



**UNIVERSIDADE ESTADUAL VALE DO ACARAÚ - UVA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA - CCET
CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

STEPHANY DE ALMEIDA GONÇALVES

**INTRODUÇÃO À LÓGICA TEMPORAL ATRAVÉS DO USO DA
FERRAMENTA NUSMV**

STEPHANY DE ALMEIDA GONÇALVES

INTRODUÇÃO À LÓGICA TEMPORAL ATRAVÉS DO USO DA FERRAMENTA
NUSMV

Monografia apresentada ao Curso de Ciências da Computação da Universidade Estadual Vale do Acaraú - UVA como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Hudson Costa Gonçalves da Cruz

INTRODUÇÃO À LÓGICA TEMPORAL ATRAVÉS DO USO DA FERRAMENTA NUSMV

Monografia apresentada ao Curso de Ciências da Computação da Universidade Estadual Vale do Acaraú - UVA como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Stephany de Almeida Gonçalves
Stephany de Almeida Gonçalves

Monografia aprovada em: 24/ 02/ 2021


Orientador: Hudson Costa Gonçalves
Hudson Costa Gonçalves da Cruz (UVA)

1º Examinador: Hudson Costa Gonçalves
Esp. Hudson Costa Gonçalves da Cruz (UVA)

2º Examinador: Gilzamir Ferreira Gomes
Me. Gilzamir Ferreira Gomes (UVA)

3º Examinador: Paulo Régis Menezes Souza
Dr. e Me. Paulo Régis Menezes Souza (UVA)

Coordenador do Curso:

 UNIVERSIDADE ESTADUAL VALE DO ACARAÚ
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
Hudson Costa Gonçalves
Hudson Costa G. da Cruz
COORDENADOR

Hudson Costa Gonçalves da Cruz

RESUMO

A Lógica Temporal está relacionada ao estudo da lógica ligada ao tempo, tendo como base a Lógica Modal, permitindo representar a variação dos estados do mundo em diferentes períodos de tempo. Esta Lógica é de grande valor para a Ciência da Computação, podendo ser aplicada para declarações de sistemas de hardware e software, sendo a ferramenta NuSMV uma das mais usadas para verificação de sistemas que usam Lógica Temporal. Este trabalho tem como objetivo principal a produção de um material introdutório para o conhecimento sobre Lógicas Modais em especial a Lógica Temporal, através do uso da ferramenta NuSMV. Trata-se de um estudo bibliográfico baseado na leitura de textos, livros e artigos científicos disponíveis em diversas fontes na internet. A pesquisa apresenta um pouco sobre a história da Lógica e seu desenvolvimento, abordando sobre Lógica Proposicional, Lógica Modal, Lógica Temporal e a ferramenta NuSMV.

Palavras-chave: Lógica. Lógica Proposicional. Lógica Modal. Lógica Temporal. NuSMV.

ABSTRACT

Temporal Logic is related to the study of time-related logic, based on Modal Logic, allowing to represent the variation of the states of the world in different periods of time. This Logic is of great value for Computer Science, being able to be applied for declarations of hardware and software systems, being the NuSMV tool one of the most used to verify systems that use Temporal Logic. This work has as main objective the production of an introductory material for the knowledge about Modal Logics, especially Temporal Logic, through the use of the NuSMV tool. This is a bibliographic study based on the reading of texts, books and scientific articles available from different sources on the internet. The research presents a little about the history of Logic and its development, addressing Propositional Logic, Modal Logic, Temporal Logic and the NuSMV tool.

Keywords: Logic. Propositional Logic. Modal Logic. Temporal Logic. NuSMV.

LISTA DE ILUSTRAÇÕES

| | |
|--|----|
| Figura 1 – Tabela da Negação | 20 |
| Figura 2 – Tabela da Conjunção | 20 |
| Figura 3 – Tabela da Disjunção | 21 |
| Figura 4 – Tabela da Implicação | 21 |
| Figura 5 – Tabela da Bi-Implicação | 21 |
| Figura 6 – Representação de Tautologia pelo princípio do terceiro excluído | 22 |
| Figura 7 – Exemplo de satisfatibilidade | 23 |
| Figura 8 – Exemplo de contradição | 23 |
| Figura 9 – Representação de Tautologia pelo princípio da não-contradição | 23 |
| Figura 10 – Exemplo de contingência | 24 |
| Figura 11 – Tabela da implicação semântica | 25 |
| Figura 12 – Tabela verdade | 26 |
| Figura 13 – Tabela verdade para a proposição $\neg(P \wedge \neg Q)$ | 26 |
| Figura 14 – Exemplo de árvore semântica | 27 |
| Figura 15 – Criação da árvore semântica- Etapa 1 | 27 |
| Figura 16 – Criação da árvore semântica- Etapa 2 | 28 |
| Figura 17 – Criação da árvore semântica- Etapa 3 | 29 |
| Figura 18 – Exemplo de Frame | 34 |
| Figura 19 – Modelo de Frame para a função V | 35 |
| Figura 20 – Modelo de Tempo Linear | 38 |
| Figura 21 – Modelo de Tempo Ramificado | 38 |
| Figura 22 – Operadores Temporais. | 41 |
| Figura 23 – Grafo de transição de estados e sua árvore de computação. | 42 |
| Figura 24 – Operadores temporais da lógica CTL. | 43 |
| Figura 25 – Construção de BDD | 46 |
| Figura 26 – Modelo de um sistema concorrente em NuSMV. | 49 |
| Figura 27 – Máquina de Estados do Sistema. | 50 |
| Figura 28 – Sistema de Transição simples | 51 |
| Figura 29 – Código NuSMV do sistema de transição simples | 51 |
| Figura 30 – Sistema de transição de um semáforo | 52 |
| Figura 31 – Especificação do semáforo em NuSMV | 53 |
| Figura 32 – Sistema de transição do contador de 3 bits | 53 |
| Figura 33 – Especificação em NuSMV do contador de 3 bits | 54 |
| Figura 34 – Sistema de transição de um anel de inversores | 55 |
| Figura 35 – Especificação em NuSMV do anel inversor | 55 |
| Figura 36 – sistema de transição de um processo | 56 |
| Figura 37 – sistema de transição dos processos com exclusão mútua | 56 |

| | |
|---|----|
| Figura 38 – Especificação NuSMV do processo de exclusão mútua | 57 |
| Figura 39 – Criação de nova variável do sistema | 58 |
| Figura 40 – Localizando o NuSMV para execução | 59 |
| Figura 41 – Execução do NuSMV | 59 |
| Figura 42 – Execução do código do semáforo | 60 |

SUMÁRIO

| | | |
|--------------|--|-----------|
| 1 | INTRODUÇÃO | 9 |
| 1.1 | JUSTIFICATIVA | 11 |
| 1.2 | OBJETIVO GERAL | 11 |
| 1.3 | OBJETIVOS ESPECÍFICOS | 11 |
| 1.4 | METODOLOGIA | 12 |
| 1.5 | ESTRUTURA DO TRABALHO | 12 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 13 |
| 2.1 | NOÇÕES BÁSICAS DE LÓGICA CLÁSSICA | 13 |
| 2.1.1 | Um pouco de história | 13 |
| 2.1.2 | Lógica Proposicional Clássica | 15 |
| 2.1.3 | Sintaxe da Lógica Proposicional | 16 |
| 2.1.3.1 | Conectivos Lógicos | 16 |
| 2.1.3.2 | Fórmulas Proposicionais | 18 |
| 2.1.3.3 | Precedência de Conectivos | 18 |
| 2.1.4 | Semântica da Lógica Proposicional | 19 |
| 2.1.5 | Valores de Verdade | 20 |
| 2.1.5.1 | Negação | 20 |
| 2.1.5.2 | Conjunção | 20 |
| 2.1.5.3 | Disjunção | 20 |
| 2.1.5.4 | Implicação | 21 |
| 2.1.5.5 | Bi-Implicação | 21 |
| 2.1.6 | Propriedades Semânticas da Lógica Proposicional | 22 |
| 2.1.6.1 | Tautologia | 22 |
| 2.1.6.2 | Satisfatibilidade | 22 |
| 2.1.6.3 | Contradição | 23 |
| 2.1.6.4 | Contingência | 24 |
| 2.1.6.5 | Equivalência Semântica | 24 |
| 2.1.6.6 | Implicação Semântica | 25 |
| 2.1.7 | Métodos de Validação de Argumentos | 25 |
| 2.1.7.1 | Tabela-Verdade | 25 |
| 2.1.7.2 | Árvore Semântica | 27 |
| 2.1.7.3 | Negação ou Redução ao Absurdo | 29 |
| 2.2 | LÓGICA MODAL | 30 |
| 2.2.1 | Sintaxe e Semântica | 31 |
| 2.2.1.1 | Sintaxe | 32 |
| 2.2.1.2 | Regras e Axiomas | 32 |

| | | |
|--------------|--|-----------|
| 2.2.1.3 | Semântica | 34 |
| 2.2.2 | Tipos de Lógicas Modais | 35 |
| 2.2.2.1 | Lógica Epistêmica | 35 |
| 2.2.2.2 | Lógica Deontica | 35 |
| 2.2.2.3 | Lógica Doxástica | 36 |
| 2.3 | LÓGICA TEMPORAL | 38 |
| 2.3.1 | Lógica Linear Temporal(LTL) | 39 |
| 2.3.1.1 | Sintaxe da LTL | 40 |
| 2.3.1.2 | Semântica da LTL | 40 |
| 2.3.2 | Lógica Computacional em árvore(CTL) | 41 |
| 2.3.2.1 | Sintaxe da CTL | 43 |
| 2.3.2.2 | Semântica da CTL | 44 |
| 2.3.3 | Aplicações da Lógica Temporal | 45 |
| 2.4 | DIAGRAMA BINÁRIO DE DECISÃO(BDD) | 46 |
| 2.5 | NUSMV | 46 |
| 2.5.1 | Sintaxe do NuSMV | 47 |
| 2.5.2 | Linguagem do NuSMV | 48 |
| 2.5.3 | Alguns exemplos utilizando NuSMV | 51 |
| 2.5.3.1 | Exemplo simples de transição | 51 |
| 2.5.3.2 | Exemplo de um semáforo simples | 52 |
| 2.5.3.3 | Contador de 3 bits | 53 |
| 2.5.3.4 | Anel de inversores | 55 |
| 2.5.3.5 | Exclusão mútua | 56 |
| 2.6 | EXECUÇÃO DO NUSMV | 58 |
| 3 | CONCLUSÃO E TRABALHOS FUTUROS | 61 |
| | REFERÊNCIAS | 62 |

1 INTRODUÇÃO

De acordo com (GORANKO; GALTON, 2015) Lógica Temporal pode ser entendida como um sistema de regras e símbolos de representação do raciocínio que abrange o tempo como elemento em uma estrutura lógica, sendo esta um tipo de abordagem da Lógica Modal, que foi introduzida por Arthur Prior por volta da década de 60 com o nome de Tense Logic ou Lógica Tensa, logo mais sendo desenvolvida por filósofos e cientistas da computação.

Inicialmente utilizada pelos filósofos, a Lógica Temporal está relacionada ao estudo da Lógica ligada ao tempo, tendo como operadores temporais F e P que denotam futuro e passado. Alguns sistemas lógicos como, por exemplo, Lógica Computacional em árvore (CTL) e Lógica Temporal Linear (LTL), tem como base a Lógica Temporal.

Em termos de verificação formal a Lógica Temporal é aplicada em declarações de sistemas de hardware e software, podendo dizer, por exemplo, que quando feita uma solicitação de um recurso o seu acesso será assegurado, não podendo ser assegurado a dois solicitantes ao mesmo tempo.

A Lógica Temporal é comumente interpretada através da estrutura de Kripke, que é um conjunto de estados e transições e uma função, que determina cada estado ao seu conjunto de propriedades verdadeiras, uma melhor abordagem sobre a estrutura de Kripke bem como sobre a Lógica Temporal será apresentada mais adiante neste trabalho.

Discussões sobre temporalidade e raciocínio sobre tempo podem ser vistas desde a antiguidade, podendo ser encontrados alguns exemplos até mesmo na Bíblia. Mas a referência científica mais antiga sobre raciocínio temporal pode ser encontrada nos trabalhos de Aristóteles, como uma antecipação da lógica temporal parcialmente desenvolvida como uma lógica modal temporal binária de primeira ordem. Na obra *On Interpretation*, Aristóteles argumenta que futuros contingentes não podem receber valores verdade no momento, pois não se pode garantir que possíveis eventos futuros acontecerão, como exemplo a frase “Haverá uma batalha naval amanhã” (GORANKO; GALTON, 2015).

A Lógica Temporal pouco se desenvolveu ao longo de séculos, até que no início dos anos 50, o lógico e filósofo neozelandês Arthur Prior (1914-1969), começou sua análise e formalização de argumentos que levaram ao desenvolvimento e criação da Lógica Temporal formal, iniciando a era moderna do raciocínio lógico temporal de extrema importância para a Filosofia, Ciência da Computação e Inteligência Artificial.

Na Ciência da Computação, a Lógica Temporal pode ser aplicada na verificação de programas com execução simultânea ou raciocínio de sistemas concorrentes, podendo ser também encontrada na Inteligência Artificial, em aplicações de construção de agentes que utilizam uma arquitetura que requer raciocínio lógico baseado em crenças e desejos. Na Engenharia de Software a Lógica Temporal é aplicada na melhoria dos ambientes de construção e qualidade de software. Em banco de dados, a Lógica Temporal é usada

para elaboração de regras de restrição de integridade e consulta a dados temporais. A Lógica Temporal é usada também como ferramenta para desenvolvimentos de sistemas de informação, compostos por módulos hierárquicos, utilizada na construção dos programas e no armazenamento e consulta de dados.

Segundo (RISSINO; TORRER; MARTINS, 2007) Lógica Temporal possui como base a Lógica Modal e permite representar a variação dos estados do mundo em diferentes períodos de tempo, ou seja, permite verificar a veracidade das asserções ao longo do tempo, já que uma asserção pode ser verdadeira em determinado instante de tempo e falsa em outro instante.

Sendo a Lógica Temporal um subcampo da Lógica Modal, é importante também falar sobre esta que deu origem a essa sub-Lógica.

A Lógica Modal é um tipo de lógica não clássica desenvolvida para estudar as diferentes modalidades nas expressões de possibilidade, impossibilidade, probabilidade e necessidade, sendo as mais usadas possibilidade e necessidade. De uma maneira mais formal a Lógica Modal é expressa através de operadores modais, por exemplo, a afirmação “pode chover hoje” e “talvez irá chover hoje”, expressam possibilidade, onde o operador “possível” está associado à frase “irá chover hoje”.

Os operadores modais geralmente são escritos como \Box e \Diamond para necessidade e possibilidade respectivamente, podendo ser expressas também por sua negação (\neg). Uma abordagem mais a fundo sobre Lógica Modal e seus operadores será posteriormente apresentada neste trabalho.

Dentro do campo da Lógica Modal são abordadas as Sub-lógicas Modais, como a Lógica Epistêmica, Temporal, Deôntica e a Doxástica, que tratam de termos como probabilidade, eventualidade, padronização, poder e dever.

A Lógica Modal tem sua origem nas obras de Aristóteles que serviu como base para que os megáricos e estóicos, entre outros lógicos posteriores pudessem desenvolver importantes trabalhos, como Pedro Abelardo (1079-1142) na era medieval, Leibniz, Hume e Kant na era moderna, C.I. Lewis e Hugh MacColl na era contemporânea, destacando também Von Wright, Arthur Prior e Saul Kripke, que foram responsáveis pelo surgimento das variações da Lógica Modal e suas semânticas.

Embora tenha sido originada desde Aristóteles na Grécia Antiga, a história da Lógica Modal começa por volta de 1906, quando Hugh MacColl publicou sua obra *Symbolic Logic and its Applications*, que serviu como base para o que seria a Lógica Modal tal como é hoje, sendo sucedido por vários outros lógicos como Clarence Irving Lewis com sua obra *A Survey of Symbolic* publicada em 1918. Kurt Gödel e Rodolf Carnap iniciaram sua procura em 1933 por uma estrutura matemática de uma lógica que lidasse com as três modalidades clássicas (possibilidade, necessidade e probabilidade).

Até que a Lógica Modal fosse estabilizada, vários modelos de sistemas foram propostos durante o século XX, como o sistema T, por Robert Feys em 1937 e o sistema M

por Georg Henrik Von Wright em 1951, mas foi com o desenvolvimento da semântica de Kripke, sistema modal normal mínimo K, proposto por Saul Kripke em 1965, que a Lógica Modal ganhou o seu padrão.

1.1 JUSTIFICATIVA

Considerando a importância da Lógica para a Ciência da Computação e visto que há apenas uma disciplina que trata de Lógica no curso de Ciência da Computação da UVA (Lógica Matemática, atualmente no 1º Período do curso), seria bastante enriquecedor para os estudantes terem acesso a outros tipos de Lógicas, como por exemplo, a Lógica Modal. Partindo dessa necessidade, o objetivo principal deste trabalho é produzir um material introdutório que possa ser utilizado como apoio, inicialmente para um minicurso de Lógica Modal com ênfase em Lógica Temporal através da ferramenta NuSMV.

O público alvo desta proposta não se limita somente aos estudantes de Ciências da Computação, abrange também qualquer pessoa que esteja interessado em entender, estudar e aprender sobre esta área, mas é recomendado que o leitor tenha algum conhecimento prévio em Lógica Matemática para melhor entendimento no assunto.

1.2 OBJETIVO GERAL

Este trabalho tem como objetivo proporcionar aos leitores deste material e interessados no assunto uma visão mais ampla sobre a Lógica Temporal e a ferramenta NuSMV.

1.3 OBJETIVOS ESPECÍFICOS

- Produção de um material introdutório para o conhecimento sobre a Lógica Modal com ênfase em Lógica Temporal através da ferramenta NuSMV
- Mostrar uma Introdução à Lógica Modal com ênfase na Lógica Temporal com base nos materiais selecionados durante a pesquisa.
- Falar sobre a ferramenta NuSMV, mostrando exemplos com Lógica Temporal e apresentar como usar a ferramenta.

1.4 METODOLOGIA

Para o desenvolvimento deste trabalho foi feita uma pesquisa de natureza qualitativa e a metodologia utilizada foi um estudo bibliográfico baseado na leitura de textos disponíveis em artigos científicos, livros e sites na internet, selecionados para busca de informações referentes aos temas sobre Lógica Modal, Lógica Temporal, Lógica Clássica, Lógica Proposicional, NuSMV, entre os assuntos descritos na seção 1.5 deste trabalho.

Após toda a coleta de dados dos materiais bibliográficos, foi desenvolvida uma Introdução à Lógica Temporal através da ferramenta NuSMV, e foi dividido em 3 capítulos descritos na seção 1.5.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está dividido em três capítulos, o primeiro se trata de uma breve introdução, apresentando os conceitos, origens, motivações, justificativa, objetivos geral e específicos e a metodologia.

O capítulo 2 traz um referencial teórico que possui quatro tópicos divididos da seguinte maneira:

- Noções Básicas de Lógica Clássica: apresenta um pouco das noções básicas sobre a Lógica Clássica e conceitos básicos da Lógica Proposicional Clássica.
- Lógica Modal: traz uma introdução sobre o estudo da Lógica Modal e seus operadores.
- Lógica Temporal: descreve o que é Lógica Temporal e como ela está classificada.
- NuSMV: será apresentada a ferramenta NuSMV.

No capítulo 3 por fim serão apresentadas as conclusões e trabalhos futuros referentes a este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

O trabalho apresentado é uma Introdução à Lógica Temporal através do uso da ferramenta NuSMV. Portanto é importante ressaltar que se trata de uma pesquisa bibliográfica. E os temas a serem abordados neste trabalho são: Lógica, Lógica Clássica ou Aristotélica, Lógica Proposicional Clássica, Lógica Modal e Lógica Temporal.

2.1 NOÇÕES BÁSICAS DE LÓGICA CLÁSSICA

2.1.1 Um pouco de história

A lógica está relacionada à capacidade de pensar ou raciocinar, e está relacionada a várias áreas do conhecimento humano e científico, considerada um instrumento para o estudo do raciocínio ou capacidade de dedução, podendo chegar a uma conclusão baseada em hipóteses, a lógica passou por várias evoluções, onde grandes filósofos, cientistas e estudiosos se dedicaram a entender e desenvolver a lógica como ciência até o que ela é hoje.

A Lógica Clássica, tem seu início a partir do período Aristotélico no século IV a.C., quando o filósofo grego Aristóteles (384-322 a.C.), com base no pensamento socrático-platônico, criou a lógica fundamentando-se nas definições universais de Sócrates, na redução ao absurdo de Zenão de Eléia, na estrutura proposicional e negação de Parmenides e Platão e nas técnicas argumentativas do raciocínio legal e provas geométricas (BEDREGAL, 2007).

As contribuições mais relevantes de Aristóteles para a lógica se encontram em sua obra *Órganon*, palavra que em grego significa “instrumento”, tal nome foi dado porque para Aristóteles a Lógica não era uma ciência nem prática e nem teórica, mas um instrumento para as ciências.

Aristóteles sustentava a ideia de que uma proposição é um complexo que envolve dois termos: sujeito e predicado, cada um com um substantivo representado gramaticalmente. Em suas pesquisas e teorias, Aristóteles procurava entender as relações entre duas proposições com os mesmos termos, e foi daí que surgiu o termo silogismo um dos primeiros sistemas dedutivos já propostos (BEDREGAL, 2007).

O silogismo é um regra de inferência que deduz uma proposição categórica a conclusão a partir de duas outras, chamadas premissas. Cada uma das premissas contém um termo comum com a conclusão o termo maior e o termo menor, respectivamente, e um termo comum com a outra premissa o termo médio (D'OTTAVIANO, 2003).

Como exposto acima, no silogismo temos três proposições declarativas, duas premissas e uma conclusão, sendo as duas premissas que apresentam exatamente um único termo que se conectam de maneira que permitem deduzir logicamente a terceira proposição

denominada conclusão. A teoria do silogismo é considerada pelos filósofos e historiadores da lógica como a descoberta mais importante em toda a história da Lógica formal por conter um dos primeiros sistemas axiomáticos construídos. Esta teoria de Aristóteles foi exposta em sua obra *Analíticos Anteriores* (D'OTTAVIANO, 2003)(BEDREGAL, 2007).

Um exemplo clássico de silogismo é:

- Todo homem é mortal \rightarrow Premissa maior (Contém o termo maior mortal e o termo médio homem)
- Sócrates é homem \rightarrow Premissa menor (Contém o termo menor Sócrates e o termo médio homem)
- Logo, Sócrates é mortal \rightarrow Conclusão (Contém o termo maior mortal e o termo menor Sócrates)

Segundo(BEDREGAL, 2007), a lógica Aristotélica passou por diversas modificações e foi desenvolvida de várias maneiras pelos alunos de Aristóteles, Eudemus de Rodes (350-290 a.C.) e Teofrastus de Lesbos (378-287 a.C.) que desenvolveram uma lógica que complementava a Aristotélica, alternativa ao silogismo, que era uma lógica completamente proposicional. Eles também estudaram várias antinomias, que são afirmações contraditórias de onde se tem os paradoxos.

A lógica pouco se desenvolveu antes da era moderna, onde ressurgiram estudos e pesquisas podendo citar o matemático alemão Leibniz (1646-1716), que pretendia desenvolver uma linguagem universal baseada em um alfabeto do pensamento, introduzindo em 1666 o projeto de construção de um sistema exato e universal de notação.

Para (MORTARI, 2001) e (CHAGAS, 2004), a Lógica Moderna teve um desenvolvimento mais significativo a partir da publicação em 1849 da obra, “Investigação sobre as leis do pensamento” de George Boole (1815-1864), onde se deu início a “simbolização” ou “matematização” da lógica. Boole apresentou um cálculo lógico, conhecido hoje como álgebra Booleana, contendo um número infinito de formas válidas de argumento, ao mesmo tempo em que Augustus De Morgan (1806-1871) também desenvolveu a Álgebra da Lógica. Neste mesmo período surge Gottlob Frege (1848-1925), que segundo Alonzo Church, foi “o maior lógico dos tempos modernos”, com sua obra *Begriffsschrift*, onde desenvolveu pela primeira vez de forma axiomática o Cálculo Sentencial, usando negação e implicação de forma primitiva, seis axiomas e regras de substituição.

O matemático e lógico Alemão Gottlob Frege, foi o responsável por desenvolver uma linguagem formal com representações matemáticas e adotar a noção de variáveis, quantificadores universal e existencial, relações e funções, Lógica Proposicional e axiomatizações da lógica. O princípio de tentar desenvolver a matemática como parte lógica foi o que motivou Frege, demonstrando seus princípios de indução a partir de seus princípios lógicos (BEDREGAL, 2007).

Os lógicos Alfred Whitehead (1861-1947) e Bertrand Russel (1872-1970) também deixaram as suas importantes contribuições para a lógica, eles publicaram juntos o livro *Principia Mathematica*, onde tentam mostrar a partir da visão de Frege, que a Matemática pode ser desenvolvida como parte da lógica, sendo sua axiomatização e formalismos referência para a lógica por mais de 20 anos. Além deles, o filósofo austríaco Ludwig Wittgenstein (1889-1951) também se inclui como contribuinte no desenvolvimento da lógica a partir do seu trabalho *Tractatus Logico-Philosophicus*, onde expôs pela primeira vez a Lógica Proposicional como ela é conhecida hoje como tabelas verdades, afirmando ainda que qualquer proposição é o resultado de sucessivas aplicações de operações lógicas sobre proposições elementares (BEDREGAL, 2007).

Partindo disto a lógica passou a ser desenvolvida aceleradamente, contando com dezenas de especialidades e subespecialidades hoje em dia, sendo considerada a lógica não mais uma parte da filosofia, mas como ciência independente (MORTARI, 2001).

Apesar de que inicialmente o objetivo da lógica fosse a análise de argumentos, o uso de linguagens ampliou seu contexto de atuação, surgindo novos usos para as linguagens da lógica. Como por exemplo, representar informações em geral por meio delas, notando-se o grande papel da lógica em investigações científicas, como na Inteligência Artificial, particularmente nas áreas de representação do conhecimento e demonstração automática (MORTARI, 2001).

2.1.2 Lógica Proposicional Clássica

Também conhecida como cálculo proposicional, a Lógica Proposicional é a forma mais simples de lógica e baseia-se na noção de proposições, e tem como finalidade a representação de argumentos, que contém conjuntos de sentenças onde existem uma conclusão e as demais são premissas, e a validação de argumentos, verificando se a conclusão é uma consequência lógica das premissas, ou seja, a Lógica Proposicional estuda como raciocinar e como deduzir a partir de um conjunto de hipóteses uma prova de que em um mesmo contexto uma conclusão seja verdadeira ou falsa.

A Lógica Proposicional contém uma linguagem formal para representação do conhecimento e métodos de inferência para representação do raciocínio.

Uma proposição é qualquer expressão declarativa que pode ser classificada como verdadeira ou falsa dependendo de sua interpretação, mas não podendo ser as duas coisas simultaneamente.

São exemplos de proposições:

1. O número 2 é par
2. O conjunto dos números reais é enumerável
3. Se a oferta de morango aumentar então o seu preço vai diminuir

Os exemplos a seguir não são proposições, pois não é possível definir um valor verdadeiro ou falso a partir das sentenças:

1. Qual é o seu nome?
2. Feche a porta!

Como descrito por (BEDREGAL, 2007) existem ainda algumas proposições que não podem ser quebradas em outras proposições mais simples, são chamadas de proposições atômicas ou minimais, elas não possuem nenhum conectivo lógico em sua expressão. Por exemplo: “Todos os corvos são pretos”, “Maria gosta de estudar lógica”.

São exemplos de proposições não atômicas: “Maria gosta de estudar lógica e teoria da computação”, “Se João não estuda para a prova, tirará nota baixa”, pois utilizam conectivos em suas expressões além do que podem ser quebradas para formar mais de uma proposição (BEDREGAL, 2007).

Devem ser considerados dois aspectos para o estudo da Lógica Proposicional, o sintático e o semântico. Enquanto a sintaxe permite identificar expressões bem formadas, a semântica está relacionada com o seu significado.

2.1.3 Sintaxe da Lógica Proposicional

Para representação simbólica da lógica, utiliza-se uma linguagem L composta por um alfabeto, constituído dos seguintes conjuntos:

- **Símbolos proposicionais:** identificadores que são representados por letras maiúsculas (P, Q, R, S, P1, P2...);
- **Símbolos verdade:** constantes True e False;
- **Conectivos Proposicionais** não (\neg), e (\wedge), ou (\vee), se então (\rightarrow), se somente se (\leftrightarrow);
- **Símbolos de pontuação:** parênteses (e);

Este conjunto quando combinado em fórmulas torna possível demonstrar sentenças complexas a partir de outras mais simples.

2.1.3.1 Conectivos Lógicos

É possível fazer a junção de duas ou mais proposições através de conectivos (não, e, ou, então, se e somente se) tornando-as proposições mais complexas. Por exemplo, “José gosta de chocolate e é inteligente” é uma proposição formada pela junção de duas proposições mais simples, “José gosta de chocolate” e “José é inteligente”.

Na Lógica Proposicional são usados os conectivos: negação, conjunção, disjunção, condicional ou implicação, bi-condicional ou bi-implicação.

Negação: cotidianamente a palavra “Não” é usada para expressar que algo não é verdade ou que não ocorre. Por exemplo: “José não gosta de chocolate” significa dizer que não é verdade que José gosta de chocolate. Logo, se uma proposição falsa for negada, esta proposição passa a ser verdadeira. Para representação do conectivo da negação na Lógica Proposicional é utilizado o símbolo \neg (BEDREGAL, 2007).

Conjunção: conhecida como conjunção, a palavra “e” é frequentemente usada no nosso vocabulário para conectar duas proposições. Tomando como exemplo as proposições “Maria gosta de João” e “João gosta de Maria” a conjunção destas duas proposições seria “Maria gosta de João e João gosta de Maria”, podendo ser ainda abreviada para “Maria e João se gostam”. Para representação do conectivo da conjunção na Lógica Proposicional é utilizado o símbolo \wedge . Portanto podemos obter de duas proposições, p_1 e p_2 , uma terceira proposição $p_1 \wedge p_2$, que será verdadeira se, e somente se p_1 e p_2 forem ambas verdadeiras (BEDREGAL, 2007).

Disjunção: conhecida como disjunção, também frequentemente usada no nosso vocabulário, a palavra “ou” é também utilizada para conectar duas proposições. Tomando como exemplo novamente as proposições “Maria gosta de João” e “João gosta de Maria”, a disjunção destas duas proposições seria “Maria gosta de João ou João gosta de Maria”. Para representação do conectivo da disjunção na Lógica Proposicional é utilizado o símbolo \vee . Portanto podemos obter de duas proposições, p_1 e p_2 , uma terceira proposição, $p_1 \vee p_2$, que será verdadeira se, e somente se p_1 ou p_2 for verdadeira, sendo ao menos uma das duas proposições verdadeira para que a disjunção delas também seja verdadeira (BEDREGAL, 2007).

Condicional ou Implicação: utilizada no português a expressão “se... então...”, é usada para conectar duas proposições. Por exemplo, “Maria e João são irmãos” e “João e Maria são parentes”, a condicional para estas proposições seria “Se Maria e João são irmãos então João e Maria são parentes”. Para representar o conectivo da Condicional ou Implicação na Lógica Proposicional é utilizado o símbolo \rightarrow . Portanto podemos obter de duas proposições, p_1 e p_2 , uma terceira proposição, $p_1 \rightarrow p_2$, que será verdadeira somente se a primeira proposição p_1 for verdadeira. É chamada de premissa a proposição que vem antes do conectivo \rightarrow , e chamada de conclusão a premissa que vem depois do conectivo \rightarrow (BEDREGAL, 2007).

Bicondicional ou Bi-Implicação: utilizada no português a expressão “se, e somente se,” é usada para conectar duas proposições. Por exemplo, “Maria vai passar de ano se, e somente se, tirar notas boas” é uma Bicondicional de “Maria vai passar de ano” com “Maria vai tirar notas boas”. Para representar o conectivo da Bicondicional ou Bi-Implicação na Lógica Proposicional é utilizado o símbolo \leftrightarrow . Portanto podemos obter de duas proposições, p_1 e p_2 , uma terceira proposição, $p_1 \leftrightarrow p_2$, que será verdadeira se, e

somente se, ambas tiverem valores iguais, ou seja, se as duas proposições tiverem o mesmo valor verdade (BEDREGAL, 2007).

2.1.3.2 Fórmulas Proposicionais

Souza (2008) afirma que as fórmulas da linguagem da Lógica Proposicional são construídas de forma indutiva a partir dos símbolos do alfabeto proposicional de acordo com as seguintes regras:

1. Todo símbolo verdade é uma fórmula;
2. Todo símbolo proposicional é uma fórmula;
3. Se P é uma fórmula, então a sua negação ($\neg P$) também é uma fórmula;
4. Se P e Q são fórmulas:
 - a) A Conjunção de P e Q ($P \wedge Q$), também é fórmula;
 - b) A Disjunção de P e Q ($P \vee Q$), também é fórmula;
 - c) A Implicação de P em Q ($P \rightarrow Q$), também é fórmula;
 - d) A Bi-implicação de P e Q ($P \leftrightarrow Q$), também é fórmula;

A partir das regras citadas anteriormente, os símbolos, P , Q e True são fórmulas. Levando em conta isso, podemos obter a partir das fórmulas P e Q a fórmula $(P \vee Q)$. E a partir da fórmula $(P \vee Q)$ e True , obtemos a fórmula $((P \vee Q) \rightarrow \text{True})$ (SOUZA, 2008).

"Esse raciocínio pode ser repetido, podendo ser obtidas infinitas fórmulas" (SOUZA, 2008).

As fórmulas a seguir, forms não são consideradas fórmulas proposicionais, pois não se pode derivá-las a partir das regras descritas anteriormente:

$$\mathbf{PQR} \text{ (1)}$$

$$\mathbf{(R \ True \rightarrow)} \text{ (2)}$$

$$\mathbf{(\text{False} \wedge \vee (\leftrightarrow \text{QP}))} \text{ (3)}$$

2.1.3.3 Precedência de Conectivos

Assim como na aritmética, existe uma ordem de precedência entre os conectivos proposicionais, o que permite a eliminação dos parênteses nas fórmulas quando estes não alteram o significado da fórmula proposicional, como exemplo a fórmula a seguir:

$$((\neg(\neg P)) \rightarrow Q) \equiv \neg \neg P \rightarrow Q \text{ (4)}$$

Na Lógica proposicional, em fórmulas em que não são utilizados os símbolos de pontuação (parênteses), a ordem de precedência dos conectivos é definida a seguinte maneira:

- \neg : maior precedência
- $\rightarrow, \leftrightarrow$: menor precedência
- \vee, \wedge : precedência intermediária

Para definir a prioridade no cálculo proposicional para conectivos de mesma precedência, segundo (MARTINS, 2018), existem as regras de associatividade, onde os conectivos \vee e \wedge são associativos à esquerda, enquanto que os conectivos \rightarrow e \leftrightarrow são associativos à direita.

$$\mathbf{P \vee Q \wedge R \equiv (P \vee Q) \wedge R} \quad (5)$$

$$\mathbf{P \rightarrow Q \leftrightarrow R \equiv P \rightarrow (Q \leftrightarrow R)} \quad (6)$$

2.1.4 Semântica da Lógica Proposicional

Dependendo dos valores atribuídos às variáveis proposicionais, as fórmulas podem ser verdadeiras ou falsas. É o que chamamos de semântica que associa um significado ou interpretação a cada objeto sintático. Este significado é determinado por uma função binária total.

De acordo com (SOUZA, 2008), a função interpretação associa cada fórmula da Lógica Proposicional a um valor de verdade "Verdadeiro" ou "Falso", representado por T e F. Desse modo o resultado de toda proposição só poderá assumir um valor verdade, princípio denominado de princípio da bivalência, onde estabelece que uma proposição será necessariamente verdadeira ou falsa.

Utilizando o conceito de função binária, temos a seguir uma definição mais formal sobre interpretação:

- Função Binária: quando o contradomínio da função possui apenas 2 elementos.
- Função Total: quando a função é definida em todos os elementos de seu domínio.
- Função Interpretação: uma interpretação I, na lógica proposicional é uma função binária em que:
 - O domínio de I é constituído pelo conjunto das fórmulas da lógica proposicional.
 - O contradomínio de I é o conjunto $\{T, F\}$.

2.1.5 Valores de Verdade

Como já mencionado, um valor verdade, é um valor que indica o grau de verdade de uma proposição dependendo da sua interpretação, podendo ser um valor Verdadeiro ou Falso. Para exemplificar melhor esses valores verdade, serão apresentados a seguir cada caso dos operadores lógicos.

2.1.5.1 Negação

Para representar a propriedade do conectivo da negação, podemos resumir na seguinte figura, onde na primeira coluna está uma fórmula α qualquer, que pode ter valor verdadeiro ou falso, e na segunda coluna está o valor correspondente a negação de α (MORTARI, 2001).

Figura 1 – Tabela da Negação

| α | $\neg \alpha$ |
|----------|---------------|
| V | F |
| F | V |

Fonte: Autoria própria.

2.1.5.2 Conjunção

Uma Conjunção $\alpha \wedge \beta$ expressa que as duas fórmulas, α e β , devem ser verdadeiras e caso uma delas seja falsa, então a conjunção delas será falsa, como resumido na figura a seguir (MORTARI, 2001).

Figura 2 – Tabela da Conjunção

| α | β | $\alpha \wedge \beta$ |
|----------|---------|-----------------------|
| V | V | V |
| F | V | F |
| V | F | F |
| F | F | F |

Fonte: Autoria própria.

2.1.5.3 Disjunção

Uma Disjunção de $\alpha \vee \beta$ expressa que o resultado do cálculo proposicional das duas fórmulas α e β , deve ser verdadeiro se uma das fórmulas for verdadeira ou se ambas forem verdadeiras, como representado na figura (MORTARI, 2001).

Figura 3 – Tabela da Disjunção

| α | β | $\alpha \vee \beta$ |
|----------|---------|---------------------|
| V | V | V |
| F | V | V |
| V | F | V |
| F | F | F |

Fonte: Autoria própria.

2.1.5.4 Implicação

Para que a implicação de duas fórmulas seja verdadeira, se os valores das duas fórmulas forem iguais: V e V ou F e F, ela será verdadeira, caso os valores sejam diferentes e o primeiro termo tiver valor falso a implicação será verdadeira, como expresso na figura a seguir (MORTARI, 2001).

Figura 4 – Tabela da Implicação

| α | β | $\alpha \rightarrow \beta$ |
|----------|---------|----------------------------|
| V | V | V |
| F | V | V |
| V | F | F |
| F | F | V |

Fonte: Autoria própria.

2.1.5.5 Bi-Implicação

Para que a implicação de duas fórmulas seja verdadeira, o valor das duas devem ser equivalentes, caso contrário a implicação será falsa, como mostrado na figura (MORTARI, 2001).

Figura 5 – Tabela da Bi-Implicação

| α | β | $\alpha \leftrightarrow \beta$ |
|----------|---------|--------------------------------|
| V | V | V |
| F | V | F |
| V | F | F |
| F | F | V |

Fonte: Autoria própria.

2.1.6 Propriedades Semânticas da Lógica Proposicional

Para (SOUZA, 2008) as propriedades semânticas são relações entre os resultados das interpretações das fórmulas, que são obtidas no mundo semântico, porém a partir de fórmulas sintáticas.

2.1.6.1 Tautologia

Uma tautologia é toda proposição composta em que a última coluna da sua tabela-verdade termina somente com o símbolo V(verdade), ou ainda, tautologia é toda proposição composta, $P(\alpha, \beta, \gamma \dots)$ em que seu valor lógico é sempre V(verdade) (FILHO, 2002).

- Definição: "H é uma Tautologia, se, e somente se, para toda interpretação I, $I[H] = T$ (SOUZA, 2008)".

O princípio do terceiro excluído: "Dizer que uma proposição ou é verdadeira ou é falsa é sempre verdadeiro (FILHO, 2002)".

Ex: A proposição $\alpha \vee \neg \alpha$

Figura 6 – Representação de Tautologia pelo princípio do terceiro excluído

| α | $\neg \alpha$ | $\alpha \vee \neg \alpha$ |
|----------|---------------|---------------------------|
| V | F | V |
| F | V | V |

Fonte: Autoria própria.

2.1.6.2 Satisfatibilidade

Uma fórmula é satisfatível quando existe pelo menos uma interpretação I, em que seu valor é igual a V. Quando uma fórmula não é satisfatível dizemos então, que ela é insatisfatível (SOUZA, 2008).

- "Definição: H é satisfatível, se, e somente se, existe uma interpretação I, tal que $I[H] = V$ (SOUZA, 2008)".

Ex: A proposição $\alpha \vee \beta$

Figura 7 – Exemplo de satisfatibilidade

| α | β | $\alpha \vee \beta$ |
|----------|---------|---------------------|
| V | F | V |
| F | V | V |
| V | F | V |
| F | V | F |

Fonte: Autoria própria.

2.1.6.3 Contradição

Uma contradição é toda proposição composta em que a última coluna da sua tabela-verdade termina somente com o valor F(falsidade). Em outras palavras é toda proposição composta $P(\alpha, \beta, \gamma, \dots)$ em que seu valor lógico é sempre F(falso). Uma contradição é o contrário de uma tautologia, pois uma tautologia é sempre verdadeira e sua negação é sempre falsa e vice-versa (FILHO, 2002).

- Definição: "H é contraditória, se, e somente se, para toda interpretação I, $I[H] = F$ (SOUZA, 2008)".

Ex: a proposição $\alpha \wedge \neg \alpha$

Figura 8 – Exemplo de contradição

| α | $\neg \alpha$ | $\alpha \wedge \neg \alpha$ |
|----------|---------------|-----------------------------|
| V | F | F |
| F | V | F |

Fonte: Autoria própria.

O princípio da não-contradição: "Dizer que uma proposição não poder ser simultaneamente verdadeira e falsa é sempre verdadeiro (FILHO, 2002)".

Ex: A proposição $\neg(\alpha \wedge \neg \alpha)$

Figura 9 – Representação de Tautologia pelo princípio da não-contradição

| α | $\neg \alpha$ | $\alpha \wedge \neg \alpha$ | $\neg(\alpha \wedge \neg \alpha)$ |
|----------|---------------|-----------------------------|-----------------------------------|
| V | F | F | V |
| F | V | F | V |

Fonte: Autoria própria.

2.1.6.4 Contingência

Toda proposição composta em que a sua última coluna na tabela-verdade existem os símbolos V e F cada um pelo menos uma vez é chamada de contingência. Em outras palavras, uma contingência é toda proposição composta que não é nem tautologia nem contradição (FILHO, 2002).

- Definição: "H é contingência, se, e somente se, existem duas interpretações I1 e I2, tais que $I1[H] = V$ e $I2[H] = F$ (SOUZA, 2008)".

Ex: A proposição $\alpha \rightarrow \neg\alpha$

Figura 10 – Exemplo de contingência

| α | $\neg\alpha$ | $\alpha \rightarrow \neg\alpha$ |
|----------|--------------|---------------------------------|
| V | F | F |
| F | V | V |

Fonte: Autoria própria.

2.1.6.5 Equivalência Semântica

Sejam H e G, duas fórmulas proposicionais, elas são equivalente semanticamente, se e somente se para toda interpretação I, $I[H] = I[G]$. O símbolo da equivalência pode ser representado por \Leftrightarrow ou \equiv .

Considerando as fórmulas $H = (\neg\alpha \wedge \neg\beta)$ e $G = \neg(\alpha \vee \beta)$, a seguir será demonstrado que elas são equivalentes.

Dada uma interpretação I,

$$\begin{aligned}
 \text{Para } I[H] = T &\Leftrightarrow I[\neg\alpha \wedge \neg\beta] = T \\
 &\Leftrightarrow I[\neg\alpha] = T \text{ e } I[\neg\beta] = T \\
 &\Leftrightarrow I[\alpha] = F \text{ e } I[\beta] = F \\
 &\Leftrightarrow I[\alpha \vee \beta] = F \\
 &\Leftrightarrow I[\neg(\alpha \vee \beta)] = T \\
 &\Leftrightarrow I[G] = T.
 \end{aligned}$$

$$\begin{aligned}
 \text{Para } I[H] = F &\Leftrightarrow I[\neg\alpha \wedge \neg\beta] = F \\
 &\Leftrightarrow I[\neg\alpha] = F \text{ ou } I[\neg\beta] = F \\
 &\Leftrightarrow I[\alpha] = T \text{ ou } I[\beta] = T \\
 &\Leftrightarrow I[\alpha \vee \beta] = T \\
 &\Leftrightarrow I[\neg(\alpha \vee \beta)] = F \\
 &\Leftrightarrow I[G] = F.
 \end{aligned}$$

Logo, H e G apresentam o mesmo valor para toda interpretação, portanto elas são equivalentes (SOUZA, 2008).

2.1.6.6 Implicação Semântica

Segundo (MORTARI, 2001), uma fórmula α implica semanticamente uma fórmula β se, para toda valoração v tal que $v(\alpha) = V$, temos que $v(\beta) = V$.

Para (FILHO, 2002), uma proposição α implica uma proposição β se β é verdadeira sempre que α foi verdadeira. O símbolo usado para representar a implicação semântica é o \rightarrow (implica).

Considerando as proposições $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \leftrightarrow \beta$, a tabela-verdade para essas proposições é:

Figura 11 – Tabela da implicação semântica

| α | β | $\alpha \wedge \beta$ | $\alpha \vee \beta$ | $\alpha \rightarrow \beta$ |
|----------|---------|-----------------------|---------------------|----------------------------|
| V | V | V | V | V |
| V | F | F | V | F |
| F | V | F | V | F |
| F | F | F | F | V |

Fonte: Autoria própria.

A partir da tabela podemos observar que a proposição " $\alpha \wedge \beta$ " tem valor V apenas na primeira linha da tabela, e nesta mesma linha, as proposições " $\alpha \vee \beta$ " e " $\alpha \leftrightarrow \beta$ " também possuem valor V. Logo a proposição " $\alpha \wedge \beta$ " implica as outras duas proposições, ou seja: $\alpha \wedge \beta \rightarrow \alpha \vee \beta$ e $\alpha \wedge \beta \rightarrow \alpha \leftrightarrow \beta$

2.1.7 Métodos de Validação de Argumentos

Para verificação da validade de fórmulas proposicionais, são utilizados alguns métodos equivalentes em muitos aspectos, são eles, tabela-verdade, árvore semântica e negação, ou redução ao absurdo.

2.1.7.1 Tabela-Verdade

É possível representar as regras semânticas e mapear todas as possíveis combinações dos símbolos proposicionais a partir da tabela-verdade, considerada um método da força bruta (SOUZA, 2008).

Dada uma fórmula proposicional, é possível construir uma tabela-verdade que corresponde a qualquer proposição composta, onde mostrará os casos em que a proposição composta será verdadeira (V) ou falsa (F), sabendo que seu valor lógico depende dos valores lógicos de suas proposições simples (FILHO, 2002).

O número de linhas da tabela-verdade vai depender da quantidade de proposições simples que integram a fórmula proposicional, ou seja, a tabela-verdade de uma proposição composta com n proposições simples contém 2^n linhas (FILHO, 2002).

Considerando uma interpretação I , onde : $I[P] = V$ e $I[Q] = F$, podemos obter todos os possíveis valores verdade desta interpretação com o auxílio da tabela-verdade, onde na primeira linha da tabela são escritos os símbolos proposicionais combinados com cada um dos conectivos, a seguir nas duas primeiras colunas são escritos os possíveis valores verdade associados aos símbolos e assim nas colunas seguintes os valores verdade obtidos em cada combinação de valores verdade e conectivos, como mostrado anteriormente na seção 2.1.5, em que se tem a tabela verdade associada a cada conectivo lógico separadamente.

Figura 12 – Tabela verdade

| P | Q | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \rightarrow Q$ | $P \leftrightarrow Q$ |
|---|---|----------|--------------|------------|-------------------|-----------------------|
| V | V | F | V | V | V | V |
| V | F | F | F | V | F | F |
| F | V | V | F | V | V | F |
| F | F | V | F | F | V | V |

Fonte: Autoria própria.

$$I[\neg P] = F, I[\neg Q] = T,$$

$$I[P \wedge Q] = F,$$

$$I[P \vee Q] = V,$$

$$I[P \rightarrow Q] = F,$$

$$I[P \leftrightarrow Q] = F.$$

Pode-se notar que os valores verdade obtidos correspondem com a segunda linha da tabela-verdade na Figura 12.

Tomando como exemplo a seguinte proposição: $\neg(P \wedge \neg Q)$ a construção de sua tabela-verdade é mostrada na figura 13.

Figura 13 – Tabela verdade para a proposição $\neg(P \wedge \neg Q)$

| P | Q | $\neg Q$ | $P \wedge \neg Q$ | $\neg(P \wedge \neg Q)$ |
|---|---|----------|-------------------|-------------------------|
| V | V | F | F | V |
| V | F | V | V | F |
| F | V | F | F | V |
| F | F | V | F | V |

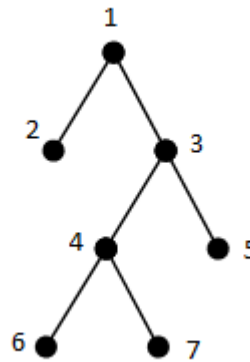
Fonte: Autoria própria.

Para fórmulas que contém muitos símbolos proposicionais, o método da tabela-verdade não é adequado, pois a tabela poderá ficar muito grande para ser elaborada manualmente, como por exemplo uma fórmula que contém 8 símbolos proposicionais, o número de linhas da sua tabela seria 256 (SOUZA, 2008).

2.1.7.2 Árvore Semântica

De acordo com (SOUZA, 2008) a árvore semântica é um método que determina a validade de uma fórmula e suas propriedades semânticas a partir de uma estrutura de dados chamada árvore. Uma árvore é um conjunto de nós ou vértices ligados por arestas, onde cada nó é numerado. Como no exemplo a seguir, onde o nó 1 é a raiz da árvore e os nós finais 2, 6, 7 e 5 são chamados folhas.

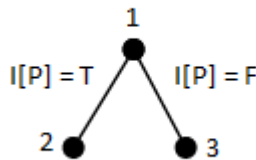
Figura 14 – Exemplo de árvore semântica



Fonte: Autoria própria.

Para demonstrar que a fórmula $H = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P))$ é uma tautologia, faremos a sua árvore semântica. Inicialmente a árvore semântica é desenhada a partir da raiz, onde suas arestas estão denominadas por $I[P] = T$ e $I[P] = F$, que são as duas possibilidades de valor verdade para P (SOUZA, 2008).

Figura 15 – Criação da árvore semântica- Etapa 1



Fonte: Autoria própria.

O nó 2 corresponde a fórmula com os significados de P escrito abaixo do seu símbolo.

$$\begin{array}{ccc} \text{Nó 2} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) & & \\ T & & T \end{array}$$

A partir daí podemos obter o significado de $(\neg P)$.

$$\begin{array}{c} \text{Nó 2} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) \\ \text{T} \qquad \qquad \qquad \text{F T} \end{array}$$

E no nó 3 onde $I[P]=F$, o significado da fórmula ficaria assim:

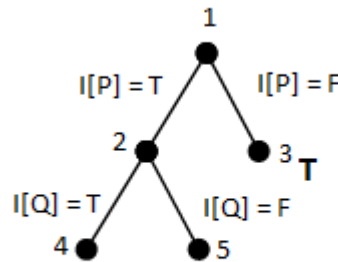
$$\begin{array}{c} \text{Nó 3} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) \\ \text{F} \qquad \qquad \qquad \text{T F} \end{array}$$

Podendo observar que a partir das leis da semântica apenas com o significado de $I[P]=F$, obtemos que o resultado da fórmula é T, pois a implicação de F em qualquer coisa é T e qualquer coisa implicando em T será T independente do valor de Q (SOUZA, 2008).

$$\begin{array}{c} \text{Nó 3} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) \\ \text{F T} \quad \text{T} \quad \text{T T F} \end{array}$$

Estendendo a árvore considerando os valores de $I[Q]$.

Figura 16 – Criação da árvore semântica- Etapa 2



Fonte: Autoria própria.

$$\begin{array}{c} \text{Nó 4} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) \\ \text{T T T} \quad \text{T} \quad \text{F T T} \quad \text{F T} \end{array}$$

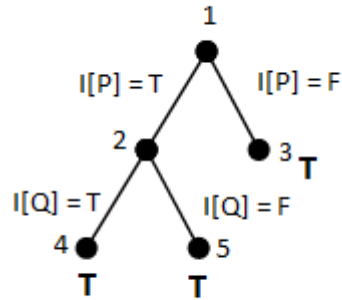
Com todas as interpretações já determinadas, o símbolo T abaixo do conectivo \leftrightarrow significa que a fórmula é verdadeira quando P e Q são verdadeiras. Esse símbolo é então escrito abaixo do nó 4 na árvore (SOUZA, 2008).

Seguindo isto, o nó 5 corresponde à fórmula:

$$\begin{array}{c} \text{Nó 5} = (P \rightarrow Q) \leftrightarrow ((\neg Q) \rightarrow (\neg P)) \\ \text{T F F} \quad \text{T} \quad \text{T F F} \quad \text{F T} \end{array}$$

Novamente, pode-se ver que o símbolo T é escrito abaixo do conectivo \leftrightarrow que será escrito abaixo do nó na árvore. Concluindo que a fórmula é uma tautologia, pois em todas as folhas da árvore aparece o símbolo T. Se as folhas da árvore resultassem todas em F, então a fórmula seria uma contradição e se pelo menos uma folha obtivesse o valor T, então a fórmula seria satisfatível (SOUZA, 2008).

Figura 17 – Criação da árvore semântica- Etapa 3



Fonte: Autoria própria.

2.1.7.3 Negação ou Redução ao Absurdo

Conforme (SOUZA, 2008), o método da negação ou redução ao absurdo, também conhecido como método de refutação, é um método geral de demonstração para demonstrar propriedades semânticas, sendo considerada inicialmente a negação daquilo que se quer demonstrar, e a partir de um conjunto de deduções concluir um fato contraditório ou absurdo. Ou seja, se quisermos demonstrar que H é uma tautologia, devemos supor que H não é uma tautologia, e a partir de uma sequência de deduções, conclui-se um absurdo, então dizer que H não é uma tautologia é um absurdo, concluindo que H é uma tautologia.

Será usado como exemplo a lei da transitividade para o conectivo \rightarrow , para demonstrar que a seguinte fórmula é uma tautologia:

$$H = ((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

Segundo o método da negação ao absurdo, para demonstrar que H é uma tautologia alguns passos devem ser seguidos. Primeiro supondo que H não é uma tautologia. Logo podemos interpretar que H é falsa em pelo menos uma interpretação. Isso é denotado inserindo o símbolo F logo abaixo do conectivo \rightarrow como mostrado a seguir (SOUZA, 2008).

$$H = ((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

F

formal de argumentos filosóficos a compreensão da Lógica Modal é muito importante onde expressões modais podem ser comuns e confusas (MELO DENIS FELIPE, 2009).

A Lógica Modal, desde o início do século XX vem sendo estudada formalmente, tendo início em 1918 com os trabalhos de Lewis, sendo introduzidos vários sistemas modais que são utilizados até hoje, porém Gödel, em 1933, apresentou a Lógica Modal como conhecemos hoje, como uma extensão da Lógica Proposicional clássica. Vários trabalhos importantes foram feitos em seguida até chegar ao que chamamos de semântica de mundos possíveis. Em 1952, Jónsson e Tarski, introduziram álgebras booleanas com operadores e um teorema de representação que indicava como construir modelos numa nova semântica (BENEVIDES, 2016).

Para (BENEVIDES, 2016), a semântica dos mundos possíveis impulsionou bastante a área da Lógica Modal. A partir dos trabalhos de Hintikka (1962) e Kripke (1959, 1962, 1963), que propunha uma semântica relacional, a Lógica Modal teve um grande desenvolvimento se tornando um dos ramos mais abundantes em Lógica Matemática e em lógicas para computação, ocorrendo grandes avanços sintáticos e semânticos, algébricos e de complexidade. Surgindo assim várias outras lógicas com base em Lógica Modal, como a Lógica Temporal, Lógica Epistêmica, Lógica Dinâmica entre outras.

O termo “alética”, usado na Lógica Modal, deriva da palavra grega “aleteia” que significa verdade. Portanto falar sobre modalidade aléticas é falar dos modos como uma frase pode ser verdadeira ou falsa, como por exemplo, necessidade e possibilidade (BERTOLINI; CUNHA; FORTES, 2017).

Na Lógica Modal, as proposições podem ser classificadas da seguinte maneira:

- Necessárias: são proposições que são necessariamente verdadeiras ou falsas, sendo impossível sua negação. Por exemplo: “ $1 + 1 = 2$ ”.
- Possíveis: são proposições que não são necessariamente falsas podendo levar a uma ocorrência. Por exemplo: “Pode estar chovendo agora em São Paulo”.
- Contingentes: são proposições que podem ou não ser verdadeiras. Por exemplo: “José era um filósofo”.
- Impossíveis: são proposições que marcam a impossibilidade de um acontecimento. Por exemplo: “Uma pedra tem emoções”.

2.2.1 Sintaxe e Semântica

Baseada na semântica dos "mundos possíveis", para (FAJARDO, 2004) a Lógica Modal analisa se uma proposição é verdadeira ou não dependendo do "mundo" em que ela está sendo analisada.

A sintaxe e a semântica da Lógica Modal é derivada da Lógica de primeira ordem. Sua sintaxe não é tão ampla quanto a sintaxe da Lógica Proposicional Clássica, sendo apenas necessário que adicionemos o restante dos conectivos (RISSINO; TORRER; MARTINS, 2007).

2.2.1.1 Sintaxe

O alfabeto modal é constituído pelos símbolos da Lógica Proposicional Clássica, como fórmulas atômicas, p, q, r, \dots , conectivos binários de conjunção \wedge , disjunção \vee , implicação \rightarrow , bi-implicação \leftrightarrow e o conectivo unário da negação \neg , além dos delimitadores "("e ")", e o símbolo do absurdo ou contradição \perp , todos estes adicionados juntamente com os operadores modais unários de necessidade ou necessariamente \Box e possibilidade ou possivelmente \Diamond (BENEVIDES, 2016) (FAJARDO, 2004).

A linguagem Modal a partir do alfabeto modal pode ser definido do seguinte modo:

$$\alpha ::= p \mid \perp \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \rightarrow \alpha_2 \mid \neg \alpha \mid \Box \alpha \mid \Diamond \alpha$$

Para (ALMEIDA, 2011) os operadores \Box e \Diamond são interdefiníveis da seguinte maneira:

$$\Box \alpha \equiv \neg \Diamond \neg \alpha$$

$$\Diamond \alpha \equiv \neg \Box \neg \alpha$$

Desse modo podemos adotar um dos conectivos modais como primitivo. Usualmente será adotado o conectivo \Box .

- Se α é um fórmula bem formada, então $\Box \alpha$ e $\neg \alpha$ também são fórmulas bem formadas.
- $\Diamond \alpha$ abrevia $\neg \Box \neg \alpha$
- Se α e β são fórmulas bem formadas, então $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$, $(\alpha \leftrightarrow \beta)$ também são fórmulas bem formadas, podendo ser dispensados os parênteses externos.
- $\Box \alpha$ é interpretado como "é necessário que α ", e $\Diamond \alpha$ como "é possível que α ".

Considerando como exemplo $\alpha =$ Beber água, temos que $\Diamond \alpha \equiv \neg \Box \neg \alpha$, que diz que é possível beber água equivale a não é necessário não beber água, e $\Box \alpha \equiv \neg \Diamond \neg \alpha$, que diz que é necessário beber água equivale a não é possível não beber água.

2.2.1.2 Regras e Axiomas

Um conjunto de regras de inferência e axiomas para uma linguagem lógica é chamado de sistema de axiomas. Os axiomas são fórmulas e as regras de inferência são as relações entre as fórmulas.

Um sistema de axiomas para a lógica modal deve conter pelo menos os seguintes axiomas e regras de inferência:

- Tautologias da lógica proposicional
- Axioma K, também conhecido como axioma da distributividade: $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$
- Substituição Uniforme: seja A um teorema, p uma fórmula atômica e B uma fórmula, substituindo todas as ocorrências de p em A por B se obtém uma fórmula que é um teorema.
- Modus Ponens: se $\vdash A$ e $\vdash A \rightarrow B$ então $\vdash B$, ou seja, se A e $A \rightarrow B$ são teoremas, então B também é teorema.
- Regra da Necessitação: se $\vdash A$ então $\vdash \Box A$. Se A é um teorema, então $\Box A$ é um teorema. Essa regra traduz a ideia inicial que queremos para a palavra 'necessário' sendo adotada em qualquer sistema de Lógica Modal (COSCARELLI, 2008).

A junção de todos os axiomas e regras de inferência citados acima é o que forma o chamado sistema mínimo K da Lógica Modal.

Na Lógica Modal também são utilizados outros axiomas para construção de sistemas, entre eles estão:

- Axioma da Reflexividade ou axioma T
 $\Box p \rightarrow p$
- Axioma da Simetria ou axioma B
 $p \rightarrow \Box \Diamond p$
- Axioma da Transitividade ou axioma 4
 $\Box p \rightarrow \Box \Box p$
- Axioma da Euclidianidade ou axioma 5
 $\Diamond p \rightarrow \Box \Diamond p$
- Axioma da Serialidade ou axioma D
 $\Box p \rightarrow \Diamond p$
- Axioma da Unicidade ou axioma CD
 $\Diamond p \rightarrow \Box p$
- Simula Convergência ou axioma X
 $\Diamond \Box p \rightarrow \Box \Diamond p$
- Simula Densidade ou axioma 2
 $\Box \Box p \rightarrow \Box p$

- Axioma de Gödel-Lob ou axioma GL

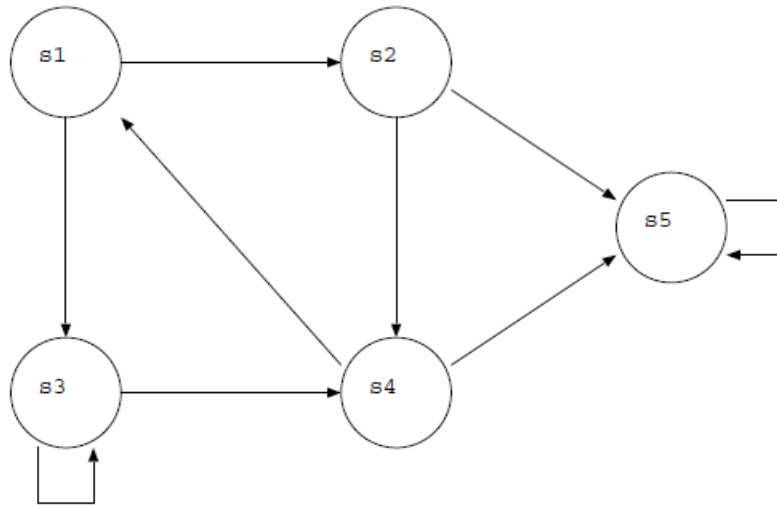
$$\Box(\Box p \rightarrow p) \rightarrow \Box p$$

2.2.1.3 Semântica

A Semântica da Lógica Modal baseia-se no modelo de Kripke, que é uma estrutura que contém um par $\langle W, R \rangle$, onde W é um conjunto não-vazio, denominado conjunto dos “mundos possíveis”, e R é uma relação binária sobre W , ou seja, $R \subset W \times W$ (FAJARDO, 2004).

Para (BENEVIDES, 2016) e (COSCARELLI, 2008) uma Estrutura também chamada de Frame é um par $F = \{W, R\}$, onde W é um conjunto de estados ou pontos e R é uma relação binária ou conjunto de pares (X, Y) , pertencente a W , dita relação de acessibilidade. Podemos dizer que $S2 \in W$ é acessível a partir de $S1 \in W$ se, e somente se, $(s1, s2) \in R$, ou seja se eles se relacionam. Na ilustração a seguir temos um exemplo de frame:

Figura 18 – Exemplo de Frame



Fonte: (BENEVIDES, 2016)

No exemplo acima, o conjunto de estados é $W = \{S1, S2, S3, S4, S5\}$ e a relação de acessibilidade entre eles é dada por $R = (S1, S2), (S1, S3), (S3, S3), (S3, S4), (S2, S4), (S2, S5), (S4, S1), (S4, S5), (S5, S5)$. O Frame é $F = (W, R)$.

Um modelo é um par $M = (F, V)$, onde $F = (W, R)$ é um Frame e V é uma função no conjunto das partes de W que corresponde a todo símbolo proposicional p . $V(p)$ corresponde ao conjunto de estados em que p é satisfeito (BENEVIDES, 2016).

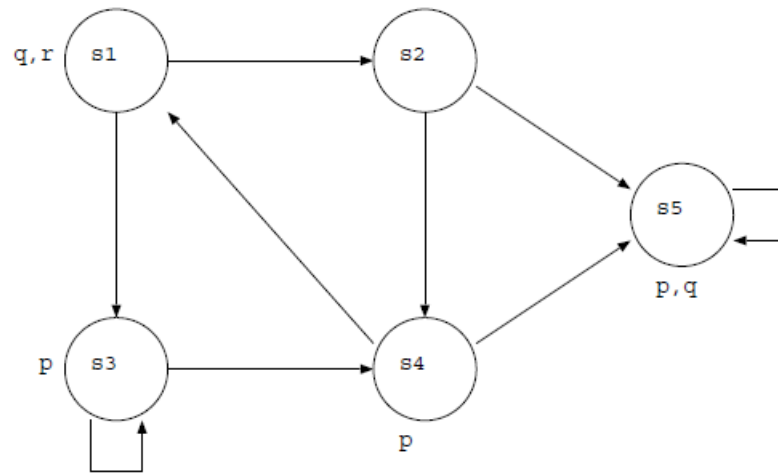
Tomando como exemplo o frame da figura anterior, a função V é dada a seguir:

- $V(p) = \{S3, S4, S5\}$

- $V(q) = \{S1, S5\}$
- $V(r) = \{S1\}$

A figura a seguir é um modelo a partir do frame que foi dado como exemplo:

Figura 19 – Modelo de Frame para a função V



Fonte: (BENEVIDES, 2016)

2.2.2 Tipos de Lógicas Modais

Ao longo dos anos a Lógica Modal adquiriu outras utilidades além de indicar somente possibilidade e necessidade, surgindo então novas modalidades dentro da Lógica Modal que indicam mais modos dependendo da situação, são elas, Lógica Epistêmica, Lógica Deôntica, Lógica Doxástica, Lógica Dinâmica, Lógica Temporal entre outras. Falaremos um pouco sobre algumas dessas lógicas antes de abordar sobre Lógica Temporal.

2.2.2.1 Lógica Epistêmica

A Lógica Epistêmica é um tipo de Lógica Modal ligada ao raciocínio sobre o conhecimento, tendo como objetivo a formalização de crenças e conhecimentos, de modo que afirma que "x acredita que y" e "x sabe que y". Desenvolvida em aplicações de diversos campos como Filosofia, Ciência da Computação, Inteligência Artificial, Linguística entre outros, a modelagem de conhecimento da Lógica Epistêmica baseia-se no modelo de mundo possíveis (MELO DENIS FELIPE, 2009).

2.2.2.2 Lógica Deôntica

Como descrito por (GOMES, 2008), a Lógica Deôntica está ligada ao estudo da validade de argumentos com expressões do tipo "É obrigatório que..." e "É permitido

que...", sendo as duas expressões representadas respectivamente pela letra maiúscula O (operador de Obrigação), e pela letra maiúscula P (Operador de Permissão). Tomando como exemplo a frase "e-mails são respondidos", Op deve ser lido como "É obrigatório que e-mails sejam respondidos" e Pp como "É permitido que e-mails sejam respondidos".

Entendida como a lógica das normas, a Lógica Deôntica vem da palavra grega *deón* que está relacionada com necessidade e o que é preciso, no sentido de permissão e obrigação. Enquanto os operadores modais L e M de necessidade e possibilidade são chamados de aléticos, O e P são os operadores deônticos. Por definição Pp (é permitido p), é equivalente a $\neg O\neg p$ (Não é obrigatório não p) e Op (É obrigatório p) é equivalente a $\neg P\neg p$ (Não é permitido não p) (GOMES, 2008).

As Lógicas Deônticas não possuem o axioma T (axioma da reflexividade), que seria correspondente dizer que toda obrigação fosse verdadeira quando sabemos que nem todas as obrigações são realmente cumpridas, o que não valeria a expressão $Op \rightarrow p$, pois diz que o que é obrigatório é verdadeiro, ao mesmo tempo em que a expressão $p \rightarrow Pp$ também não valeria pois segundo esta, o que é verdadeiro é permitido, o que não seria verdade se tomarmos como exemplo a frase "João matou José", esta frase seria verdadeira mas não quer dizer que João teria permissão para matar José (GOMES, 2008).

2.2.2.3 Lógica Doxástica

A Lógica Doxástica é um tipo de Lógica Modal que trata de raciocínio sobre as crenças, o termo doxástica vem do grego onde *doxa* quer dizer crença. Na Lógica Doxástica normalmente é utilizado Bx para acredita-se que x é verdadeiro, e o conjunto β como conjunto das crenças (MELO DENIS FELIPE, 2009).

$\beta : b1, b2, \dots, bn$, onde $b1 = B(x)$, $b2 = B(p)$...

O lógico Raymond Smullyan definiu alguns tipos de raciocínio, para demonstração das propriedades do conjunto de crenças, listados a seguir:

- Raciocínio preciso: é quando um raciocínio nunca crê em qualquer proposição falsa.

$$\forall p(Bp \rightarrow p)$$

- Raciocínio impreciso: é quando se existe uma proposição em que se crê e esta é falsa.

$$\exists p(Bp \wedge \neg p)$$

- Raciocínio presumido: é quando se crer que um raciocínio nunca será impreciso, ou seja, ele necessariamente implicará em uma imprecisão.

$$B(\neg \exists p(Bp \wedge \neg p))$$

- Raciocínio consistente: é quando não se pode crer em uma proposição e sua negação simultaneamente.

$$\neg \exists p (Bp \wedge B\neg p)$$

- Raciocínio normal: é quando sempre que se crê em p , se crê também que se crê em p .

$$\forall p (Bp \rightarrow BBp)$$

- Raciocínio peculiar: é quando se existe alguma proposição p tal que se acredita em p e se acredita também que não se crê em p .

$$\exists p (Bp \wedge B\neg Bp)$$

- Raciocínio regular: é quando sua crença é distributiva sobre as operações lógicas.

$$\forall p \forall q ((p \rightarrow q) \rightarrow (q \rightarrow p))$$

- Raciocínio reflexivo: é quando se para toda proposição p existe uma proposição q tal que o raciocínio acredita que $q \equiv (Bq \rightarrow p)$. Como consequência, se um raciocínio reflexivo crê $Bp \rightarrow p$, logo ele acreditará em p .

$$\forall p (B(Bp \rightarrow p) \rightarrow Bp)$$

- Raciocínio instável: é quando se existe alguma proposição p tal que ela crê em p , mas na realidade ela não crê em p .

$$\exists p (BBp \rightarrow \neg Bp)$$

- Raciocínio estável: é quando um raciocínio não é instável, ou seja, se para todo p , se ele crê Bp então ele crê em p .

$$\forall p (BBp \rightarrow Bp)$$

- Raciocínio Modesto: é quando para toda proposição p , ele crê $Bp \rightarrow p$ se ele também crê em p .

$$\forall p (B(Bp \rightarrow p) \rightarrow Bp)$$

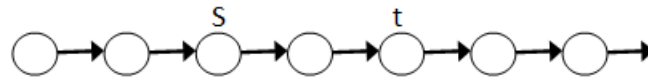
2.3 LÓGICA TEMPORAL

A Lógica Temporal, como já mencionado na introdução deste trabalho, de acordo com (SOUSA, 2007), é um tipo de Lógica Modal que permite a representação e o raciocínio sobre proposições relacionadas ao tempo, sendo possível através desta expressar frases do tipo "Eu SEMPRE estou com fome", "Eu FUTURAMENTE ficarei com fome" ou "Eu ficarei com fome ATÉ comer algo".

A Lógica Temporal é usada com muita frequência para especificação e verificação de sistemas reativos, ou seja, sistemas que interagem continuamente com o ambiente.

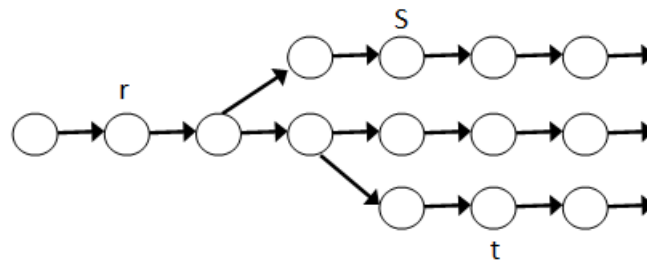
Na Lógica Temporal existem dois modelos de tempo, um deles é chamado Modelo de Tempo Linear, que pensa no tempo como um caminho de pontos no tempo, onde para quaisquer dois pontos podemos descobrir que um é anterior ao outro, ou seja, o Modelo Temporal Linear se comporta como um conjunto de traços infinitos começando no seu estado inicial, ilustrado na figura 20. O outro modelo é chamado de Modelo de Tempo Ramificado baseado em uma estrutura de tempo de ramificação, onde um ponto no tempo pode ter dois ou mais futuros pontos que não estão relacionados um ao outro. O Modelo de Tempo Ramificado é representado por uma árvore computacional de profundidade ilimitada em que sua raiz é o estado inicial, como ilustrado na figura 21 (WANG, 2017).

Figura 20 – Modelo de Tempo Linear



Fonte: Autoria própria

Figura 21 – Modelo de Tempo Ramificado



Fonte: Autoria Própria

Utilizada para especificar a ordenação temporal de eventos em sistemas concorrentes, também podendo ser usada para descrever uma sequência temporal de transições entre estados. Na Lógica Temporal se destacam as lógicas CTL(Computation Tree Logic) e LTL(Linear Temporal Logic). Enquanto que a LTL apresenta somente operadores

de lógica linear, a CTL combina tanto operadores de ramificação quanto lineares. Estas duas lógicas, são subconjuntos da lógica CTL*, a qual descreve propriedades de árvores computacionais (BARROS, 2010).

Para (RISSINO; TORRER; MARTINS, 2007), na Lógica Temporal as propriedades a serem analisadas são atribuídas por meio das fórmulas de Kripke (abordagem dos mundos possíveis mostrada na seção 2.2.1.3), representado por um conjunto de estados, transições entre estados e funções que determinam cada estado com o conjunto de propriedades verdadeiras dentro dele.

A principal característica da Lógica Temporal é a representação de valores verdade diferentes em instantes distintos de tempo por uma determinada fórmula lógica, característica esta que é padronizada através da introdução da sintaxe da linguagem da Lógica de primeira ordem de uma variedade de operadores temporais (RISSINO; TORRER; MARTINS, 2007).

Definida a partir dos seguintes operadores: G, F, H, P, temos o método de argumentos temporais, onde A é uma fórmula qualquer, P é um instante no passado, F é um instante no futuro, G todos os instantes no futuro e H todos os instantes no passado.

- **FA** - A será verdade em algum instante no futuro;
- **PA** - A foi verdade em algum instante no passado;
- **GA** - A será verdade em todos instantes do futuro;
- **HA** - A foi verdade em todos instantes do passado;

Sendo G e H operadores primitivos, através das seguintes equivalências é possível introduzir os outros operadores:

$$FA \equiv \neg G \neg A$$

$$PA \equiv \neg H \neg A$$

2.3.1 Lógica Linear Temporal(LTL)

Na Lógica Temporal Linear, para que uma fórmula seja verdadeira ela deve descrever o conjunto infinito de sequências. Em LTL o futuro pode ser visto como uma sequência de estados. Para que um sistema satisfaça pelo menos uma propriedade, é necessário que ele satisfaça a todas as sequências especificadas pela fórmula, senão o sistema não satisfará a propriedade temporal (WILGES, 2014).

A LTL é implicitamente quantificada universalmente, pois ela não pode expressar propriedades sobre um único caminho, adotando o tempo como uma sequência de execução de um sistema, onde cada possível caminho de computação é considerado isoladamente. Os caminhos das fórmulas LTL são interpretados individualmente expressando propriedades sobre uma sequência linear de execução do sistema.

Como descrito por (WILGES, 2014), uma sequência infinita de estados comumente padronizados por meio de um loop define o modelo de Lógica Temporal Linear (LTL). Fórmulas temporais são analisadas em cada loop da sequência de estados simultaneamente, isto é, os sinais das propriedades do sistema devem se manter estáveis no decorrer da fase de análise. Uma fórmula em LTL é determinada por um operador de tempo e uma propriedade p do sistema que pode ser qualquer afirmação lógica, como por exemplo, se pressionado o botão $b1$, então a porta $p1$ será aberta ($b1 \rightarrow p1$, lê-se, se $b1$ então $p1$).

2.3.1.1 Sintaxe da LTL

Para que uma fórmula LTL seja válida, deve ser formulada pelas variáveis proposicionais $p1$, $p2$, (...), conectivos da Lógica Proposicional (\neg , \vee , \wedge , \rightarrow) e os operadores temporais a seguir:

- **X(Next)**: $X\alpha$ - α é verdadeiro no estado seguinte;
- **F(Future)**: $F\alpha$ - α é verdadeiro em algum tempo no futuro (em algum estado do caminho);
- **G(Globally)**: $G\alpha$ - α é sempre verdadeiro (em todo estado no caminho);
- **U(Until)**: $\alpha U \beta$ - α é verdadeira no caminho até que β seja verdadeira;
- **R(Release)**: $\alpha R \beta$ - quando a ocorrência de um estado onde α é válida liberta β de o ser;
- **E(Exists)**: $E\alpha$ - α é verdadeiro num caminho S se existe um caminho começando em um estado S ;
- **A(All)**: $A\alpha$ - α é verdadeiro para todo caminho começando no estado S .

Seja Π um conjunto de letras proposicionais e $P \in \Pi$ uma letra proposicional. A linguagem Lógica de LTL é gerada pela seguinte BNF:

$$\Phi_p ::= P \mid (\neg \Phi_p) \mid (\Phi_p \wedge \Phi_p) \mid (\Phi_p \vee \Phi_p) \mid (\Phi_p \rightarrow \Phi_p) \mid (X\Phi_p) \mid (F\Phi_p) \mid (G\Phi_p) \mid (\Phi_p \text{ U } \Phi_p)$$

Onde os conectivos X , F e G , também podem ser representados respectivamente pelos símbolos \bigcirc , \diamond e \square .

2.3.1.2 Semântica da LTL

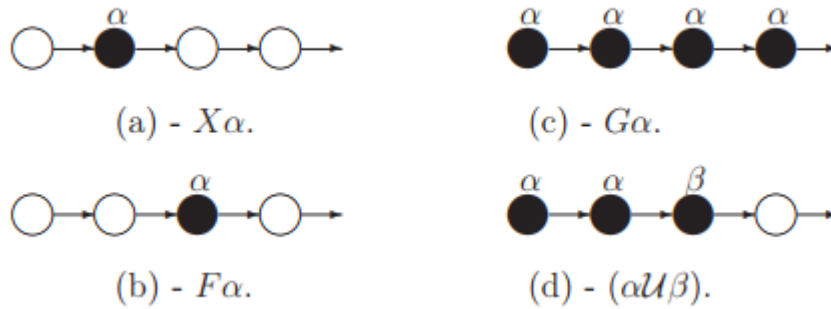
Como apresentado por (FERNANDES, 2011), a semântica da LTL pode ser definida como uma sequência infinita de estados de uma estrutura de Kripke $\sigma = s_0 s_1 \dots$ onde s_0 representa o primeiro estado da sequência, mas nem sempre significando o estado

inicial da estrutura. É definido inicialmente a notação $M, \sigma \models p$ que significa que a propriedade p é válida no decorrer da sequência s da estrutura de Kripke M .

Pela definição apresentada por (VASCONCELOS, 2007), seja Π um conjunto de letras proposicionais, P uma letra proposicional em Π , $K = \langle S, S_0, R, L \rangle$ uma estrutura de Kripke para Π , π um caminho da estrutura K , e α uma fórmula de LTL. $K \models_{\pi} \alpha$ significa que a fórmula α é satisfeita na estrutura K no caminho π . A seguir é apresentada a definição de satisfação:

- $K \models_{\pi} P \Leftrightarrow P \in L(\pi_0)$.
- $K \models_{\pi} (\neg \alpha) \Leftrightarrow \text{NÃO } K \models_{\pi} \alpha$.
- $K \models_{\pi} (\alpha \wedge \beta) \Leftrightarrow K \models_{\pi} \alpha \text{ E } K \models_{\pi} \beta$.
- $K \models_{\pi} (\alpha \vee \beta) \Leftrightarrow K \models_{\pi} \alpha \text{ OU } K \models_{\pi} \beta$.
- $K \models_{\pi} (\alpha \rightarrow \beta) \Leftrightarrow \text{SE } K \models_{\pi} \alpha \text{ ENTÃO } K \models_{\pi} \beta$.
- $K \models_{\pi} (X\alpha) \Leftrightarrow K \models_{\pi_1, \infty} \alpha$ (ver figura 22.a).
- $K \models_{\pi} (F\alpha) \Leftrightarrow \text{Existe um } k \geq 0 \text{ tal que } K \models_{\pi_k, \infty} \alpha$ (ver figura 22.b).
- $K \models_{\pi} (G\alpha) \Leftrightarrow \text{Para todo } k \geq 0 \text{ vale que } K \models_{\pi_k, \infty} \alpha$ (ver figura 22.c).
- $K \models_{\pi} (\alpha U \beta) \Leftrightarrow \text{Existe } k \geq 0 \text{ tal que } K \models_{\pi_k, \infty} \alpha \text{ e para todo } 0 \leq l < k \text{ vale que } K \models_{\pi_l, \infty} \alpha$ (ver figura 22.d).

Figura 22 – Operadores Temporais.



Fonte: (VASCONCELOS, 2007)

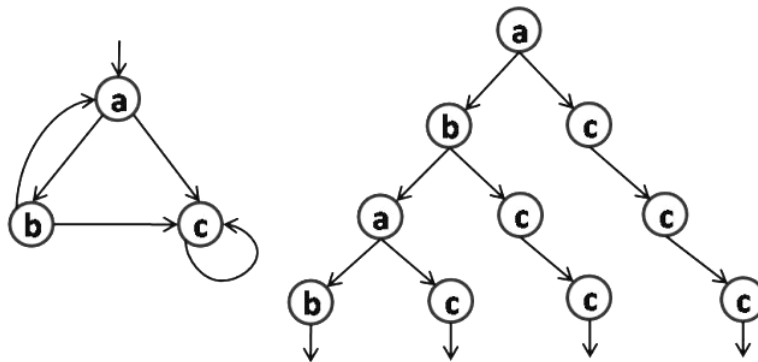
2.3.2 Lógica Computacional em árvore(CTL)

Para (OLIVEIRA, 2010) a Lógica de Árvore Computacional(Computation Tree Logic - CTL) introduz quantificadores sobre caminhos, moldando o futuro como ramificações de uma árvore. A CTL é eficiente para quantificar sobre os diferentes caminhos que

uma execução seguirá, por exemplo, se dois processos usam o mesmo recurso ao mesmo tempo, em algum momento em qualquer dos caminhos de execução possíveis.

Como (BARROS, 2010) descreve, a CTL pode ser usada para expressar propriedades que serão validadas por um verificador de modelos, utilizando árvores de computação derivantes do grafo de transição de estados a partir da estrutura de Kripke. O grafo de transição de estados origina uma árvore infinita em que a raiz é o estado inicial do grafo. Os caminhos da árvore constituem todos os resultados possíveis no modelo como mostrado na figura 23.

Figura 23 – Grafo de transição de estados e sua árvore de computação.



Fonte: (BARROS, 2010)

A Lógica de Árvore Computacional como mostrado por (FERNANDES, 2011), é uma Lógica Temporal que verifica os possíveis caminhos de uma estrutura de Kripke, sendo uma fórmula CTL verdadeira quando é satisfeita em alguns caminhos ou em caminhos que iniciam a partir de um estado. Os operadores de CTL são descritos abaixo, ressaltando que os operadores agrupados dentro dos delimitadores '[' e ']', funcionam como um único operador que não poderão se separar.

- $[EX]\alpha$ - Existe um caminho tal que no próximo estado vale α .
- $[AX]\alpha$ - Para todo caminho no próximo estado vale α .
- $[EF]\alpha$ - Existe um caminho tal que no futuro vale α .
- $[AF]\alpha$ - Para todo caminho no futuro vale α .
- $[EG]\alpha$ - Existe um caminho tal que sempre vale α .
- $[AG]\alpha$ - Para todo caminho vale sempre α .
- $E(\alpha U \beta)$ - Existe um caminho tal que vale α até que vale β .
- $A(\alpha U \beta)$ - Para todo caminho vale α até que vale β .

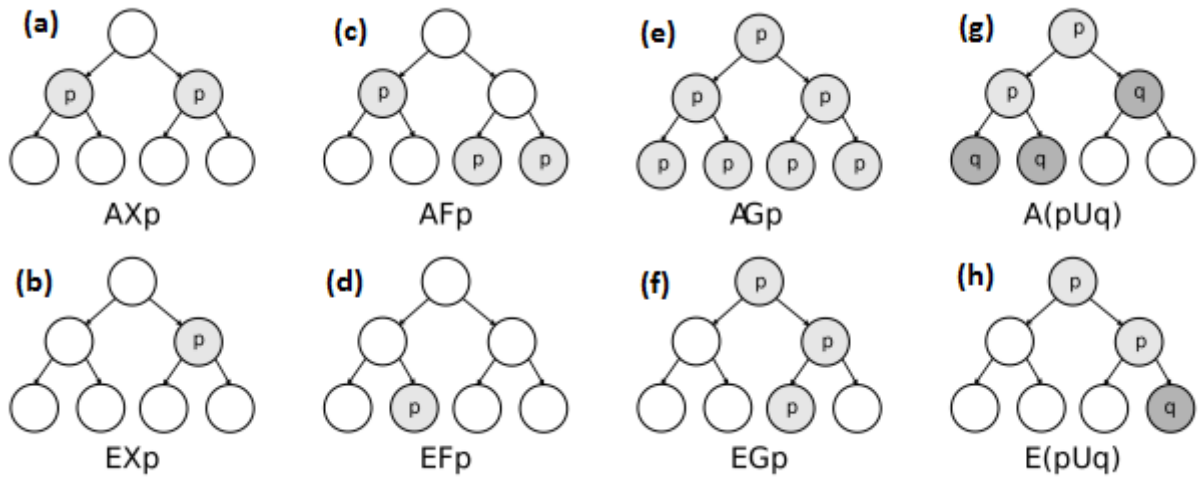
2.3.2.1 Sintaxe da CTL

A sintaxe da Lógica de Árvore Computacional é definida pela fórmula seguinte:

$$\Phi ::= P \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \rightarrow \Phi) \mid (AX\Phi) \mid (AG\Phi) \mid (AF\Phi) \mid (EX\Phi) \mid (EG\Phi) \mid (EF\Phi) \mid E(\Phi \cup \Phi) \mid A(\Phi \cup \Phi)$$

Onde p é um átomo proposicional, \neg , \wedge , \vee e \rightarrow são operadores da Lógica Proposicional, e os demais são operadores temporais. Cada operador temporal constitui um quantificador de caminho (E, "existe um caminho" ou A, "para todos os caminhos") seguido por um operador de estado (X, "próximo estado no caminho", U, "até que no caminho", G, "globalmente no caminho" ou F, "futuramente no caminho"). Na figura a seguir é ilustrada a interpretação de cada um destes operadores:

Figura 24 – Operadores temporais da lógica CTL.



Fonte: (OLIVEIRA, 2010)

Como exemplificado por (OLIVEIRA, 2010), com esses operadores podem ser expressadas sentenças como:

- Sempre que tenho sono, acabo dormindo em algum momento: $(AG(\text{sono} \rightarrow AF \text{ dormir}))$;
- É impossível que eu durma e não tenha sono: $(\neg EF(\text{dormir} \wedge \neg \text{sono}))$;
- Sempre fico sem sono logo após tomar café: $(\text{café} \rightarrow AX \neg \text{sono})$;
- Fico como sono até que eu durma ou tome café: $(A[\text{sono} \vee (\text{dormir} \vee \text{café})])$;
- É possível em algum momento no futuro que eu passe a tomar café para sempre e nunca volte a dormir: $(EF AG(\text{café} \wedge EF \neg \text{dormir}))$;

2.3.2.2 Semântica da CTL

Como apresentado por (VASCONCELOS, 2007), a semântica da Lógica de Árvore Computacional é definida da seguinte maneira:

Seja Π um conjunto de letras proposicionais, P uma letra proposicional em Π , $K = \langle S, S_0, R, L \rangle$ uma estrutura de Kripke para Π , $s \in S$ um estado da estrutura K , e α uma fórmula de CTL. $K \models_s \alpha$ significa que a fórmula α é satisfeita no estado S da estrutura K . A seguir é apresentada a definição de satisfação:

- $K \models_s P \Leftrightarrow P \in L(s)$.
- $K \models_s (\neg \alpha) \Leftrightarrow \text{NÃO } K \models_s \alpha$.
- $K \models_s (\alpha \wedge \beta) \Leftrightarrow K \models_s \alpha \text{ E } K \models_s \beta$.
- $K \models_s (\alpha \vee \beta) \Leftrightarrow K \models_s \alpha \text{ OU } K \models_s \beta$.
- $K \models_s (\alpha \rightarrow \beta) \Leftrightarrow \text{SE } K \models_s \alpha \text{ ENTÃO } K \models_s \beta$.
- $K \models_s ([AX]\alpha) \Leftrightarrow \text{Para todo caminho } \pi \text{ a partir de } s \text{ vale que } K \models_{\pi_1} \alpha$. (ver figura 24.a).
- $K \models_s ([EX]\alpha) \Leftrightarrow \text{Existe um caminho } \pi \text{ a partir de } s \text{ tal que } K \models_{\pi_1} \alpha$. (ver figura 24.b).
- $K \models_s ([AF]\alpha) \Leftrightarrow \text{Para todo caminho } \pi \text{ a partir de } s \text{ vale que existe um } k \geq 0 \text{ tal que } K \models_{\pi_k} \alpha$. (ver figura 24.c).
- $K \models_s ([EF]\alpha) \Leftrightarrow \text{Existe um caminho } \pi \text{ a partir de } s \text{ tal que existe um } k \geq 0 \text{ tal que } K \models_{\pi_k} \alpha$. (ver figura 24.d)
- $K \models_s ([AG]\alpha) \Leftrightarrow \text{Para todo caminho } \pi \text{ a partir de } s \text{ vale que para todo } k \geq 0 \text{ vale que } K \models_{\pi_k} \alpha$. (ver figura 24.e)
- $K \models_s ([EG]\alpha) \Leftrightarrow \text{Existe um caminho } \pi \text{ a partir de } s \text{ tal que para todo } k \geq 0 \text{ vale que } K \models_{\pi_k} \alpha$. (ver figura 24.f).
- $K \models_s (A(\alpha \text{ U } \beta)) \Leftrightarrow \text{Para todo caminho } \pi \text{ a partir de } s \text{ vale que existe um } k \geq 0 \text{ tal que } K \models_{\pi_k} \alpha \text{ e para todo } 0 \leq l < k \text{ vale que } K \models_{\pi_l} \alpha$. (ver figura 24.g)
- $K \models_s (E(\alpha \text{ U } \beta)) \Leftrightarrow \text{Existe um caminho } \pi \text{ a partir de } s \text{ tal que existe } k \geq 0 \text{ tal que } K \models_{\pi_k} \alpha \text{ e para todo } 0 \leq l < k \text{ vale que } K \models_{\pi_l} \alpha$. (ver figura 24.h)

Uma fórmula CTL ϕ é satisfeita por uma estrutura de Kripke $M = (S, S_0, R, L)$ se e somente se em algum dos seus estados iniciais ϕ é satisfeita.

$$M \models \phi \text{ sse } (M, s) \models \phi \text{ e } s \in S_0$$

2.3.3 Aplicações da Lógica Temporal

A Lógica Temporal tem apresentado uma larga aplicação em Engenharia e Ciência da Computação, prestando-se a verificações de software e hardware e síntese de programas concorrentes na área computacional, descrevendo o comportamento dos programas e garantindo que certas propriedades ocorrerão e outras não (JUNIOR, 1992).

Para (WILGES, 2014) a Lógica Temporal é importante para verificação de sistemas reativos que precisam ser especificados de acordo com o comportamento do ambiente. Os sistemas reativos possuem como entrada, variáveis do ambiente que por sua vez podem ser botões, sensores ou até mesmo variáveis de limite que quando são acionados devem iniciar uma tarefa, interagindo dinamicamente com o sistema. Os sistemas precisam de um espaço de tempo limitado para dar respostas, estes possuem requisitos de segurança e eventos que ocorrem em tempos determinados. São exemplos de sistemas reativos, sistemas de controle de tráfego aéreo, sistemas de tempo real e protocolos de comunicação.

Para inúmeros sistemas como os sistemas de ferrovias, aeronaves, caixas de atendimento automático bancário, plantas industriais, sistema financeiro, controles automáticos de aeronaves, controle de usinas nucleares, intertravamentos metroviários, etc, uma eventual falha causaria pequenos ou grandes prejuízos, sejam econômicos ou até mesmo podendo colocar a vida de pessoas em perigo o que causariam enormes transtornos para a sociedade. Por isso, a garantia de correção dos sistemas de controle é importante, e para isso existe a técnica de verificação de modelos ou Model Checking, um conjunto de técnicas de verificação automática que tem como entrada um modelo baseado em uma máquina de estados Finita e uma especificação em Lógica Temporal (FERREIRA, 2005).

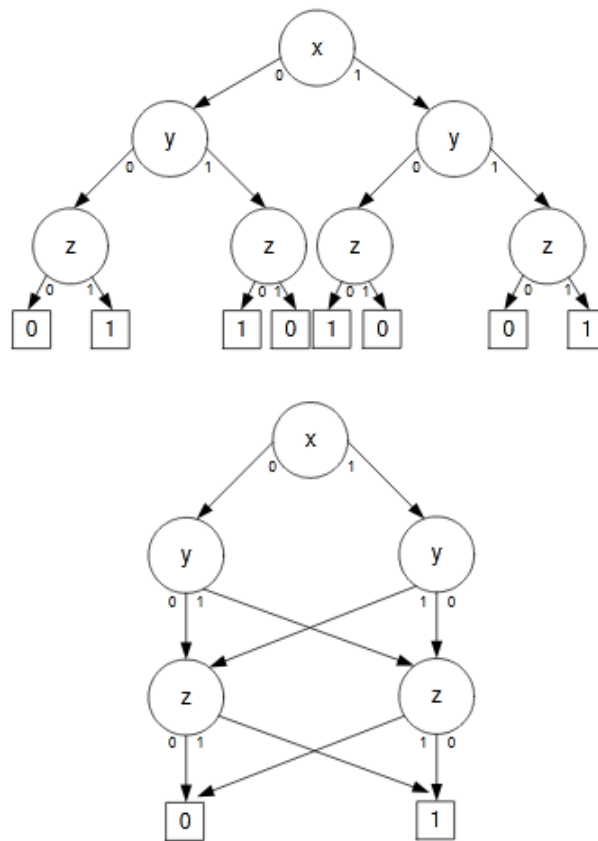
Como descrito por (RISSINO; TORRER; MARTINS, 2007), a Lógica Temporal pode ser aplicada na Engenharia de Software através do uso de técnicas de Inteligência Artificial para a melhoria dos ambientes de construção e qualidade de software, como por exemplo o ambiente de desenvolvimento PANDORA, que se trata de uma máquina de processos que utiliza a linguagem PROLOG com conceitos de Lógica Temporal. Tal ambiente possui um algoritmo de aplicação de regras em Lógica Temporal que otimiza os passos da execução, permitindo expressar e estabelecer quais ações são permitidas a cada instante, e ainda como elas serão sincronizadas, onde o tempo é descrito por uma sequência simples de eventos.

A Lógica Temporal também pode ser utilizada para elaboração de regras de restrição de integridade e em consultas de dados temporais em implementações de bancos de dados históricos, vista como uma alternativa para o tratamento de dados e recuperação de informações históricas.

2.4 DIAGRAMA BINÁRIO DE DECISÃO(BDD)

Diagramas de Decisão Binário(BDD) do inglês, Binary Decision Diagram, são grafos acíclicos direcionados que representam funções booleanas, uma árvore de decisão para a função almejada, que segue as restrições de caminho da raiz até a folha. Cada variável do BDD aparece uma única vez a partir de uma ordenação específica e cada caminho descreve uma atribuição das variáveis da função. As folhas representam o valor para um caminho determinado (NEVES, 2009).

Figura 25 – Construção de BDD



Fonte: (NEVES, 2009)

2.5 NUSMV

Uma das ferramentas mais conhecidas e usadas para verificação de sistemas é a NuSMV, desenvolvida a partir do Symbolic Model Verifier (SMV), no fim dos anos 90, pelo centro de pesquisa italiano ITC-IRST em conjunto com a Universidade de Carnegie-Mellon. Baseada no antecessor SMV, a linguagem do NuSMV foi projetada para permitir a descrição de sistemas síncronos e assíncronos, por meio de um sistema de transição de

estados, permitindo também a modularização de componentes que facilitam a manutenção e reutilização destes.

A ferramenta NuSMV possui código livre e representações através de BDDs (evitando explosão de dados) e além destas vantagens, permite verificações de sistemas de estados finitos fundamentados pelas Lógicas Temporais CTL e LTL. Além do mais, possui um algoritmo próprio para a verificação de invariantes¹, mostrando o menor caminho até o estado inconsistente. A maior desvantagem desta ferramenta é que até então ela não possui interface gráfica (SOUSA, 2007).

O NuSMV possibilita descrever propriedades temporais de forma precisa e concisa de uma especificação CTL determinando se ela é ou não satisfeita. Para isso ela utiliza os algoritmos de verificação baseados em Ordered Binary Decision Diagrams (OBDD)².

Conforme descrito por (FERNANDES, 2011) e (SOUSA, 2007), o NuSMV faz uso do cálculo proposicional para descrever as transições de uma estrutura de Kripke. A linguagem permite que qualquer fórmula proposicional possa ser utilizada para descrever as relações de transição da estrutura de Kripke, sendo bastante flexível, entretanto, apresenta certo risco de inconsistências, como a presença de contradições que podem resultar em um ou mais estados sem sucessor. Por descrever máquinas de estados finitos, a linguagem do NUSMV permite apenas tipos de dados finitos, como booleanos, escalares e vetores de tipos de dados básicos.

Um programa NuSMV compara-se a um conjunto de equações simultâneas onde o resultado destas determinará o estado seguinte. O compilador garante que em cada estado aconteça somente uma atribuição para cada variável e que o programa não apresente dependências circulares e erros de tipagem (FERNANDES, 2011).

2.5.1 Sintaxe do NuSMV

Como descrito por (SOUSA, 2007) a sintaxe do NuSMV é composta principalmente por:

- Expressões Simples: é possível descrever variáveis booleanas, numéricas, operadores lógicos e numéricos do sistema.
- Expressões Case: utilizadas para relacionar as pré-condições com as pós-condições;
- Expressões Set: utilizadas para descrever a teoria dos conjuntos, com a definição de conjuntos, teste de inclusão e união de conjuntos.

¹ Uma invariante é uma relação entre os valores das variáveis que é considerada verdadeira no início de cada iteração e não se altera de uma iteração para outra. As invariantes explicam o funcionamento do processo iterativo provando por indução que o mesmo está correto.

² Diagramas de Decisão Binária Ordenados ou em inglês Ordered Binary Decision Diagram, são como os BDDs mostrados na seção 2.4, a diferença é que os OBDDs possuem uma restrição de ordenação às variáveis e o compartilhamento de subgrafos que representam funções equivalentes.

- Expressões Next: utilizadas para expressar as transições relacionadas a uma dada variável na máquina de estados finitos, utilizada com frequência junto com as expressões case.
- Expressões CTL: usadas para definir a propriedade a ser verificada.

2.5.2 Linguagem do NuSMV

A seguir são descritos os principais conceitos, especificações e estruturas da linguagem do NuSMV baseadas no "NuSMV 2.6 User Manual" e "NuSMV 2.6 Tutorial" bem como alguns exemplos.

- **MODULE main:** é o módulo principal na linguagem do NuSMV, sendo um módulo obrigatório em toda especificação, sem parâmetros e que representa o sistema.
- **MODULE:** criados para particionar a modelagem do sistema, encapsulando as declarações das variáveis, incluindo as de estado e os eventos declarados como *BOOLEAN*. Em cada módulo pode existir a regra de transição de estados e a especificação das propriedades, onde é possível declarar um módulo com parâmetros para reutilização de código, sua passagem é feita por referência.
- **VAR:** bloco onde as declarações das variáveis são realizadas, podendo ser do tipo enumerado, intervalar, booleano e instâncias de módulos.
- **ASSIGN:** são declaradas nesta seção, as regras que determinam a inicialização e transições para o próximo valor de uma variável. A atribuição direta pelo comando *init (variável)*, estabelece um valor inicial para a variável. O comando *next (variável)* fornece o valor da variável no próximo estado, a partir do estado atual. A atribuição em *next (variável)* pode ser feita a partir da regra case, onde é possível determinar o próximo valor em função de várias condições.
- **DEFINE:** utilizada para associar uma variável a uma expressão, assim uma variável em *DEFINE* quando encontrada na especificação, será sempre substituída pela sua definição.
- **CTLSPEC:** para verificar as propriedades em formato CTL, é preciso especificá-las precedidas da palavra-chave *CTLSPEC*.
- **LTLSPEC:** para verificar as propriedades em formato LTL, é preciso especificá-las precedidas da palavra-chave *LTLSPEC*.
- **INIT:** esta seção possui uma expressão booleana que determina um estado inicial ou um conjunto de estados iniciais para o modelo.

- **TRANS:** nesta seção são descritas as transições de um modelo através de expressões booleanas que avaliam o estado atual e o próximo estado das variáveis de um modelo. Para verificar o próximo estado de uma variável, é tilizado o operador *next* do seguinte modo *next (variável) = valor*.
- **If-Then-Else:** como várias linguagens de programação, o NuSMV possui a expressão *If-Then-Else* ou operador ternário, sendo sua estrutura representada da seguinte maneira "*ExpressãoDeCondição ? Valor1 : Valor2*". Após verificar o valor da "*ExpressãoDeCondição*" retorna "*Valor1*" caso a "*ExpressãoDeCondição*" for verdadeira e caso contrário retorna "*Valor2*".

Figura 26 – Modelo de um sistema concorrente em NuSMV.

```

1  MODULE main
2  VAR
3      request : boolean;
4      state : {ready, busy};
5  ASSIGN
6      init(state) := ready;
7      next(state) :=
8          case
9              state = ready & request: busy;
10             1 : {ready, busy};
11         esac;
12  LTLSPEC
13      G(request -> F state=busy) ..

```

Fonte: (CAETANO, 2018)

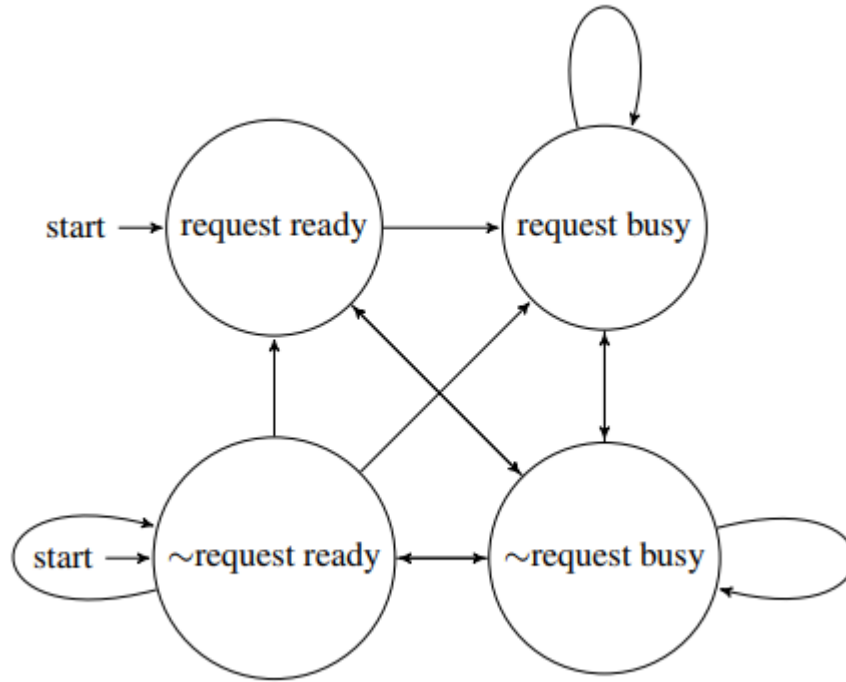
A figura acima apresenta o modelo do sistema descrito ou programado em NuSMV. Assim como algumas linguagens de programação, (CAETANO, 2018) alega que um programa em NuSMV pode conter um ou mais módulos, um deles deve ser denominado "main". As variáveis e seus valores são declaradas dentro dos módulos, normalmente são atribuídas com um valor inicial para cada uma delas e depois é especificado o próximo valor através de uma expressão constituída pelos valores correntes das variáveis.

O programa em NuSMV mostrado na figura 26, é um exemplo de um sistema hipotético que compõe de duas variáveis, *request* do tipo booleana e *state* do tipo enumerado "*ready, busy*", onde 0 e 1 representam "false" e "true" respectivamente. Os valores da variável *request* são definidos somente pelo ambiente durante a execução do programa. A variável *state* é inicialmente definida como "*ready*" podendo mudar para "*busy*" caso a variável *request* for "true", entretanto se *request* for "false" o valor de *state* fica indeterminado.

A expressão *case* na linha 9 da figura 26, caso *ready* e *request* foram verdadeiras, será atribuído à variável *state* o estado *busy* através do sinal ":"(dois pontos). Na linha

10 caso a condição apresentada na linha 9 não for satisfeita por default a variável *state* se mantém inalterada. em outras palavras na primeira avaliação antes do sinal ":" (dois pontos) que possui valor verdadeiro, será atribuída o valor da direita, "*busy*" para a variável *state* declarada. Sendo assim caso nenhuma das avaliações sejam verdadeiras o valor "1:" é mantido como avaliação padrão.

Figura 27 – Máquina de Estados do Sistema.



Fonte: (MOREIRA, 2016)

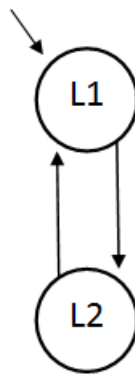
O programa da figura 26 expressa o sistema de transição da figura 27, que consiste de quatro estados, cada um definido em função dos possíveis valores para as duas variáveis. O nome "*~request*" é a negação para a variável "*request*". O programa e o sistema de transição são não-determinísticos, ou seja, o estado seguinte não é unicamente definido. Todas as transições de estado baseado no desempenho da variável "*state*" aparecem em pares, resultado na transição para um estado sucessor onde "*request*" pode ser falso ou verdadeiro. A partir do estado "*~request, busy*", na figura 27, é notável que o sistema pode seguir quatro estados destinos incluindo ele mesmo. As especificações em LTL são incorporadas pela palavra LTLSPEC, sendo simples fórmulas LTL. No exemplo de programa descrito, a especificação declara que para qualquer estado, se o valor de "*request*" é verdadeiro, então eventualmente ocorrerá um estado "*busy*" (MOREIRA, 2016).

2.5.3 Alguns exemplos utilizando NuSMV

Os conceitos e exemplos sobre NuSMV abordados neste trabalho, são apresentados de forma resumida, portanto, para maiores informações e aprofundamento sobre a ferramenta NuSMV consulte "NuSMV 2.6 User Manual" e "NuSMV 2.6 Tutorial" ambos disponíveis em <<https://nusmv.fbk.eu/>>.

2.5.3.1 Exemplo simples de transição

Figura 28 – Sistema de Transição simples



Fonte: Autoria própria

Na figura 28 temos um exemplo simples de um sistema de transição que possui 2 estados, l1 e l2, onde l1 é o estado inicial e existe uma transição de l1 para l2, e uma de l2 para l1. Este sistema de transição é descrito da seguinte maneira na linguagem do NuSMV:

Figura 29 – Código NuSMV do sistema de transição simples

```
1 MODULE main
2   VAR
3     location:{l1,l2};
4   ASSIGN
5     init(location) := l1;
6     next(location) := case
7       (location = l1) : l2;
8       (location = l2) : l1;
9     esac;
```

Fonte: Autoria própria

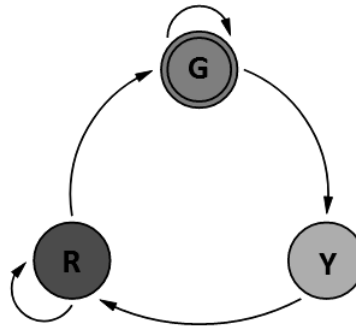
Todo código NuSMV inicia com a instrução "*MODULE main*" seguindo com as descrições dos estados no bloco "*VAR*", onde é definida uma variável nomeada "*location*"

do tipo enumerado que assume os valores l1 e l2, que são os 2 estados do sistema de transição.

O bloco de atribuição "*ASSIGN*", descreve as transições dos estados da variável "*location*", inicialmente especificando qual é o estado inicial, no caso l1, então o valor inicial "*init (location)*" será l1, em seguida as transições em "*next (location) := case*" indicará os próximos estados, se caso a variável "*location*" for l1, o próximo estado será l2, e se a variável "*location*" for l2, o próximo estado será l1;

2.5.3.2 Exemplo de um semáforo simples

Figura 30 – Sistema de transição de um semáforo



Fonte: (MATEOS, 2013)

A figura acima é um sistema de transição que representa o funcionamento de um semáforo, que contém 3 estados representados na figura pelas letras G, Y e R, Verde (Green), Amarelo (Yellow) e Vermelho (Red) respectivamente, que se alternam da seguinte maneira:

Verde \rightarrow Amarelo \rightarrow Vermelho \rightarrow Verde...

As luzes verde e vermelha podem ficar acesas por um período indeterminado de tempo, enquanto a luz amarela fica acesa apenas por um momento. As propriedades desejáveis para o semáforo são as seguintes:

- Após o semáforo ficar verde, em algum momento ele ficará vermelho, ou seja, ele não ficará bloqueado no verde. Em LTL: $G ((\text{luz} = \text{verde}) \rightarrow F (\text{luz} = \text{vermelho})) \neg FG (\text{luz} = \text{verde})$.
- Após o semáforo ficar vermelho, em algum momento ele ficará verde, ou seja, ele não ficará bloqueado no vermelho. Em LTL: $G ((\text{luz} = \text{vermelho}) \rightarrow F (\text{luz} = \text{verde})) \neg FG (\text{luz} = \text{vermelho})$
- A luz amarela pode acender em algum momento, na verdade ela deve acender um número infinito de vezes no tempo. Em CTL: $EF (\text{luz} = \text{amarelo}) AG AF (\text{luz} = \text{amarelo})$

Figura 31 – Especificação do semáforo em NuSMV

```

1  MODULE main
2      VAR
3          luz : {vermelho,verde,amarelo};
4      ASSIGN
5          init(luz) := verde;
6          next(luz) := case
7              luz = verde : {verde,amarelo};
8              luz = vermelho : {vermelho,verde};
9              luz = amarelo : vermelho;
10         esac;
11  FAIRNESS
12      luz = amarelo;
13  LTLSPEC G ((luz = verde) -> F (luz = vermelho));
14  LTLSPEC ! F G (luz = verde);
15  LTLSPEC G ((luz = vermelho) -> F (luz = verde));
16  LTLSPEC ! F G (luz = vermelho);
17  SPEC EF (luz = amarelo);
18  SPEC AG AF (luz = amarelo);

```

Fonte: Autoria própria

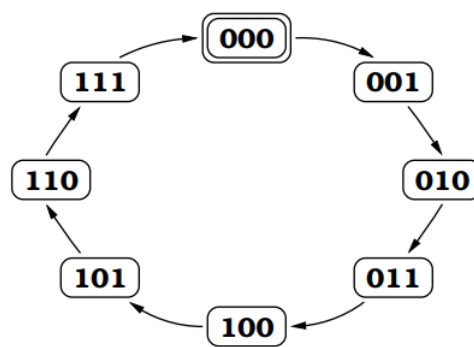
As fórmulas LTL são indicadas pelo termo LTLSPEC e as fórmulas CTL são indicadas pelo termo SPEC.

A seção FAIRNESS é adicionada ao código como uma condição de imparcialidade, onde a fórmula será atendida por uma quantidade infinita de vezes ao longo do tempo.

2.5.3.3 Contador de 3 bits

Assim como (MATEOS, 2013), consideremos um contador binário de três bits em que o sistema de transição associado é o seguinte:

Figura 32 – Sistema de transição do contador de 3 bits



Fonte: (MATEOS, 2013)

Apesar de ser possível especificar este modelo no NuSMV por meio de uma variável com 8 valores distintos, no exemplo a seguir é especificado com 3 variáveis, uma para cada bit.

Figura 33 – Especificação em NuSMV do contador de 3 bits

```

1  MODULE main
2      VAR
3          bit0 : celula(TRUE);
4          bit1 : celula(bit0.saida);
5          bit2 : celula(bit1.saida);
6      SPEC
7          AG AF bit2.saida;
8
9  MODULE celula(entrada)
10     VAR
11         valor : boolean;
12     ASSIGN
13         init(valor) := FALSE;
14         next(valor) := case
15             valor: !entrada;
16             TRUE: entrada;
17         esac;
18     DEFINE
19         saida := valor & entrada;

```

Fonte: Autoria própria

Cada bit do contador é especificado como uma instância do módulo "*celula*" que possui o argumento de entrada "*entrada*" que terá que ser especificado ao fazer a instância. A variável "*saida*" é definida na seção *DEFINE*.

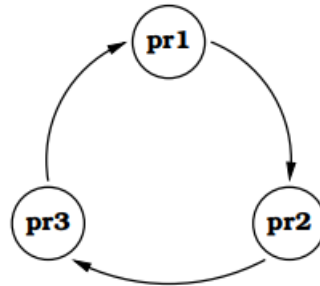
Como explicado por (MATEOS, 2013), os argumentos de entrada são variáveis que intervêm na descrição dos estados do sistema de transição, cujo valor inicial provém de outro módulo. O módulo *main* não pode ter argumentos de entrada. As variáveis definidas em uma seção *DEFINE* não intervêm na descrição do conjunto de estados da transição.

O contador de três bits é especificado com três variáveis onde cada uma delas é uma instância do módulo *celula*. O valor de entrada para *bit0* é 1, enquanto que o valor de entrada para o *bit1* é o valor de saída definido no primeiro bit "*bit0.saida*" e o valor de entrada para o *bit2* é o valor de saída definido no segundo bit "*bit1.saida*". Neste modelo verifica-se que o terceiro bit está ativado a um número infinito de vezes no tempo em CTL: "*AG AF bit2.saida*" (MATEOS, 2013).

2.5.3.4 Anel de inversores

Seja um anel formado por três inversores conectados da seguinte maneira:

Figura 34 – Sistema de transição de um anel de inversores



Fonte: (MATEOS, 2013)

Especificando este exemplo na linguagem NuSMV, teremos três sistemas de transição executados de forma assíncrona, relacionados como mostrado na figura acima.

Figura 35 – Especificação em NuSMV do anel inversor

```

1  MODULE main
2      VAR
3          pr1 : process inversor(pr3.saida);
4          pr2 : process inversor(pr1.saida);
5          pr3 : process inversor(pr2.saida);
6
7  MODULE inversor(input)
8      VAR
9          saida : boolean;
10     ASSIGN
11         init(saida) := FALSE;
12         next(saida) := !input;
13     FAIRNESS
14         running;
```

Fonte: Autoria própria

Das linhas 7 á 12, cada inversor é especificado como uma instância do módulo "*inversor*", que possui entrada do argumento que terá que ser especificado ao fazer a instância e a variável de saída armazena o valor do argumento de entrada "*input*".

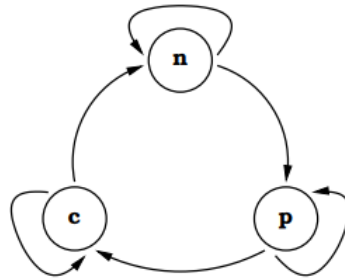
Das linhas 1 á 5, o anel de inversores é especificado com três variáveis, onde cada uma delas é uma instância do módulo "*inversor*". Cada inversor receberá como entrada o valor da variável "*saida*" de outro inversor. Por padrão, a atribuição de valores às variáveis dos diferentes módulos de uma especificação, são feitas em paralelo, se comportando de maneira assíncrona, onde apenas um dos módulos está em operação a todo o tempo. O comportamento assíncrono é alcançado incluindo a palavra "*process*" na frente do nome do módulo.

Como ressaltado por (MATEOS, 2013), o comportamento assíncrono pode fazer com que um módulo nunca seja executado, e para forçar todos os módulos a serem executados de forma equilibrada, devem ser adicionadas condições de imparcialidade como feito nas linhas 13 e 14. Cada módulo executado de forma assíncrona tem a variável *"running"* associada, cujo valor é 1 se o módulo estiver em execução, caso contrário o seu valor é 0. A condição de imparcialidade é estabelecida nesta variável.

2.5.3.5 Exclusão mútua

Neste exemplo vamos considerar dois processos que usam um recurso comum em exclusivo, ou seja, ambos os processos não podem usar o mesmo recurso simultaneamente. Cada processo pode estar em três situações: normal (n), solicitando o recurso (p) ou utilizando o recurso (c), de acordo com o esquema de transição a seguir:

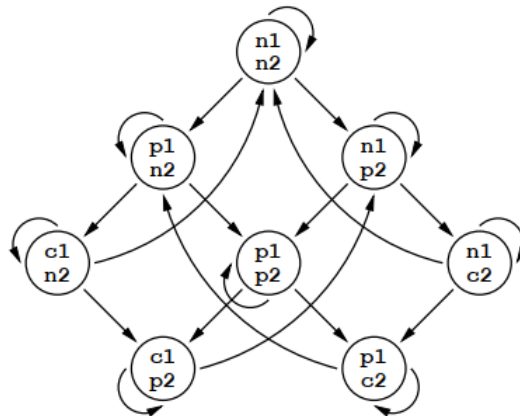
Figura 36 – sistema de transição de um processo



Fonte: (MATEOS, 2013)

O sistema de transição com exclusão mútua para este exemplo é ilustrado a seguir, onde cada processo é identificado com um número (1 ou 2), e cada estado com uma letra (n, p, c).

Figura 37 – sistema de transição dos processos com exclusão mútua



Fonte: (MATEOS, 2013)

Como explicado (MATEOS, 2013), em um modelo de exclusão mútua, são desejáveis as seguintes propriedades:

- Os processos não podem ocupar um mesmo recurso simultaneamente.
Em LTL: $G (\neg(c1 \wedge c2))$
- Se um processo solicita o recurso, em algum momento ele deverá usá-lo.
Em LTL: $G (t1 \rightarrow F c1)$
 $G (t2 \rightarrow F c2)$
- Os processos devem sempre ter a possibilidade de solicitar o recurso.
Em CTL: $AG (n1 \rightarrow EX p1)$
 $AG (n2 \rightarrow p2)$
- O uso de um recurso pelos processos não é feito estritamente de maneira sequencial.
Em CTL: $EF (c1 \wedge E[c1 U (\neg c1 \wedge E[\neg c2 U c1]))]$.
 $EF (c2 \wedge E[c2 U (\neg c2 \wedge E[\neg c1 U c2]))]$.

Abaixo é apresentado um código em NuSMV do processo de exclusão mútua de acordo com a figura 37.

Figura 38 – Especificação NuSMV do processo de exclusão mútua

```

1  MODULE main
2      VAR
3          proceso1 : process proceso(proceso2.estado,turno,FALSE);
4          proceso2 : process proceso(proceso1.estado,turno,TRUE);
5          turno : boolean;
6      ASSIGN
7          init(turno) := FALSE;
8
9  MODULE proceso(otro,turno,miturno)
10     VAR
11         estado: {n, p, c};
12     ASSIGN
13         init(estado) := n;
14         next(estado) :=
15             case
16                 (estado = n) : {p,n};
17                 (estado = p) & (otro = n) : c;
18                 (estado = p) & (otro = p) & (turno = miturno) : c;
19                 (estado = c) : {c,n};
20             TRUE : estado;
21         esac;
22     next(turno) :=
23         case
24             (turno = miturno) & estado = c : !turno;
25             TRUE : turno;
26         esac;
27     FAIRNESS
28         running
29     FAIRNESS
30         !(estado = c);

```

Fonte: código de (MATEOS, 2013)

O argumento "*miturno*" serve para identificar um processo versus o outro. A variável "*turno*" serve apenas para estabelecer uma mudança de acesso ao recurso quando ambos os processos o solicitarem simultaneamente, os processos não usam o recurso em sequência estrita (MATEOS, 2013).

Para evitar que um dos processos solicite um recurso indefinidamente enquanto o outro usa, é estabelecido que cada processo use o recurso em turnos alternados, para isso utiliza-se uma variável auxiliar para 'duplicar' o estado denominado com [p1 p2].

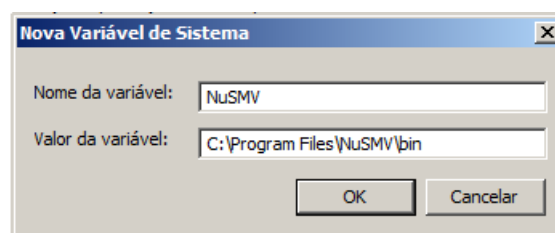
Das linhas 27 a 30, para que os processos funcionem de maneira equilibrada, é adicionada a condição de imparcialidade "*running*". Esta condição ! (Estado = c) garante que um processo não monopolize o uso do recurso compartilhado, liberando-o no seu tempo certo.

2.6 EXECUÇÃO DO NUSMV

Para este trabalho, foi utilizado um computador com Windows 7 e para que a execução da ferramenta NuSMV fosse possível, foram feitos os seguintes passos que podem ser considerados para qualquer versão do Windows:

- Primeiramente o download da ferramenta disponível em seu site oficial <<https://nusmv.fbk.eu/>>, onde terá um link para cada versão de sistema.
- Após concluído o download foi feita a descompactação da pasta dos arquivos e em seguida foi feita uma cópia da pasta extraída para dentro da pasta "Program Files" localizada na unidade de disco local onde está instalado o sistema.
- Em seguida abrindo as propriedades do sistema no painel de controle em "Configurações avançadas do sistema" \Rightarrow "Avançado" \Rightarrow "Variáveis de ambiente" e em "variáveis do sistema" clicar em "novo", digitar o nome da variável, que foi chamada de "NuSMV" e digitar o valor da variável que seria o endereço onde está localizado o arquivo executável do NuSMV e clicar em "ok":

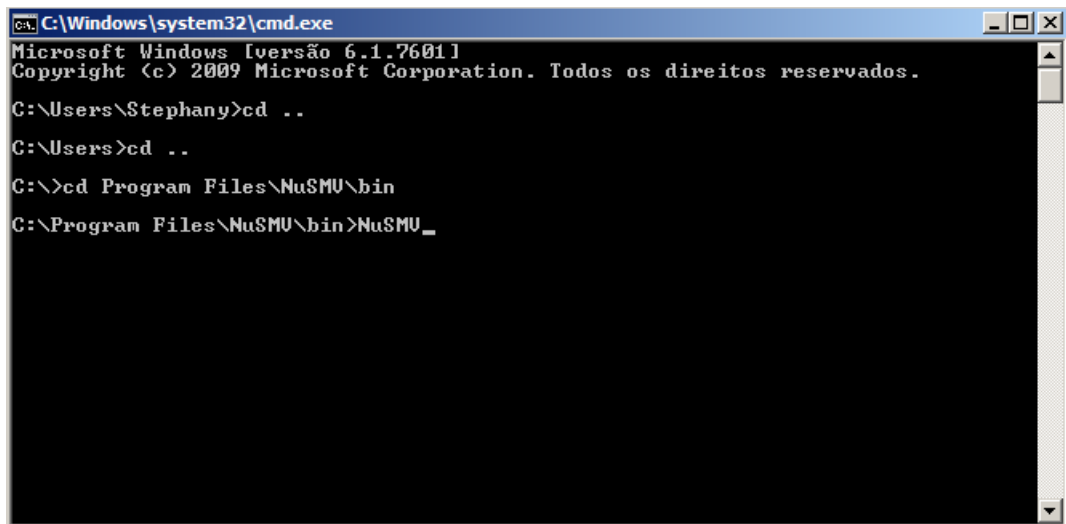
Figura 39 – Criação de nova variável do sistema



Fonte: Autoria própria

- Logo após estes passos, é aberto o cmd do windows e digitado o comando para entrar na pasta do NuSMV, o endereço da pasta onde ele está localizado e o nome da variável "NuSMV" que foi adicionada no passo anterior:

Figura 40 – Localizando o NuSMV para execução



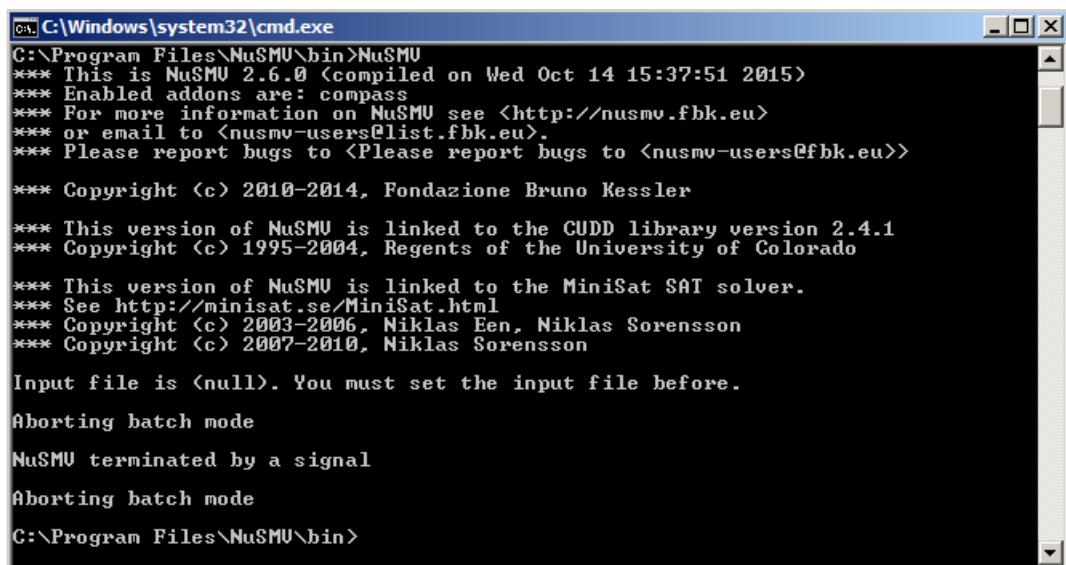
```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Stephany>cd ..
C:\Users>cd ..
C:\>cd Program Files\NuSMV\bin
C:\Program Files\NuSMV\bin>NuSMV_
```

Fonte: Autoria própria

- Feito isso vamos executar a ferramenta NuSMV pelo cmd dando um "Enter":

Figura 41 – Execução do NuSMV



```
C:\Windows\system32\cmd.exe
C:\Program Files\NuSMV\bin>NuSMV
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler
*** 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

Input file is (null). You must set the input file before.
Aborting batch mode
NuSMV terminated by a signal
Aborting batch mode
C:\Program Files\NuSMV\bin>
```

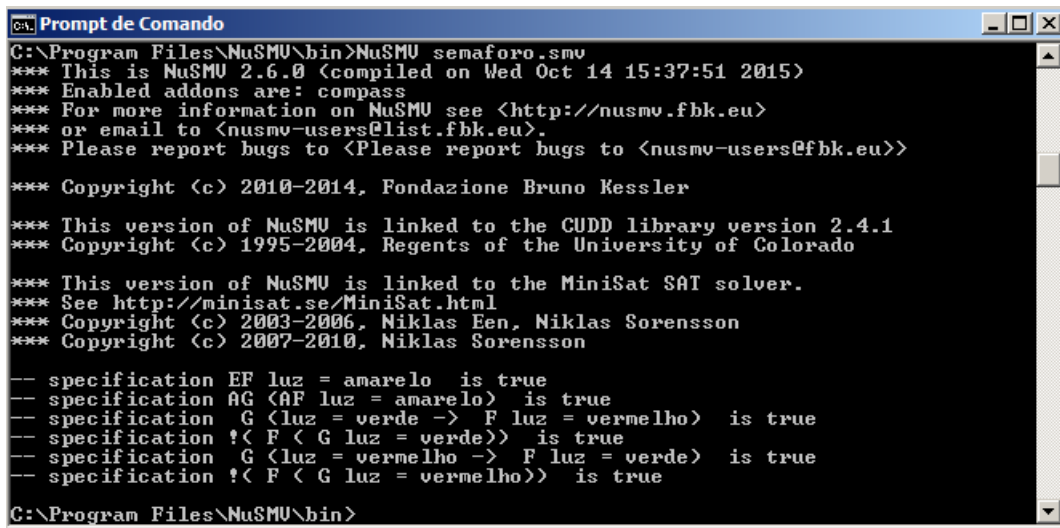
Fonte: Autoria própria

- Para que um programa em smv seja executado é preciso que ele esteja copiado dentro da pasta "bin" onde se encontra o arquivo "NuSMV.exe", assim ele poderá ser chamado a ser executado e compilado pela ferramenta NuSMV através do prompt de comando.
- Para exemplificar a execução de um programa na linguagem NuSMV, consideremos código do exemplo do semáforo mostrado na seção 2.5.3.2 na figura 31. Todo programa em NuSMV deve ser escrito em algum bloco de notas e salvo com a extensão ".smv", como no exemplo: "semaforo.smv".

- Após a instalação e execução da ferramenta NuSMV, vamos executar o programa pelo cmd digitando o seu nome *"semaforo.smv"*, logo após "NuSMV":

"C:Files>NuSMV semaforo.smv"

Figura 42 – Execução do código do semáforo



```
C:\Program Files\NuSMV\bin>NuSMV semaforo.smv
*** This is NuSMV 2.6.0 <compiled on Wed Oct 14 15:37:51 2015>
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** 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 EF luz = amarelo is true
-- specification AG <AF luz = amarelo> is true
-- specification G <luz = verde -> F luz = vermelho> is true
-- specification !< F < G luz = verde>> is true
-- specification G <luz = vermelho -> F luz = verde> is true
-- specification !< F < G luz = vermelho>> is true

C:\Program Files\NuSMV\bin>
```

Fonte: Autoria própria

- Feita a execução do código da figura 31 da seção 2.5.3.2, são mostradas no cmd as especificações do programa e se elas são verdadeiras ou falsas como ilustrado na figura acima.

3 CONCLUSÃO E TRABALHOS FUTUROS

A partir do desenvolvimento deste trabalho foi possível perceber que o NuSMV é uma ótima ferramenta no suporte para aprendizado da Lógica Temporal e verificação de modelos, além da importância desta área de estudo para a Ciências da Computação e outras áreas do conhecimento.

Técnicas de verificação de modelos utilizando a Lógica Temporal se tornam cada vez mais presentes para a verificação de sistemas reativos, uma vez que suas execuções são determinadas por eventos externos, como é o caso de sistemas de controle de usinas nucleares, controle automático de aeronaves, sistemas metroviários entre outros, onde a segurança e confiabilidade são imprescindíveis.

É notável a grande importância da Lógica para a Ciência da Computação, e ainda que esta área seja bastante abrangente, muitos estudantes deste curso não conhecem ou não tem acesso a outros tipos de lógica que consideravelmente contribuíram e ainda contribuem para o crescimento da Ciência da Computação como é o caso da Lógica Temporal.

O objetivo inicial deste trabalho foi desenvolver um material introdutório sobre Lógica Modal com ênfase em Lógica Temporal através do uso da ferramenta NuSMV para ser aplicado a um grupo de alunos do curso de Ciências da Computação da Universidade Estadual Vale do Acaraú, porém, devido a pandemia e visto que todas as atividades presenciais foram suspensas, a aplicação do mesmo não pôde ser realizada. Portanto, o objetivo atual deste trabalho foi a produção de um material introdutório para auxiliar no estudo sobre Lógica Temporal e a ferramenta NuSMV.

As informações apresentadas neste trabalho foram frutos de um estudo, leitura e pesquisa sobre todos os tópicos abordados aqui. E tendo em vista que o objetivo inicial deste trabalho não pôde ser cumprido, devido ser algo um pouco complexo a ser feito de forma remota, fica a proposta para um possível trabalho futuro a aplicação deste material para um grupo de estudantes de Ciência da Computação de forma presencial como proposto inicialmente, e consequentemente os resultados obtidos poderão dar origem a um material para produção e publicação de um futuro artigo.

Nesse sentido, esse trabalho poderá contribuir para uma visão mais ampla sobre o estudo da Lógica Temporal, permitindo a qualquer pessoa interessada sobre o assunto entender de uma forma mais sucinta sobre este campo de estudo e assim abrir mais horizontes para o desenvolvimento de novos trabalhos.

REFERÊNCIAS

- ALMEIDA, D. C. P. de. A persistência do paradoxo da cognoscibilidade. *Unicamp*, Campinas - SP, 2011.
- ALVES, A. S. Avaliação do problema de ordenação em diagramas de decisão binária. *UFRJ*, Rio de Janeiro, 2001. Disponível em: <<https://www.cos.ufrj.br/uploadfile/publicacao/843.pdf>>.
- BARROS, C. de M. Verificação automatizada de regras de sequenciamento de execução em workflows. Belo Horizonte, 2010.
- BEDREGAL, B. M. A. B. R. *Introdução à Lógica Clássica para a Ciência da Computação*. Natal: [s.n.], 2007.
- BENEVIDES, P. M. *Lógicas Modais*. [S.l.]: UFRJ, 2016.
- BERTOLINI, C.; CUNHA, G. B. da; FORTES, P. R. *Lógica Matemática*. 1. ed. Santa Maria- RS: UAB/NTE/UFSM, 2017.
- CAETANO, A. de S. F. Verificação automática de sistemas descritos usando a linguagem ladder. Belo Horizonte, 2018.
- CAVADA ALESSANDRO CIMATTI, C. A. J. G. K. E. O. M. P. M. R. A. T. R. Nusmv 2.6 tutorial. *Universidade de Trento*, Itália, 2010. Disponível em: <<https://nusmv.fbk.eu/>>.
- CAVADA ALESSANDRO CIMATTI, C. A. J. G. K. E. O. M. P. M. R. A. T. R. Nusmv 2.6 user manual. *Universidade de Trento*, Itália, 2010. Disponível em: <<https://nusmv.fbk.eu/>>.
- CHAGAS, E. M. P. de F. Apresentando alguns aspectos históricos do desenvolvimento da lógica clássica, ciência das ideias e dos processos da mente. *Espectrum*, 2004.
- COSCARELLI, B. C. Introdução à lógica modal. 2008.
- DIANA, R. R. M. Verificação de código sql via verificação de modelos. *Pontifícia Universidade Católica de Minas Gerais*, Belo Horizonte, 2011.
- DOMÍNGUEZ, D. Hybrid logic as extension of modal and temporal logics. *Humanities Journal of Valparaíso*, Universidade de Salamanca - Espanha, n. 13, 2019.
- D'OTTAVIANO, H. d. A. F. Ítala M. L. Sobre a história da lógica, a lógica clássica e o surgimento das lógicas não-clássicas. 2003.
- FAJARDO, R. A. dos S. Combinações sobre lógicas modais não-normais. 2004.
- FERNANDES, F. G. Um arcabouço para verificação automática de modelos uml. Belo Horizonte, 2011.
- FERREIRA, N. F. G. Verificação formal de sistemas modelados em estados finitos. *USP*, São Paulo, 2005.
- FILHO, E. de A. *Iniciação à lógica matemática*. Nobel, 2002. ISBN 9788521304036. Disponível em: <<https://books.google.com.br/books?id=s7GKHIjAQC4C>>.

- GABBAY, D. M.; WOODS, J. *Handbook of the History of Logic - Vol. 7 - Logic and the Modalities in the Twentieth Century*. 1. ed. Amsterdam: Elsevier, 2006.
- GOMES, N. G. Um panorama da lógica deôntica. *KRITERION*, Belo Horizonte, 2008.
- GORANKO, V.; GALTON, A. Temporal logic. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2015. [S.l.]: Metaphysics Research Lab, Stanford University, 2015.
- GORSKY, S. B. A semântica algébrica para lógicas modais e seu interesse filosófico. 2008.
- HUTH, M.; RYAN, M. *Logic in Computer Science: Modelling and Reasoning About Systems*. New York, NY, USA: Cambridge University Press, 2004. ISBN 052154310X.
- IKEDA, M. T. Y. Teste de software baseado em lógica temporal linear em sistemas reativos. *Universidade Federal de Ouro Preto*, João Monlevade - MG, 2018.
- JUNIOR, B. I. da S. Lógica temporal de tempo real generalizada aplicada ao controle e simulação de sistemas dinâmicos e eventos discretos. *UNICAMP*, Campinas, 1992.
- JUNIOR, W. P. Mineração de padrões temporais híbridos especificados na lógica temporal de intervalos. *Universidade Federal de Uberlândia*, Uberlândia - MG, 2007.
- MAGOSSI, J. C. Uma lógica modal temporal. *UNICAMP*, 1994.
- MARTINS, L. G. A. *Apostila de Lógica Proposicional(Fundamentos básicos)*. Uberlândia: UFU, 2018.
- MATEOS, F. J. M. Nusmv: Interacción con el sistema. *Universidad de Sevilla*, Espanha, 2013. Disponível em: <<https://www.cs.us.es/cursos/ra/temas/tema-NuSMV-4.pdf>>.
- MELO DENIS FELIPE, D. J. M. I. R. M. S. J. P. d. C. C. L. C. P. B. P. N. Lógica modal. 2009.
- MOREIRA, A. L. G. S. A. Modelagem e verificação automática de um protocolo de controle de fluxo adaptativo usando traços de execução. *Universidade Federal de Campina Grande*, Paraíba, 2016.
- MORTARI, C. A. *Introdução à Lógica*. 1. ed. São Paulo: Editora UNESP, Imprensa Oficial do Estado, 2001.
- NEVES, J. C. V. Algoritmo distribuído para exploração de cláusulas com bounded model checking através de uma busca em profundidade. *Pontifícia Universidade Católica de Minas Gerais*, Belo Horizonte, 2009.
- OHRSTROM, P.; HASLE, P. F. V. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Dodrecht/Boston/London: Kluwer Academic Publishers, 1995.
- OLIVEIRA, P. de T. G. Revisão de modelos ctl. São Paulo, 2010.
- RISSINO, S.; TORRER, G. L.; MARTINS, H. G. Lógica temporal aplicada a sistemas de informação. 2007.

- ROVER, A. J. Representação do conhecimento legal em sistemas especialistas: O uso da técnica de enquadramentos. *Universidade Federal de Santa Catarina*, Florianópolis, 1999.
- SANTOS, B. R. Um método para verificação formal e dinâmica de sistemas de software concorrentes. *Universidade Federal de Alagoas*, Maceió, 2016.
- SCHECHTER, L. M. Aplicações de lógicas modais a teoria de grafos e sistemas concorrentes. *UFRJ*, Rio de Janeiro, 2010.
- SOUSA, T. C. de. Revisão de modelos formais de sistemas de estados finitos. *USP*, São Paulo, 2007.
- SOUZA, J. N. de. *Lógica para Ciência da Computação: uma introdução concisa*. 2. ed. Rio de Janeiro: Elsevier, 2008. ISBN 978-85-352-2961.
- SRIVATHSAN, B. Simple models in nusmv. *NPTEL*, Índia, 2015. Disponível em: <https://www.youtube.com/watch?v=GIrOek9sGyQ&t=240s>.
- VASCONCELOS, D. R. de. Lógica modal de primeira-ordem para raciocinar sobre jogos. *PUC-Rio*, Rio de Janeiro, 2007.
- WANG, J. *Real-Time Embedded Systems*. 1. ed. USA: Wiley, 2017.
- WILGES, P. Verificador temporal de propriedades em vhdl durante a simulação. *UFRGS*, Porto Alegre, 2014.
- WONDRACEK, A. do C. S. Tradução de modelos de redes de automatos estocásticos para a linguagem do nusmv. 2013.