Estados Iniciais

Os estados iniciais indicam a posição inicial da cabine, da direção e dos pedidos e pode ser visualizadas em código, na Figura 6.

```
init(cabin)          := 0;
init(dir)            := up;
init(request[0])     := FALSE;
init(request[1])     := FALSE;
init(request[2])     := FALSE;
init(request[3])     := FALSE;
```

Figura 6: Estados iniciais do elevador.

4.4 Especificação

A especificação a ser verificada escrita em lógica CTL, para esse exemplo, verifica 3 situações: i) a existência de deadlock; ii) se todos os pedidos são eventualmente satisfeitos; e iii) se os pedidos são eventualmente todos satisfeitos (simultaneamente). A sintaxe para essas situações respectivamente, pode ser vista no código mostrado na Figura 7.

```
SPEC
    AG EX TRUE
SPEC
    AG (AF!request[0] & AF!request[1] & AF!request[2] & AF!request[3])
SPEC
    AG AF (!request[0] & !request[1] & !request[2] & !request[3])
```

Figura 7: Especificações da verificação do problema do elevador.

4.5 Resultados da Verificação

O resultado da verificação do problema do elevador para as especificações anteriormente descritas, pode ser visualizado na Figura 8. Para a primeira verificação, o programa retorna que a ausência de deadlock é verdadeira. Para a segunda especificação o programa retorna que é verdadeiro que todos os pedidos são eventualmente satisfeitos, entretanto para a terceira, é falso que os pedidos são eventualmente dos satisfeitos (simultaneamente). Para a terceira verificação existe um contraexemplo que mostra uma situação a qual há um pedido no primeiro andar e em seguida o elevador é solicitado no andar 0. Porém quando o elevador chega no primeiro andar é visto que o modelo só cumpriu um dos pedidos.

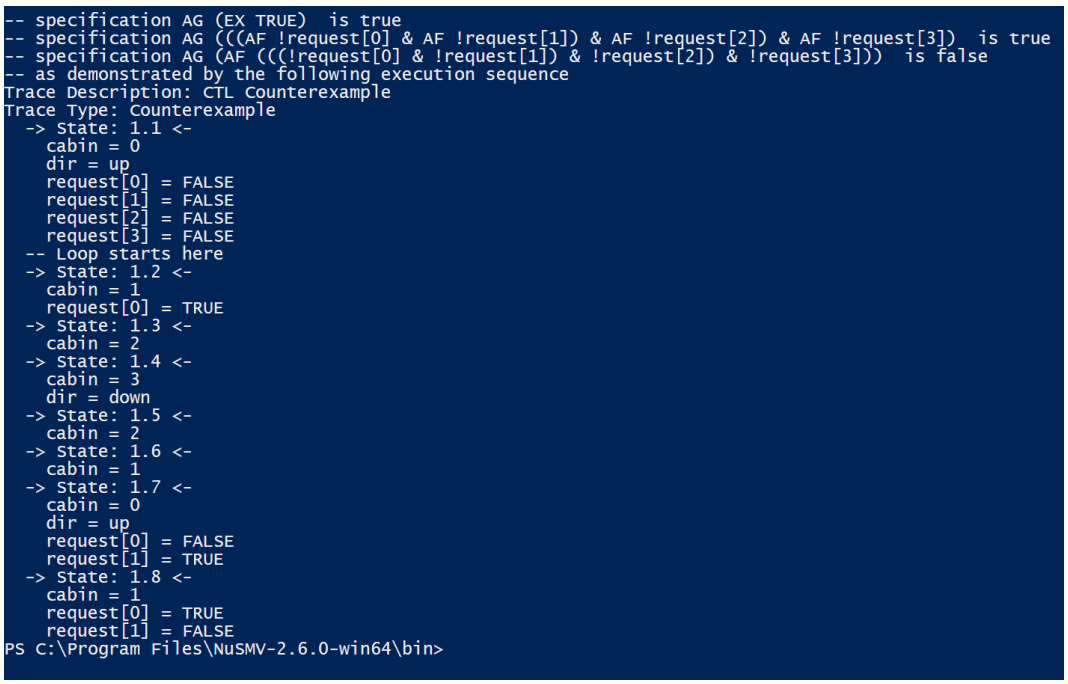

Figura 8: Resultado da verificação do Model Checking do problema do elevador.

5 Problema do Escoamento Multifásico

Para elucidar a versatilidade da ferramenta, é proposto uma aplicação do NuSMV em uma aplicação real encontrada na Indústria de Petróleo e Gás. O problema é baseado no trabalho da Ana Elisa Araújo Martins – Diagnose de falhas de uma unidade de separação trifásica usando modelos a eventos [7], e ela desenvolveu a modelagem de sistemas a eventos discretos para uma planta de processamento primário de petróleo. Como pode ser visto na Figura 9, o petróleo do fundo do mar é extraído com o auxílio de vários equipamentos que estão interligados à plataforma. O processamento primário é o primeiro tratamento do petróleo na Plataforma. Seu objetivo é efetuar a separação gás/óleo/água do fluído que é extraído e ainda não foi processado.

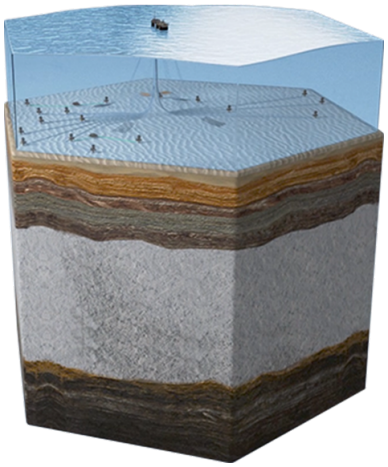

Figura 9: Plataforma de exploração offshore do Pré-Sal. [8]

No trabalho de Martins, é realizada a simulação de uma planta de escoamento multifásico que é composta por um separador e três hidrociclones. A planta foi utilizada em seu trabalho para realizar diagnóstico de falhas através de simulação, utilizando dados de uma plataforma da Bacia de Campos. O fluído que entra no reservatório é separado entre gás, óleo e água, e seu sistema pode ser visto na Figura 10.

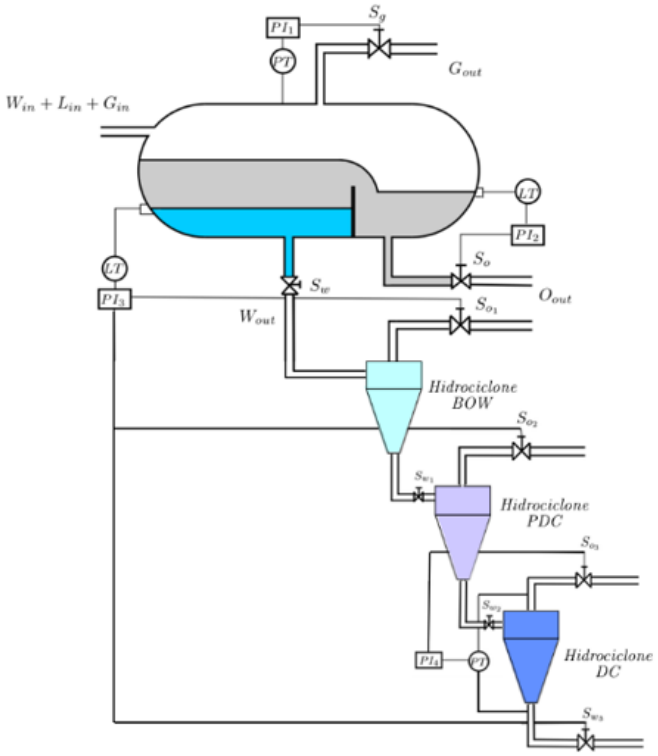

Figura 10: Sistema multifásico do problema abordado. [7]

Devido ao fato de que as partículas menores sobem, o gás é extraído pela válvula de gás, assim como para cada fluido existente, uma válvula e um sensor de nível correspondente controla o escape do mesmo. Este sistema envolve criticidade devido ao fato de que nenhuma variável pode exceder seus níveis de capacidade, para evitar trágicos acidentes.

A modelagem da planta corresponde a composição paralela do modelo da válvula de controle e do controlador de nível. Para cada par válvula/controlador é realizado o mesmo trabalho de modelagem, sendo que o comportamento de cada um, é adequado tanto para as válvulas de nível quanto à válvula de pressão. O funcionamento é o seguinte: o comando de abrir a válvula ocorre quando o valor da variável está acima do setpoint, ou então o comando de fechar a válvula ocorre quando o valor da variável está abaixo do setpoint. Portanto, o problema de modelagem está em manter o nível estável dentro do setpoint.

5.1 Modelagem da válvula

A modelagem da válvula foi feita com base na abertura que ela pode ficar. A válvula vai ficar totalmente aberta (VO) se ocorrer um evento de abrir totalmente a válvula (ot). Seguindo a mesma lógica, ela ainda pode fechar totalmente ou abrir/fechar parcialmente, conforme pode ser visto na tabela 1.

Devido ao fato de que ao inicializar o sistema não é possível saber em qual estado a válvula deve estar, há um estado inicial Vi que pode assumir qualquer um dos estados. O autômato da válvula pode ser visto na Figura 11.

Tabela 1: Estados e Eventos da válvula.

Estado	**Descrição**
VC	totalmente fechada
VO	totalmente aberta
VPOC	parcialmente aberta
Vi	estado inicial
Evento	**Descrição**
ocp	abrir ou fechar em posição intermediária
ot	abrir a válvula totalmente
ct	fechar tortalmente
re	reset

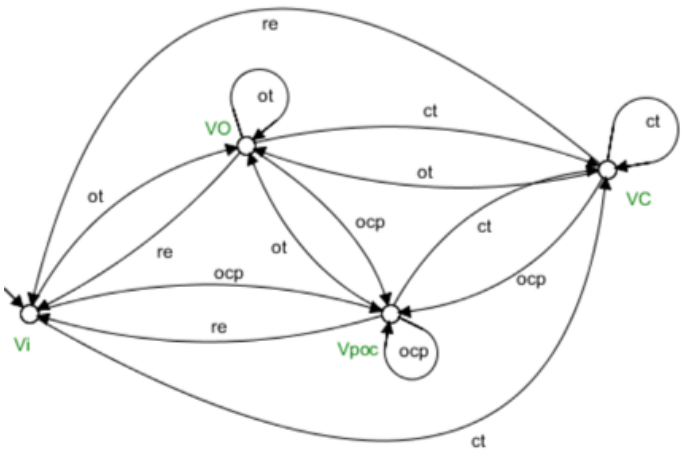

Figura 11: Modelo da válvula

5.2 Modelagem do controlador

Já a modelagem do controlador é feita com base no nível que o sistema se encontra. O controlador pode assumir o estado acima do setpoint subindo (CUR), se o nível do tanque estiver acima do setpoint subindo (nur). Seguindo a mesma lógica, o nível do sistema ainda pode estar acima do setpoint descendo, abaixo do setpoint descendo, e abaixo do setpoint subindo, conforme pode ser visto na Tabela 2.

O controlador só vai assumir o estado de controlador em nível no setpoint (CSP) quando o sensor do sistema determinar que o nível está atingindo o setpoint (nsp). Além disso, quando o nível estiver alto é permitida que a válvula possa abrir para reduzir o volume a fim de que se alcance o setpoint. Devido ao fato de que na inicialização o estado do controlador não é conhecido, há um estado inicial que pode assumir qualquer um dos estados. O autômato do controlador pode ser visto na Figura 12.

Tabela 2: Estados e eventos do controlador.

Estado	**Descrição**
CSP	nível no setpoint
CDD	abaixo do setpoint, descendo
CDR	abaixo do setpoint, subindo
CUD	acima do setpoint, descendo
CUR	acima do setpoint, subindo
CI	estado inicial
Evento	**Descrição**
nsp	sensor no setpoint
ndd	nível abaixo do setpoint, descendo
ndr	nível abaixo do setpoint, subindo
nud	nível acima do setpoint, descendo
nur	nível acima do setpoint, subindo
re	evento de reset

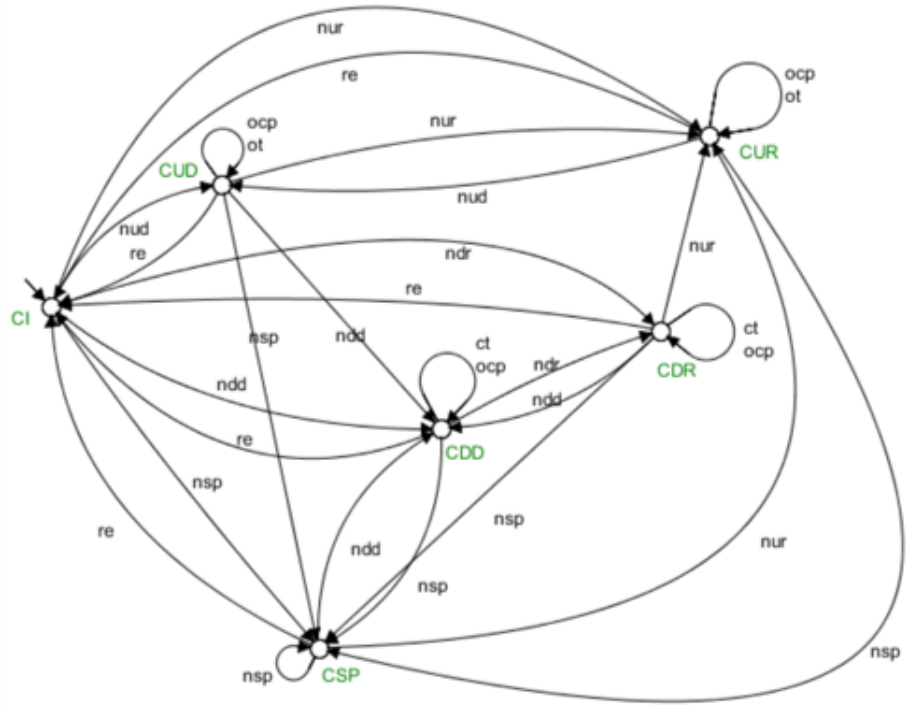

Figura 12: Modelo do controlador.

Um exemplo de representação dos níveis do reservatório pode ser visto na Figura 13 para os níveis altos, e na Figura 14 para os níveis baixos.

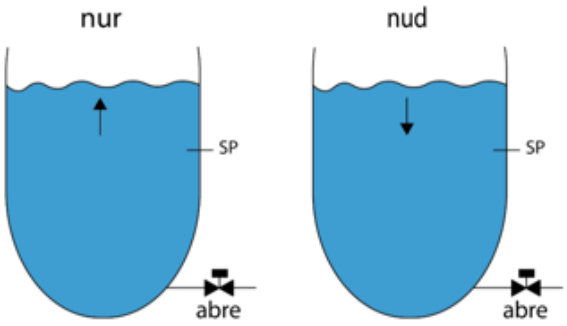

Figura 13: Níveis altos.

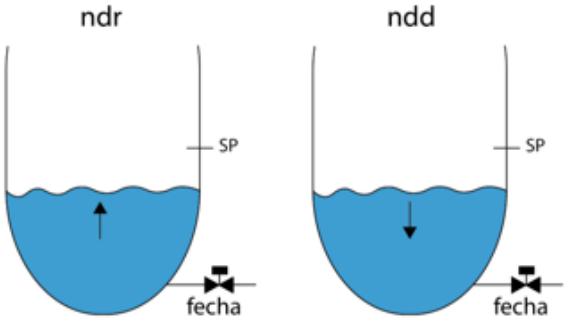

Figura 14: Níveis baixos.

5.3 Aplicação prática

A seguir será apresentado o código NuSMV desenvolvido para a aplicação do escoamento multifásico, ele pode ser encontrado no Apêndice B. O código será dividido em: descrição das variáveis, descrição da válvula, descrição do controlador e condição inicial, similar ao que foi feito no exemplo do elevador.

5.3.1 Declaração das variáveis

As variáveis declaradas para esse problema correspondem ao autômato da válvula e do controlador. Portanto serão 4 conjuntos: i) os estados da válvula, ii) os eventos de abertura da válvula, iii) os estados do controlador, e iv) os níveis do sistema. O trecho desenvolvido para esta parte do código pode ser visto na Figura 15.

```
MODULE main
VAR
    valvula :  {VI,VO,VC,VPOC};
    movimento : {ocp,ot,ct,re};
    controlador : {CI,CSP,CDD,CDR,CUD,CUR};
    nivel : {nsp,ndd,ndr,nud,nur,re};
```

Figura 15: Declaração das variáveis do sistema multifásico.

5.3.2 Descrição da válvula

A válvula é descrita de acordo com as suas relações de transição. Elas indicam qual posição a válvula pode assumir a depender do movimento que ela pode executar. Por exemplo: se a válvula estiver aberta e ocorrer o movimento de fechar a válvula, o próximo estado será válvula fechada. O trecho desenvolvido para esta parte do código pode ser visto na Figura 16.

```
ASSIGN
    next(valvula):= case
            valvula=VI & movimento=ot : VO;
            valvula=VI & movimento=ocp : VPOC;
            valvula=VI & movimento=ct : VC;
            valvula=VO & movimento=re : VI;
            valvula=VO & movimento=ot : VO;
            valvula=VO & movimento=ct : VC;
            valvula=VO & movimento=ocp : VPOC;
            valvula=VPOC & movimento=ocp : VPOC;
            valvula=VPOC & movimento=re : VI;
            valvula=VPOC & movimento=ot : VO;
            valvula=VPOC & movimento=ct : VC;
            valvula=VC & movimento=ct : VC;
            valvula=VC & movimento=re : VI;
            valvula=VC & movimento=ot : VO;
            valvula=VC & movimento=ocp : VPOC;
            TRUE : {VI,VO,VC,VPOC};
                esac;
```

Figura 16: Declaração das transições da válvula do sistema multifásico.

5.3.3 Descrição do controlador

Da mesma forma, a descrição do controlador irá estabelecer as suas relações de transições. Com isso, será indicado para qual estado o controlador pode assumir uma posição a depender do nível que o tanque se encontra. Por exemplo: se o controlador estiver em estado de setpoint e o nível aumentar, o controlador assume o estado controlador em nível alto. Caso o controlador estiver em nível alto, a válvula pode realizar o movimento de abrir. O trecho desenvolvido para esta parte do código pode ser visto na Figura 17.

```
next(controlador):= case
    controlador=CI & nivel=nur : CUR;
    controlador=CI & nivel=nud : CUD;
    controlador=CI & nivel=ndr : CDR;
    controlador=CI & nivel=ndd : CDD;
    controlador=CI & nivel=nsp : CSP;
    controlador=CSP & nivel=nsp : CSP;
    controlador=CSP & nivel=re : CI;
    controlador=CSP & nivel=ndd : CDD;
    controlador=CSP & nivel=nur : CUR;
    controlador=CUR & (movimento=ocp | movimento=ot) : CUR;
    controlador=CUR & nivel=re : CI;
    controlador=CUR & nivel=nud : CUD;
    controlador=CUR & nivel=nsp : CSP;
    controlador=CUD & (movimento=ocp | movimento=ot) : CUD;
    controlador=CUD & nivel=re : CI;
    controlador=CUD & nivel=nur : CUR;
    controlador=CUD & nivel=ndd : CDD;
    controlador=CUD & nivel=nsp : CSP;
    controlador=CDD & (movimento=ocp | movimento=ct) : CDD;
    controlador=CDD & nivel=re : CI;
    controlador=CDD & nivel=nsp : CSP;
    controlador=CDD & nivel=ndr : CDR;
    controlador=CDR & (movimento=ocp | movimento=ct) : CDR;
    controlador=CDR & nivel=re : CI;
    controlador=CDR & nivel=ndd : CDD;
    controlador=CDR & nivel=nsp : CSP;
    controlador=CDR & nivel=nur : CUR;
    TRUE : {CI,CSP,CDD,CDR,CUD,CUR};
        esac;
```

Figura 17: Declaração das transições do controlador do sistema multifásico.

5.3.4 Condição inicial

As condições iniciais do sistema indicarão a posição inicial da válvula e do controlador. O trecho desenvolvido para esta parte do código pode ser visto na Figura 18.

```
init(valvula)       := VI;
init(controlador)   := CI;
```

Figura 18: Declaração dos estados iniciais do sistema multifásico.

5.3.5 Verificação das propriedades

Após a construção do código, resta verificar as propriedades de nosso interesse para saber se o o modelo está funcionando conforme o desejado. A verificação formal desta aplicação será dividida em verificação de: acessibilidade, segurança, vivacidade, liberdade de deadlock e justiça. A seguir será apresentada cada uma delas, com exemplos e sintaxe CTL correspondente.

Acessibilidade A propriedade de acessibilidade tem o intuito de verificar se alguma situação pode ser alcançada. Algumas sentenças foram elaboradas para tal verificação. São elas:

- "O controlador sempre pode retornar ao SetPoint"(A1)
- "Não podemos ter um estado de nível subindo e uma válvula fechada"(A2)

- "Sempre podemos retornar para uma condição inicial"(A3)
- "Verifica se no futuro cada um dos estados será alcançado um dia"(A4)

Para cada uma delas, a sintaxe CTL representativa é apresentada a seguir: *AG (EF controlador=CSP)* para **(A1)**, *!EF (nivel=nur valvula=VC)* para **(A2)**, *AG EF (controlador=CI valvula=VI)* para **(A3)**, *AG EF ((controlador=CI | controlador=CSP | controlador=CDD | controlador=CDR | controlador=CUD | controlador=CUR))* para **(A4)**.

Segurança As propriedades de segurança têm o intuito de verificar se um evento nunca pode ocorrer, principalmente quando se refere a uma situação perigosa. Algumas sentenças foram elaboradas para tal verificação. São elas:

- "Desde que a válvula não esteja fechada, o nível do tanque não sobre"(S1)
- "A válvula e o controlador serem resetados nunca ocorrerá" (S2)

Para cada uma delas, a sintaxe CTL representativa é apresentada a seguir: *A[(!(nivel=nur)) U (valvula=VC)]* para **(S1)**, *AG !(movimento=re nivel=re)* para **(S2)**.

Vivacidade As propriedades de vivacidade têm o intuito de verificar se sob certas condições, algum evento acabará por ocorrer. Algumas sentenças foram elaboradas para tal verificação. São elas:

- "Se a válvula abrir, o nível do tanque vai abaixar eventualmente"(V1)
- "Verifica se um determinado estado do controlador não vai ocorrer sempre"(V2)

Para cada uma delas, a sintaxe CTL representativa é apresentada a seguir: *AG(movimento=ot -> EF nivel=ndd)* para **(V1)**, *AG ((controlador=CI -> EF !(controlador=CI)) (controlador=CSP -> EF !(controlador=CSP)) (controlador=CDD -> EF !(controlador=CDD)) (controlador=CDR -> EF !(controlador=CDR)) (controlador=CUD -> EF !(controlador=CUD)) (controlador=CUR -> EF !(controlador=CUR)))* para **(V2)**.

Liberdade de Deadlock A propriedade de liberdade de deadlock tem o intuito de verificar se algum evento indesejável nunca ocorrerá. Para isso foi construída a seguinte sentença para tal verificação:

- "Qualquer que seja o estado alcançado, existirá um sucessor imediato"(D1)

A sintaxe CTL representativa é apresentada a seguir: *AG EX TRUE*.

Justiça As propriedades de justiça têm o intuito de verificar se sob certas condições, um evento ocorrerá (ou deixará de ocorrer) indefinidamente. Algumas sentenças foram elaboradas para tal verificação. São elas:

- "O nível do tanque subirá infinitamente frequentemente"(J1)
- "O nível do tanque descerá infinitamente frequentemente"(J2)

Para cada uma delas, a sintaxe CTL representativa é apresentada a seguir: *AG AF (nivel=nur)* para **(J1)**, *AG AF (nivel=ndd)* para **(J2)**.

5.4 Resultados

As propriedades foram simuladas conforme descritas na seção 5.3.5 anteriormente apresentadas. O resultado das simulações a sentença verificada na especificação, como verdadeira ou falsa. No caso de uma verificação falsa, um contraexemplo é apresentado. A seguir serão apresentados os resultados das verificações de cada uma das propriedades.

5.4.1 Acessibilidade

As propriedades de acessibilidade resultaram em 3 propriedades verdadeiras e uma falsa, conforme pode ser visto na Figura 19. A verificação (A1), (A3) e (A4) são verdadeiras, enquanto (A2) é falsa. Conforme pode ser visto na Figura, a sequência que invalida tal afirmação é apresentada pelo programa. O fato da válvula fechar no passo 1.2, em seguida estar fechada no passo 1.3 e o nível subir, invalida a afirmação de não poderia haver um estado de nível alta com a válvula fechada.

```
*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (EF controlador = CSP)  is true
-- specification !(EF (nivel = nur & valvula = VC))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    valvula = VI
    movimento = ocp
    controlador = CI
    nivel = nsp
  -> State: 1.2 <-
    valvula = VPOC
    movimento = ct
    controlador = CSP
    nivel = nud
  -> State: 1.3 <-
    valvula = VC
    movimento = ocp
    controlador = CI
    nivel = nur
-- specification AG (EF (((((controlador = CI | controlador = CSP) | controlador = CDD) | controlador = CDR) | controlador = CUD) |
trolador = CUR))  is true
-- specification AG (EF (controlador = CI & valvula = VI))  is true
PS C:\Program Files\NuSMV-2.6.0-win64\bin>
```

Figura 19: Resultado da verificação das propriedades de acessibilidade.

5.4.2 Segurança

As propriedades de segurança resultaram em 2 propriedades falsas, conforme pode ser visto na Figura 20. Percebe-se que as sequências que invalidam tais afirmações são

apresentadas pelo programa. Para (S1) é observado no passo 2.2 que o nível consegue subir antes mesmo da válvula fechar. Já para (S2) é observado no passo 1.2 que o movimento de reset pode ocorrer por parte da válvula e do controlador ao mesmo tempo.

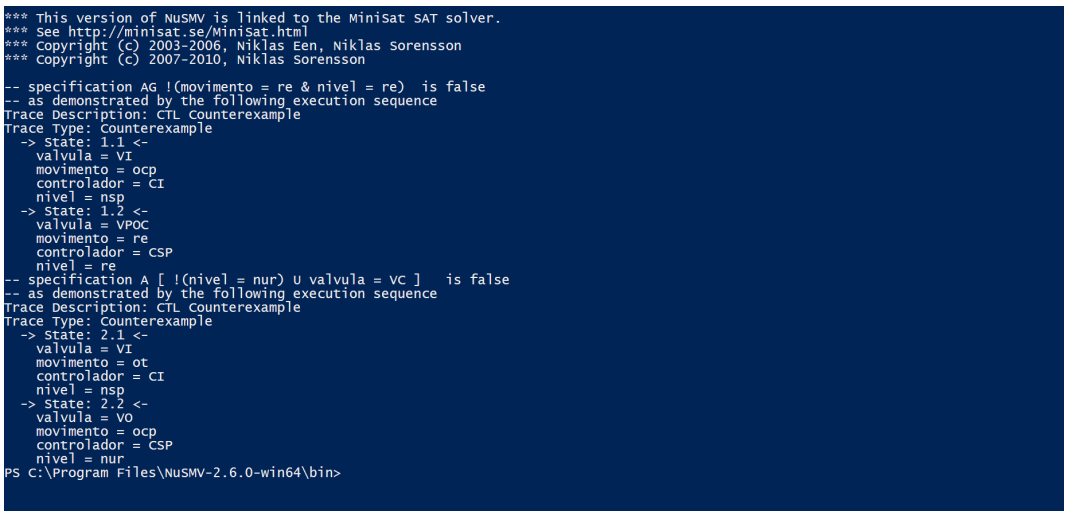

Figura 20: Resultado da verificação das propriedades de segurança.

5.4.3 Vivacidade

As propriedades de vivacidade resultaram em 2 propriedades verdadeiras (V1 e V2), conforme pode ser visto na Figura 21.

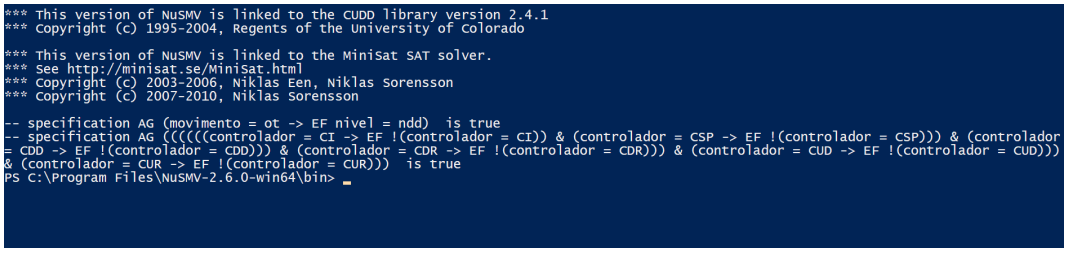

Figura 21: Resultado da verificação das propriedades de vivacidade.

5.4.4 Liberdade de Deadlock

A propriedade de liberdade de deadlock resultou em verdadeira (D1), conforme pode ser visto na Figura 22.

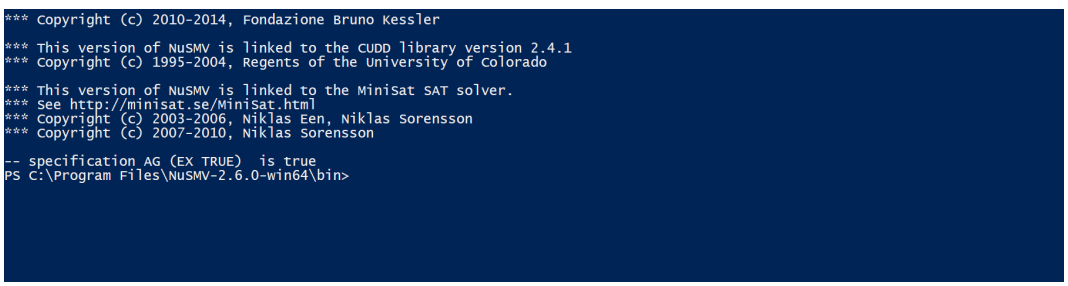

Figura 22: Resultado da verificação das propriedades de liberdade de deadlock.

5.4.5 Justiça

As propriedades de justiça resultaram em 2 propriedades falsas, conforme pode ser visto na Figura 23. É possível verificar que as sequências que invalidam tais afirmações são apresentadas pelo programa. Para (J1), é verificado que o nível não subirá indefinidamente frequentemente, já que os passos apontam que o nível do sistema pode variar. Para (J2) acontece algo similar. Nesse caso é verificado que o nível não abaixará indefinidamente frequentemente, pelos mesmos motivos.

```
-- specification AG (AF nivel = nur)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    valvula = VI
    movimento = ocp
    controlador = CI
    nivel = nsp
  -> State: 1.2 <-
    valvula = VPOC
    movimento = re
    controlador = CSP
    nivel = nud
  -> State: 1.3 <-
    valvula = VI
    movimento = ocp
    controlador = CI
    nivel = nsp
-- specification AG (AF nivel = ndd)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 2.1 <-
    valvula = VI
    movimento = ocp
    controlador = CI
    nivel = nsp
  -> State: 2.2 <-
    valvula = VPOC
    movimento = re
    controlador = CSP
    nivel = nud
  -> State: 2.3 <-
    valvula = VI
    movimento = ocp
    controlador = CI
    nivel = nsp
PS C:\Program Files\NuSMV-2.6.0-win64\bin>
```

Figura 23: Resultado da verificação das propriedades de justiça.

6 Conclusão

Este trabalho conseguiu introduzir os aprendizados do NuSMV através de exemplos teóricos, características da ferramenta, instruções de uso, de instalação e execução, e apresentação da estrutura. Devido a pouca experiência de utilização, este trabalho foi voltado ao uso dos principais recursos da ferramenta, mas mesmo assim conseguiu atingir o objetivo de propiciar uma experiência inicial para quem deseja aprender mais sobre o assunto.

Em termos da ferramenta, ela requer poucas habilidades computacionais para instalação e execução. Através dos exemplos mencionados foi possível construir analogias para construção do código do problema do escoamento multifásico. Em termos dos resultados encontrados, foi possível analisar importantes propriedades principalmente da segurança do equipamento. As propriedades (A2) e (S1) sinalizaram um ponto de atenção para uma verificação posterior do modelo, pois apontam que o nível do sistema pode subir em situações indesejáveis.

Espera-se que trabalhos futuros venham acrescentar mais conteúdo ao trabalho já construído, através da utilização de mais funcionalidades, aplicação em diversos outros problemas e a exibição do NuSMV frente às outro programas de verificação formal, afim de difundir mais essa importante ferramenta.

Repositório GitHub contendo o conteúdo deste trabalho:

`https://github.com/rqmoser/VF_NuSMV`

Referências

[1] M. Carnegie, "Model checking guided tour." Disponível em: <http://www.cs.cmu.edu/~modelcheck/tour.htm>, Acesso em: 09 mar. 2022.

[2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, and P. McKenzie, *Systems and Software Verification*. Springer Berlin Heidelberg, 2001.

[3] FBK-IRST, "An overview of nusmv." Disponível em: <https://nusmv.fbk.eu/NuSMV/>, Acesso em: 25 mar. 2022.

[4] R. Cavada, A. Cimatti, G. Keighren, E. Olivetti, M. Pistore, and M. Roveri, "Nusmv 2.6 tutorial."

[5] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, "Nusmv 2.6 user manual."

[6] DREAMSTIME, "Homens e mulheres no elevador, vetor desenhado à mão." Disponível em: <https://pt.dreamstime.com/homens-e-mulheres-no-elevador-vetor-desenhado-%C3%A0-m%C3%A3o-desenho-de-image166262561>, Acesso em: 09 mar. 2022.

[7] A. E. A. Martins, "(tcc) diagnose de falhas de uma unidade de separação trifásica usando modelos a eventos discretos," 2018.

[8] Petrobras, "Pré-sal." Disponível em: https://petrobras.com.br/pt/nossas-atividades/areas-de-atuacao/exploracao-e-producao-de-petroleo-e-gas/pre-sal/, Acesso em: 18 set. 2021.

A Código desenvolvido para o Problema do Elevador

```
1  % Autor: Erique Moser
2  % Data: 27/03/2022
3  % Trabalho: Model Checking no NuSMV
4  % Disciplina: Verificacao Formal
5  % Professor Orient.: Max Hering de Queiroz
6  % UNIVERSIDADE FEDERAL DE SANTA CATARINA ..
7  % - CAMPUS FLORIANOPOLIS
8  % DEPARTAMENTO DE AUTOMACAO E SISTEMAS
9  % Mestrado.
10
11 MODULE main
12 VAR
13     cabin : 0 .. 3;
14     dir :  {up,down};
15     request: array 0 .. 3 of boolean;
16
17 ASSIGN
18     next(cabin):=   case
19                         dir=up & cabin < 3 : cabin + 1;      --moves up
20                         dir=down & cabin > 0 : cabin - 1;    --moves down
21                         TRUE : cabin;                            --stuck
22                     esac;
23
24                             --sempre que chega em uma das ...
                                   extremidades, muda direcao
25     next(dir) :=    case
26                         dir=up & next(cabin) = 3 : down;     --switch dir
27                         dir=down & next(cabin) = 0 : up;     -- switch dir
28                         TRUE: dir;                               ...
                               --keep on
29                     esac;
30                             --aqui a logica esta em imaginar que o ...
                                   request so vai mudar seu ...
                                   comportamento para..
31                             --requisicao verdadeiramente feita, se um ...
                                   request for feito, ou entao...
32                             --a partir do momento que ele chega ...
                                   proximo do andar, a requisicao eh ...
                                   desfeita.
33     next(request[0]) := case
34                             next(cabin) = 0 : FALSE;     --disappears
35                             request[0] : TRUE;           --remains
36                             TRUE : {FALSE,TRUE};
37                         esac;
38     next(request[1]) := case
```

```
39                              next(cabin) = 1 : FALSE;    --disappears
40                              request[1] : TRUE;          --remains
41                              TRUE : {FALSE,TRUE};
42                          esac;
43      next(request[2]) := case
44                              next(cabin) = 2 : FALSE;    --disappears
45                              request[2] : TRUE;          --remains
46                              TRUE : {FALSE,TRUE};
47                          esac;
48      next(request[3]) := case
49                              next(cabin) = 3 : FALSE;    --disappears
50                              request[3] : TRUE;          --remains
51                              TRUE : {FALSE,TRUE};
52                          esac;
53
54      init(cabin)         := 0;
55      init(dir)           := up;
56      init(request[0])    := FALSE;
57      init(request[1])    := FALSE;
58      init(request[2])    := FALSE;
59      init(request[3])    := FALSE;
60
61  SPEC
62      AG EX TRUE
63  SPEC
64      AG (AF!request[0] & AF!request[1] & AF!request[2] & AF!request[3])
65  SPEC
66      AG AF (!request[0] & !request[1] & !request[2] & !request[3])
```

B Código desenvolvido para o Problema do Escoamento Multifásico

```
1  % Autor: Erique Moser
2  % Data: 27/03/2022
3  % Trabalho: Model Checking no NuSMV
4  % Disciplina: Verificacao Formal
5  % Professor Orient.: Max Hering de Queiroz
6  % UNIVERSIDADE FEDERAL DE SANTA CATARINA ..
7  % - CAMPUS FLORIANOPOLIS
8  % DEPARTAMENTO DE AUTOMACAO E SISTEMAS
9  % Mestrado.
10
11 MODULE main
12 VAR
13     valvula :  {VI,VO,VC,VPOC};
```

```
14      movimento : {ocp,ot,ct,re};
15      controlador : {CI,CSP,CDD,CDR,CUD,CUR};
16      nivel : {nsp,ndd,ndr,nud,nur,re};
17
18  ASSIGN
19      next(valvula):= case
20              valvula=VI & movimento=ot : VO;
21              valvula=VI & movimento=ocp : VPOC;
22              valvula=VI & movimento=ct : VC;
23              valvula=VO & movimento=re : VI;
24              valvula=VO & movimento=ot : VO;
25              valvula=VO & movimento=ct : VC;
26              valvula=VO & movimento=ocp : VPOC;
27              valvula=VPOC & movimento=ocp : VPOC;
28              valvula=VPOC & movimento=re : VI;
29              valvula=VPOC & movimento=ot : VO;
30              valvula=VPOC & movimento=ct : VC;
31              valvula=VC & movimento=ct : VC;
32              valvula=VC & movimento=re : VI;
33              valvula=VC & movimento=ot : VO;
34              valvula=VC & movimento=ocp : VPOC;
35              TRUE : {VI,VO,VC,VPOC};
36                      esac;
37
38          next(controlador):= case
39              controlador=CI & nivel=nur : CUR;
40              controlador=CI & nivel=nud : CUD;
41              controlador=CI & nivel=ndr : CDR;
42              controlador=CI & nivel=ndd : CDD;
43              controlador=CI & nivel=nsp : CSP;
44              controlador=CSP & nivel=nsp : CSP;
45              controlador=CSP & nivel=re : CI;
46              controlador=CSP & nivel=ndd : CDD;
47              controlador=CSP & nivel=nur : CUR;
48              controlador=CUR & (movimento=ocp | movimento=ot) : CUR;
49              controlador=CUR & nivel=re : CI;
50              controlador=CUR & nivel=nud : CUD;
51              controlador=CUR & nivel=nsp : CSP;
52              controlador=CUD & (movimento=ocp | movimento=ot) : CUD;
53              controlador=CUD & nivel=re : CI;
54              controlador=CUD & nivel=nur : CUR;
55              controlador=CUD & nivel=ndd : CDD;
56              controlador=CUD & nivel=nsp : CSP;
57              controlador=CDD & (movimento=ocp | movimento=ct) : CDD;
58              controlador=CDD & nivel=re : CI;
59              controlador=CDD & nivel=nsp : CSP;
60              controlador=CDD & nivel=ndr : CDR;
61              controlador=CDR & (movimento=ocp | movimento=ct) : CDR;
62              controlador=CDR & nivel=re : CI;
```

```
63              controlador=CDR & nivel=ndd : CDD;
64              controlador=CDR & nivel=nsp : CSP;
65              controlador=CDR & nivel=nur : CUR;
66              TRUE : {CI,CSP,CDD,CDR,CUD,CUR};
67                      esac;
68
69      init(valvula)       := VI;
70      init(controlador)   := CI;
71
72
73  --Propriedades da Logica Temporal:
74
75  --*************acessibilidade**********************
76  --diz que alguma situacao pode ser alcancada
77  --("O controlador sempre pode retornar ao SetPoint")
78  SPEC AG (EF controlador=CSP)
79
80  --("nao podemos ter um estado de nivel subindo e uma valvula fechada")
81  SPEC !EF (nivel=nur & valvula=VC)
82
83  --("sempre podemos retornar para uma condicao inicial")
84  SPEC AG EF (controlador=CI & valvula=VI)
85
86  --("Verifica se no futuro cada um dos estados serao alcancados um dia")
87  SPEC AG EF ((controlador=CI | controlador=CSP | controlador=CDD | ...
       controlador=CDR |
88   controlador=CUD | controlador=CUR))
89
90  --*************seguranca**********************
91  --diz que um evento nunca pode ocorrer
92  --("desde que a valvula nao esteja fechada, o nivel do tanque nao sobe")
93  SPEC A[(!(nivel=nur)) U (valvula=VC)]
94
95  --("a valvula e o controlador serem resetados nunca ocorrera)
96  SPEC AG !((movimento=re) & (nivel=re))
97
98  --*************vivacidade**********************
99  --diz que sob certas condicoes, algum evento acabara por ocorrer
100 --("se a valvula abrir, o nivel do tanque vai abaixar eventualmente")
101 SPEC AG(movimento=ot -> EF nivel=ndd)
102
103 --("verifica se um determinado estado do controlador nao vai ocorrer ...
       sempre")
104 SPEC AG ((controlador=CI -> EF !(controlador=CI)) & (controlador=CSP ...
       -> EF !(controlador=CSP)) &
105      (controlador=CDD -> EF !(controlador=CDD)) & (controlador=CDR -> ...
           EF !(controlador=CDR)) &
106      (controlador=CUD -> EF !(controlador=CUD)) & (controlador=CUR -> ...
           EF !(controlador=CUR)))
```

```
107
108 --**************liberdade de deadlock***********************
109 --diz que algum evento indesejavel nunca ocorrera
110 --("qualquer que seja o estado alcancado, existira um sucessor imediato")
111 SPEC AG EX TRUE
112
113 --**************justica***********************
114 --sob certas condicoes, um evento ocorrera (ou deixara de ocorrer) ...
       indefinidamente
115 --("o nivel do tanque subira infinitamente frequentemente")
116 SPEC AG AF (nivel=nur)
117 --("o nivel do tanque descera infinitamente frequentemente")
118 SPEC AG AF (nivel=ndd)
```