

CENTRO UNIVERSITÁRIO FEI
ALEXANDER GUILHERME SATURNO

**PROVA DE CORREÇÃO DO ALGORITMO α ASP(MDP) UTILIZANDO
REPRESENTAÇÃO DIAGRAMÁTICA**

São Bernardo do Campo

2019

ALEXANDER GUILHERME SATURNO

**PROVA DE CORREÇÃO DO ALGORITMO oASP(MDP) UTILIZANDO
REPRESENTAÇÃO DIAGRAMÁTICA**

Qualificação de Mestrado apresentada ao
Centro Universitário da FEI para obtenção
do título de Mestre em Engenharia Elétrica.
Orientado pelo Prof. Dr. Paulo Eduardo
Santos.

São Bernardo do Campo

2019

Saturno, Alexander Guilherme.

Prova de correção do algoritmo oASP (MDP) utilizando
representação diagramática / Alexander Guilherme Saturno. São
Bernardo do Campo, 2019.

96 f. : il.

Dissertação - Centro Universitário FEI.

Orientador: Prof. Dr. Paulo Eduardo Santos.

1. Raciocínio Diagramático. 2. Prova de Correção. 3. Método
da Indução Matemática. I. Santos, Paulo Eduardo, orient. II.
Título.

Aluno: Alexander Guilherme Saturno

Matrícula: 117201-4

Título do Trabalho: Prova de correção do algoritmo oASP (MDP) utilizando representação diagramática.

Área de Concentração: Inteligência Artificial Aplicada à Automação e Robótica

Orientador: Prof. Dr. Paulo Eduardo Santos

Data da realização da defesa: 22/03/2019

ORIGINAL ASSINADA

Avaliação da Banca Examinadora:

São Bernardo do Campo, / / .

MEMBROS DA BANCA EXAMINADORA

Prof. Dr. Paulo Eduardo Santos

Ass.: _____

Prof. Dr. Plinio Thomaz Aquino Junior

Ass.: _____

Prof. Dr. Flávio Soares Correa da Silva

Ass.: _____

A Banca Julgadora acima-assinada atribuiu ao aluno o seguinte resultado:

APROVADO

REPROVADO

VERSÃO FINAL DA DISSERTAÇÃO

**APROVO A VERSÃO FINAL DA DISSERTAÇÃO EM QUE
FORAM INCLUÍDAS AS RECOMENDAÇÕES DA BANCA
EXAMINADORA**

Aprovação do Coordenador do Programa de Pós-graduação

Prof. Dr. Carlos Eduardo Thomaz

AGRADECIMENTOS

Agradeço à Deus que abriu tantas portas na minha vida e me deu forças para enfrentar as dificuldades nos momentos em que mais precisei.

Agradeço ao Professor Doutor Paulo Eduardo Santos que me deu essa oportunidade e acreditou no meu potencial durante todo o desenvolvimento do mestrado.

Agradeço aos meus familiares que estão ao meu lado desde sempre, com muito amor e incentivo.

Agradeço também ao Centro Universitário FEI e aos professores que me disponibilizaram recursos e ensinamentos.

Por fim, agradeço à FAPESP que, com a bolsa de estudos, permitiu que eu desenvolvesse este trabalho.

RESUMO

Raciocínio Diagramático vem sendo estudado por pesquisadores na investigação da cognição humana e na maneira com que diagramas são interpretados, tendo em vista que para os seres humanos o entendimento de representações diagramáticas costuma ser, intuitivamente, melhor compreendidos do que a representação textual. Baseado nos componentes essenciais da área de raciocínio diagramático, esse trabalho tem por objetivo a Prova de Correção por Indução Matemática de um algoritmo que combina *Answer Set Programming com Markov Decision Processes* na solução de problemas espaciais. Esta prova envolve a investigação de uma representação do conhecimento em diagramas, que são gerados automaticamente, a fim de garantir a possibilidade da construção física dos passos da solução de problemas espaciais compostos por objetos rígidos, cordas flexíveis e buracos.

Palavras-chave: Raciocínio Diagramático, Prova de Correção, Método da Indução Matemática.

ABSTRACT

Diagrammatic Reasoning has been studied by researchers in the investigation of human cognition and in the way that diagrams are interpreted, knowing that for human beings the understanding about diagrammatic representations are used to be better understood than a textual representation. Based on the Essentials components of the diagrammatic reasoning area, this work has the objective of a Correction Proof using the Mathematical Induction method of an algorithm that combines Answer Set Programming with Markov Decision Processes to solve spatial puzzles. This proof implicates in a investigation of a knowledge representation by using diagrams, that are automatically generated, in order to guarantee the possibility of a physical construction of the steps of the solution of spatial puzzles composed of rigid objects, flexible strings and holes.

Keywords: Diagrammatic Reasoning, Correction Proof, Mathematical Induction Method.

LISTA DE FIGURAS

Figura 1 – Teorema de Pitágoras representado por diagramas.....	15
Figura 2 - Princípio da Soma dos Números Naturais Ímpares.....	15
Figura 3 – Somatória Geométrica.....	15
Figura 4 – Teorema de Bolzano.....	16
Figura 5 – 14ª solução do Teorema de Pitágoras apresentada por Nelsen	17
Figura 6 – 15ª solução do Teorema de Pitágoras apresentada por Nelsen	18
Figura 7 – Representação do estado inicial do quebra-cabeças de Fisherman’s Folly	24
Figura 8 – Estado inicial do quebra-cabeças de Fisherman’s Folly	25
Figura 9 – Estado Final do quebra-cabeças de Fisherman’s Folly	25
Figura 10 – Solução do quebra-cabeças de Fisherman’s Folly	26
Figura 11 – Exemplo de Programa em ASP	27
Figura 12 – Programa positivo em ASP	29
Figura 13 – Exemplo de programa ASP com negação	29
Figura 14 – Relação entre Interpretação, Reduto e Modelo Proposicional da Figura 11	30
Figura 15 – Implicação em alfa	38
Figura 16 – Disjunção em alfa.....	39
Figura 17 – Exemplo grafo alfa	39
Figura 18 – Exemplo 1 de grafo beta.....	40
Figura 19 – Exemplo 2 de grafo beta.....	40
Figura 20 – Exemplo 3 de grafo beta.....	41
Figura 21 – Exemplo 4 de grafo beta.....	41
Figura 22 – Exemplo de desigualdade em grafos beta.....	42
Figura 23 – Exemplo de desigualdade na visão de Shin (2002).	42
Figura 24 – Exemplo 1 com condicionais.....	43
Figura 25 – Exemplo 2 com condicionais.....	44
Figura 26 – Exemplo 3 com condicionais.....	44
Figura 27 – Exemplo 1 de Raciocínio Não-Monotônico	45
Figura 28 – Exemplo 2 de Raciocínio Não-Monotônico	45
Figura 29 – Exemplo 1 de condicionais com linhas de identidade.....	46
Figura 30 – Exemplo 2 de condicionais com linhas de identidade.....	46
Figura 31 – Exemplo 3 de condicionais com linhas de identidade.....	47
Figura 32 – Exemplo de igualdade com linhas de identidade.....	47
Figura 33 – Leitura da lista com base na Figura 7.....	50

Figura 34 – Representação Diagramática do Estado Inicial desenhada pelo Software desenvolvido	53
Figura 35 – Estado Inicial no quebra-cabeças de Fisherman’s Folly real.....	54
Figura 33 – Simples ideia por trás da Indução Matemática (Efeito Dominó).....	56
Figura 37 – Representação Diagramática do Estado Inicial	61
Figura 38 – Estado Inicial no quebra-cabeças de Fisherman’s Folly.....	61
Figura 37 – Resultado da Ação 1 no Estado Inicial – Representação Diagramática	63
Figura 38 – Resultado da Ação 1 no Estado Inicial - Fisherman’s Folly	64
Figura 39 – Resultado da Ação 2 no Estado Inicial – Representação Diagramática	65
Figura 40 – Resultado da Ação 2 no Estado Inicial - Fisherman’s Folly	65
Figura 41 – Resultado da Ação 3 no Estado Inicial – Representação Diagramática	66
Figura 42 – Resultado da Ação 3 no Estado Inicial – Fisherman’s Folly	67
Figura 43 – Estado originador 5 – Representação Diagramática.....	70
Figura 44 – Estado originador 5 – Fisherman’s Folly	70
Figura 45 – Efeito da Ação 0 no Estado originador 5 – Representação Diagramática.....	72
Figura 46 – Efeito da Ação 0 no Estado originador 5 – Fisherman’s Folly	72
Figura 47 – Efeito da Ação 1 no Estado originador 5 – Representação Diagramática.....	73
Figura 48 – Efeito da Ação 1 no Estado originador 5 – Fisherman’s Folly	74
Figura 49 – Efeito da Ação 2 no Estado originador 5 – Representação Diagramática.....	75
Figura 50 – Efeito da Ação 2 no Estado originador 5 – Fisherman’s Folly	76
Figura 51 – Efeito da Ação 3 no Estado originador 5 – Representação Diagramática.....	76
Figura 52 – Efeito da Ação 3 no Estado originador 5 – Fisherman’s Folly	77
Figura 53 – Efeito da Ação 6 no Estado originador 5 – Representação Diagramática.....	78
Figura 54 – Efeito da Ação 6 no Estado originador 5 – Fisherman’s Folly	79
Figura 55 – Efeito da Ação 7 no Estado originador 5 – Representação Diagramática.....	79
Figura 56 – Efeito da Ação 7 no Estado originador 5 – Fisherman’s Folly	80
Figura 57 – Efeito da Ação 10 no Estado originador 5 – Representação Diagramática.....	81
Figura 58 – Efeito da Ação 10 no Estado originador 5 – Fisherman’s Folly	81
Figura 59 – Efeito da Ação 11 no Estado originador 5 – Representação Diagramática.....	82
Figura 60 – Efeito da Ação 11 no Estado originador 5 – Fisherman’s Folly	83

LISTA DE TABELAS

Tabela 1 – Representação dos elementos dos diagramas	53
Tabela 2 – Ações e Resultados Possíveis	59
Tabela 3 – Analogia entre Indução Matemática Numérica e Diagramática	62
Tabela 4 – Todas possibilidades de configurações dos elementos	68
Tabela 5 – Estados Originadores e os efeitos de cada ação.....	85

SUMÁRIO

1	INTRODUÇÃO	10
2	REVISÃO BIBLIOGRÁFICA	13
2.1	RACIOCÍNIO DIAGRAMÁTICO E INDUÇÃO MATEMÁTICA	13
2.2	SOLUÇÃO DE PUZZLES	19
3	REVISÃO TEÓRICA	22
3.1	DOMÍNIO E FORMALIZAÇÃO	22
3.2	ONLINE ANSWER SET PROGRAMMING PARA PROCESSO DE DECISÃO DE MARKOV (oASP(MDP))	27
3.2.1	Answer Set Programming (ASP)	27
3.2.2	MDP e Aprendizado por Reforço	30
3.2.3	Algoritmo Q-Learning	32
3.2.4	oASP(MDP)	33
3.2.5	oASP(MDP) para o Fisherman's Folly	35
3.3	GRAFOS EXISTENCIAIS	37
3.4	GRAFOS DE EQUILÍBRIO	43
4	SOFTWARE DESENVOLVIDO PARA CONSTRUIR DIAGRAMAS	49
5	PROVA DE CORREÇÃO DO oASP(MDP) POR INDUÇÃO MATEMÁTICA	55
5.1	MÉTODO DE INDUÇÃO MATEMÁTICA	55
5.2	MÉTODO DE INDUÇÃO MATEMÁTICA PARA O oASP(MDP)	57
6	CONCLUSÃO E TRABALHOS FUTUROS	86
	REFERÊNCIAS	88
	APÊNDICE A	92

1 INTRODUÇÃO

Neste trabalho é desenvolvida uma prova de correção utilizando o método da indução matemática (VELLEMAN, 1994) do algoritmo oASP(MDP) (FERREIRA, 2018). Esse algoritmo combina Answer Set Programming com Markov Decision Processes na solução de problemas espaciais. Esta prova compreende a investigação de uma representação de conhecimento em diagramas, a fim de garantir a possibilidade de construção física de problemas espaciais com domínio composto por objetos rígidos, cordas flexíveis e buracos, como por exemplo, o quebra-cabeças de Fisherman's Folly (CABALAR e SANTOS, 2011).

Os quebra-cabeças são eficazes para testes de soluções em Inteligência Artificial (IA) já que fornecem uma pequena quantidade de objetos, necessitando de um conhecimento base mínimo sobre recursos, enquanto preservam complexidade suficiente para constituir um problema desafiador de representação de conhecimento. Decorrente desta complexidade, a maioria dos trabalhos em raciocínio espaço-temporal tem focado em Raciocínio Espacial Qualitativo (*Qualitative Spational Reasoning*, QSR), como observado nos trabalhos de Stock (1997), Cohn e Renz (2008), Ligozat (2013). QSR é uma área de pesquisa que busca a formalização de conhecimento espacial baseado em relações primitivas definidas sobre entidades elementares espaciais. Essa característica é fundamental para lidar com um domínio que inclui as características do quebra-cabeças de Fisherman's Folly, que será utilizado como ferramenta de testes para a realização deste trabalho.

Em contrapartida, a área de Raciocínio sobre Ações e Mudanças (*Reasoning about Actions and Changes*, RAC), como visto no trabalho de McCarthy (1998), é especializada na formalização de raciocínio de senso comum em domínios dinâmicos e tem interesse na propriedade de tolerância à elaboração da representação de conhecimento, que expressa que a representação deve ser suficientemente flexível para acomodar novos fenômenos ou mudanças de circunstâncias. Com as ações e mudanças, aparecem problemas típicos de representação que devem ser solucionados, tais como: o problema do quadro (*frame problem*), que se refere à impossibilidade de especificar todos os atributos que se mantiveram inalterados após a execução de determinada ação; o problema de ramificação, que está relacionado

ao tratamento dos efeitos indiretos que as ações causam; ou o problema de qualificação, que lida com as precondições que determinam se uma ação é executável ou não.

Soluções encontradas para os problemas estudados pela área de RAC incluem algum tipo de Raciocínio Não-Monotônico (*Non-monotonic Reasoning*, NMR) (PEARCE, 95, 96), que é um modelo de inferência lógica não clássica onde a inserção de novas informações pode ocasionar a retração de uma conclusão obtida previamente, ou na dedução de fatos novos. Dada as características do domínio de quebra-cabeças espaciais, a aplicação de ações em busca da solução pode causar alteração nos estados e nas conclusões obtidas, tendo em vista que a consequência de uma ação no quebra-cabeças somente será verificada caso não haja restrições que a impeça de acontecer.

Tendo em vista o Raciocínio Não-Monotônico, este trabalho utiliza o algoritmo oASP(MDP) (Online Answer Set Programming para Processo de Decisão de Markov), proposto por Ferreira (2018). Esse algoritmo combina conceitos de uma linguagem de programação que se baseia em programação lógica e raciocínio não-monotônico (ASP (*Answer Set Programming*)), com uma área de estudos que permite ao computador o aprendizado na base de interações de um Agente com um Ambiente (Aprendizado por Reforço (*Reinforcement Learning*, RL)). Para esse aprendizado, o oASP(MDP) utiliza o algoritmo Q-Learning, que é capaz de encontrar uma política ótima para a solução de um MDP finito. Um MDP é formado por uma tupla que contém o conjunto de estados do domínio, o conjunto de ações possíveis de serem executadas, a função de transição e a função de recompensa. A descrição do algoritmo oASP(MDP) e a forma que ele busca os estados para solucionar o quebra-cabeças de Fisherman's Folly estão apresentados na seção 3.2.

Contudo, não há garantia de que os estados encontrados pelo algoritmo oASP(MDP), na busca pela solução do quebra-cabeças, são possíveis de serem construídos fisicamente. Esta incerteza motiva a realização de uma prova de correção pelo método da indução matemática.

O método da indução matemática foi adotado por ser capaz de provar fatos dado um número infinito de proposições. Por causa das iterações do algoritmo de solução, o número de possíveis estados a serem alcançados no quebra-cabeças de

Fisherman's Folly torna factível a utilização do método da indução matemática. Nesse método, primeiro é provado o comportamento das ações dado o estado inicial e, em seguida, é provado que, para qualquer estado encontrado, o comportamento do domínio é o mesmo para as ações executadas. Por não ser um domínio que apresente valores numéricos, é utilizada a representação diagramática para a aplicação do método da indução matemática para realizar a prova de correção do algoritmo oASP(MDP).

Baseando-se nos fundamentos dos Grafos Existenciais de Peirce (1882) e dos exemplos de provas de Brown (2008) e Nelsen (1993, 2015), a prova de correção por indução matemática está desenvolvida a partir de uma representação diagramática dos estados gerados pelo algoritmo oASP(MDP).

Portanto, neste trabalho, foi desenvolvido um método automático para tradução dos estados gerados pelo oASP(MDP), na solução do Fisherman's Folly, para diagramas. Estes diagramas serão base da prova de correção por indução matemática da solução proposta pelo algoritmo oASP(MDP), o que garante que os estados encontrados pelo algoritmo, em sua busca, são possíveis de serem construídos fisicamente no quebra-cabeças real, que é o objetivo deste trabalho.

Na seção 2 é apresentada uma revisão bibliográfica de trabalhos desenvolvidos na área de raciocínio diagramático, na utilização de diagramas para representações e na solução de puzzles. Na seção 3 são apresentados os conceitos empregados no desenvolvimento desse trabalho. A seção 4 apresenta uma descrição do software desenvolvido para a construção dos diagramas utilizados para a prova de correção do algoritmo oASP(MDP) com a utilização do método da indução matemática, que, por sua vez, está explicada na seção 5. Finalmente, na seção 6 estão apresentadas as conclusões obtidas com o desenvolvimento deste trabalho e sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção são descritos outros trabalhos que exploram o raciocínio diagramático, tanto em sua teoria quanto em representações (seção 2.1), e também que exploram a solução automática de puzzles (seção 2.2).

2.1 RACIOCÍNIO DIAGRAMÁTICO E INDUÇÃO MATEMÁTICA

Sloman (1998) indaga sobre o que acontece quando imaginamos diagramas ou imagens, sobre o que pode ser relacionado a problemas mais profundos sobre a visualização e manipulação espacial. Sloman (1998) cita um exemplo, que é a ação realizada pelo personagem Mr. Bean na tentativa de remover a roupa íntima sem tirar as calças, o que gera inúmeras questões. Primeiro se realmente é possível essa remoção, sem que a calça seja movida na região da cintura, apenas com o contorcionismo do personagem e a elasticidade do tecido. Depois, se é importante saber o quanto a calça é apertada. Questões como essas podem ser resolvidas por pessoas apenas pensando a respeito do problema, tentando visualizar o processo requerido, mas existem diversas formas de pensar sobre a ação em questão.

Para facilitar a ação a ser realizada pelo personagem citado por Sloman, é melhor pensarmos apenas na elasticidade do tecido da roupa íntima, como se alguém tentasse tirá-la para o Mr. Bean enquanto ele permanecesse imóvel. Buscando a solução, também é possível não focar nas características métricas como por exemplo os formatos e as medidas, concentrando-se apenas na topologia e encolhendo o corpo do personagem para uma esfera. Esse tipo de simplificação do problema será importante para a representação do quebra-cabeças de Fisherman's Folly que é utilizado neste trabalho, considerando que as características dos objetos que o compõem apresentariam inúmeras situações a serem ponderadas, contudo, mais detalhes sobre essas características estão apresentados na seção 3.1.

Quando se raciocina, seja por imagens, palavras, movimentos imaginados ou qualquer outro tipo de raciocínio, é normal processar uma sequência de estruturas

espaciais na qual a transição de um elemento da sequência para o próximo corresponde à um passo no raciocínio que pode ser representado por um diagrama.

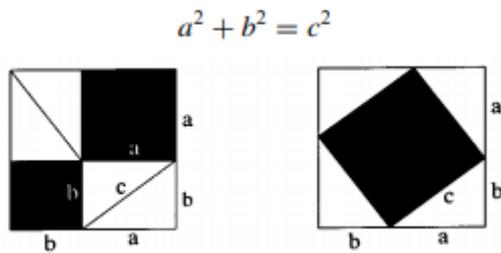
A importância desses diagramas fica evidente até mesmo pela melhor representatividade que isso tem para o ser humano. É uma representação extremamente intuitiva e um método que pode trabalhar juntamente com o texto para exemplificar ou até mesmo substituí-lo. A maior vantagem de um diagrama é que ele armazena informações precisamente e explicitamente representa as relações entre os elementos, dando suporte à muitas inferências perceptivas.

Os diagramas em explicações de teoremas e provas geométricas já eram usados na Grécia Clássica (NELSEN, 1993), porém, mais recentemente, com a invenção da lógica simbólica, os diagramas perderam esse papel formal na prova de teoremas e conceitos. Algo que vem sendo rebatido por pesquisadores na intenção de investigar como os diagramas podem ser utilizados em provas formais, como por exemplo a investigação dos Grafos Existenciais de Peirce (GEP) (1882). GEP é um sistema que permite a formulação de teorias em lógica de 1ª ordem com diagramas. A ideia por trás destes diagramas servirá como base para o desenvolvimento da representação dos diagramas, utilizados na prova de correção por indução matemática da solução, do quebra-cabeças de Fisherman Folly, gerada pelo algoritmo oASP(MDP), que é o objetivo deste trabalho.

Outros autores também apresentaram representações com diagramas, como por exemplo Nelsen (1993) com provas de teoremas como:

- O teorema de Pitágoras, onde a equação característica está representada por diagramas (Figura 1). O primeiro quadrilátero prova que os quadrados dos catetos a e b , quando somados, se equivalem ao quadrado da hipotenusa de lado c , como mostrado no segundo quadrilátero;
- Na Figura 2 observamos o Princípio da soma dos números naturais ímpares. Dado n números ímpares (representados pelos círculos preenchidos pretos e delimitados por linhas no diagrama) o valor da soma desses números equivale a este mesmo número n elevado ao quadrado;
- A Figura 3 contém a Somatória Geométrica, que prova que a somatória infinita de $1/2^n$ tende ao valor finito 1.

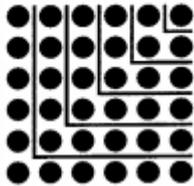
Figura 1 – Teorema de Pitágoras representado por diagramas.



Fonte: Nelsen, 1993

Figura 2 - Princípio da Soma dos Números Naturais Ímpares

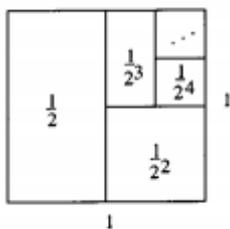
$$1 + 3 + \dots + (2n - 1) = n^2$$



Fonte: Nelsen, 1993

Figura 3 – Somatória Geométrica

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = 1$$

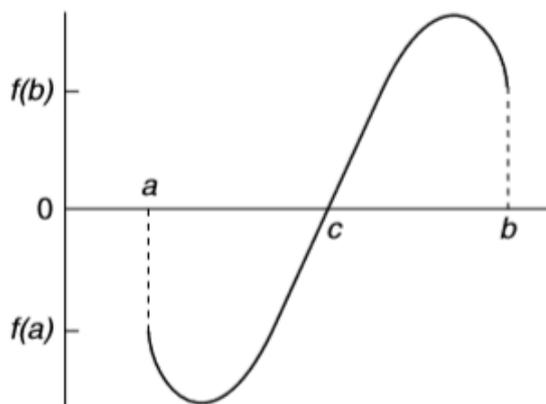


Fonte: Nelsen, 1993

Brown (2008) defende a utilização de diagramas como prova de teoremas matemáticos, afirmando que diagramas fornecem informações completas e que raciocinar sobre eles gera conclusões corretas sobre determinado assunto, mesmo que independente da prova analítica.

Brown (2008) também destaca a importância da utilização de imagens, afirmando que elas podem explicar teoremas. Como exemplo a explicação sobre o Teorema de Bolzano (Figura 4) onde, analisando apenas a imagem é possível concluir que, considerando uma função f contínua, compreendida entre o intervalo $[a, b]$, e f tem alteração no sinal (seja positivo para negativo ou vice versa), então existe um c entre a e b cuja sua função $f(c) = 0$.

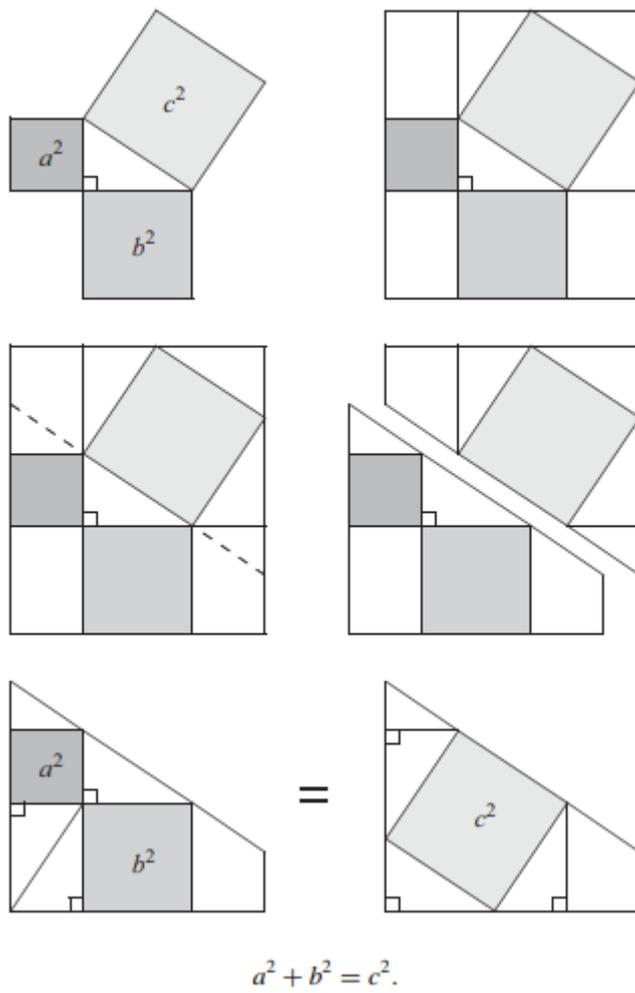
Figura 4 – Teorema de Bolzano



Fonte: Brown, 2008

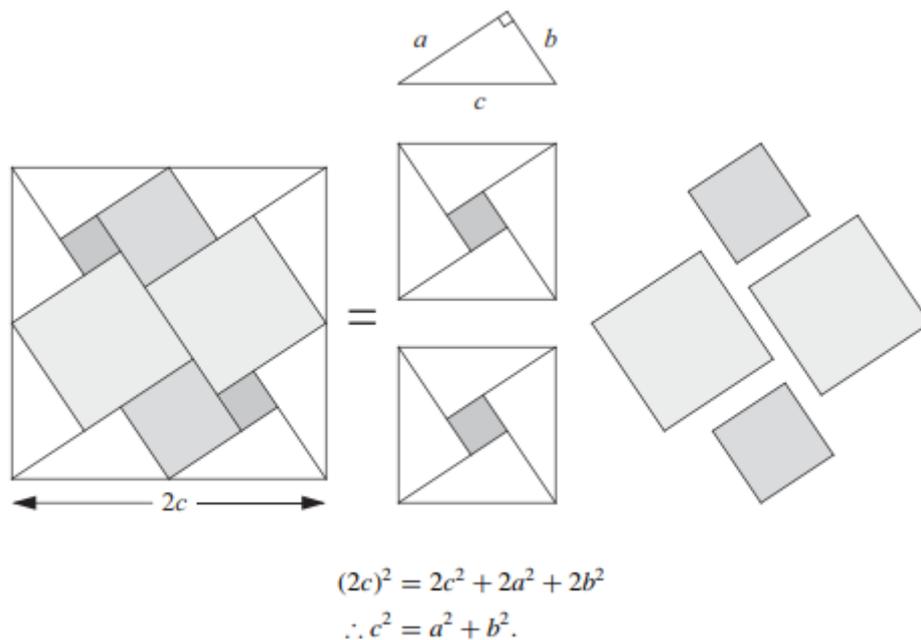
Nelsen (2015) apresenta sua mais recente versão de literatura composta por provas matemáticas (abrangendo tópicos de geometria, álgebra, séries matemáticas, entre outros) inteiramente com a utilização de diagramas. Dentre os exemplos apresentados na bibliografia de Nelsen (2015), é mostrado que existem diversas maneiras de solucionar, por exemplo, o Teorema de Pitágoras (como já visto na Figura 1). As alternativas enumeradas por Nelsen demonstram a extensão de possibilidades que os diagramas fornecem. Na Figura 5 e na Figura 6 é possível verificar mais duas soluções diagramáticas distintas para o Teorema de Pitágoras.

Figura 5 – 14ª solução do Teorema de Pitágoras apresentada por Nelsen



Fonte: Nelsen, 2015

Figura 6 – 15ª solução do Teorema de Pitágoras apresentada por Nelsen



Fonte: Nelsen, 2015

Velleman (1994) e Hammack (2013) apresentam a técnica de prova conhecida por método da indução matemática, que inicialmente foi designada para a prova de propriedades dos números naturais. No entanto, o método da indução matemática foi aplicado com a utilização de diagramas, que servem como base para a prova de correção da solução do Fisherman's Folly encontrada pelo oASP(MDP). Esse método da indução matemática está detalhado na seção 5.

De Toffoli e Giardino (2015) relatam como uma sequência de diagramas, quando utilizada em topologias de baixa dimensão, pode servir como prova matemática se seguida por regras, algo que é fundamental neste trabalho por causa da utilização de uma representação topológica simplificada do quebra-cabeças de Fisherman's Folly e a utilização de sequências de diagramas para o desenvolvimento da prova de correção com a utilização do método da indução matemática.

Há trabalhos que solucionam outros puzzles, no entanto, sem a utilização do oASP(MDP) e raciocínio diagramático, conforme apresentado na próxima seção.

2.2 SOLUÇÃO DE PUZZLES

A solução automática de problemas em geral surge na época clássica da Inteligência Artificial entre os anos 50 e 70, onde é desenvolvido o *General Problem Solver* (GPS) por Simon, Shaw e Newell (1959). Basicamente, o GPS era um programa que teve como propósito entender o processo de raciocínio humano para que houvesse uma aplicação em um mecanismo solucionador geral, que servisse para solucionar diversos problemas com a mesma ideia de raciocínio, utilizando ações e pré-condições. Em bibliografias clássicas da Inteligência Artificial como Russel e Norvig (2009) e Nilsson (1998) são detalhados métodos de resolução de problemas e jogos. Neste trabalho, no entanto, o foco está nos quebra-cabeças.

Exemplos de quebra-cabeças são apresentados por Kendall, Parkes e Spoerer (2008) que realizaram uma pesquisa sobre os quebra-cabeças do tipo NP-completo. NP é uma sigla em inglês para *Non-Deterministic Polynomial Time* (tempo polinomial não-determinístico) que diz respeito à complexidade computacional de solução, ou seja, um problema que possa ser solucionado em tempo polinomial por uma máquina de Turing não-determinística. Os problemas do tipo NP-Completo são caracterizados por não serem possíveis de se encontrar uma solução, por nenhum algoritmo, em tempo polinomial. Entretanto, existem soluções verificáveis em tempo polinomial para os problemas NP-Completo. Um dos quebra-cabeças apresentados (KENDALL G., PARKES A., SPOERER K., 2008) é o n -Puzzles que quando determinada a quantidade de peças, se torna um problema solucionável.

O n -Puzzle consiste em um quebra-cabeças com uma quantidade n de peças numeradas de 1 a n , e uma quantidade $n+1$ de espaços para que as peças possam ser deslocadas com o intuito de alinhá-las na ordem crescente dos números. Foram encontradas soluções para os quebra-cabeças de 8-puzzles até de 899-puzzle. Drogoul e Dubreuil (1993) solucionaram n -Puzzle com a utilização de agentes descritos pelo modelo *Eco-Problem-Solving*, que é baseado nos paradigmas da Inteligência Artificial Distribuída, ou seja, o problema foi dividido em conjuntos menores de subproblemas para o agente solucionar.

Outro puzzle citado por Kendall, Parkes e Spoerer (2008) é o Sudoku, onde são propostas soluções como a de Abdel-Raouf, El-henawy, Abdel-Baset (2014) que fazem a utilização de uma adaptação do algoritmo, motivado pela natureza, de polinização das flores. Entretanto, a solução proposta para o Sudoku por Eiter, et al. (2009) apresenta maior relação com este trabalho, do que a proposta da polinização das flores, por utilizar a linguagem de programação ASP (seção 3.2.1), fundamental para o desenvolvimento do oASP(MDP) (seção 3.2.4).

Cayli, et al. (2007) utilizam a linguagem de programação ASP, comparando os resultados obtidos dentre diferentes solucionadores como SMOBELS, CMOBELS e CLASP, para solucionar quatro quebra-cabeças de Nikoli: Nurikabe, Bag Puzzle, Masyu e Heyawake, que são quebra-cabeças japoneses que envolvem regras matemáticas e lógicas. Nikoli é uma empresa japonesa especializada no desenvolvimento de quebra-cabeças lógicos, entre os mais conhecidos no mundo está o Sudoku. No entanto, outros exemplos de quebra-cabeças lógicos podem ser utilizados como ferramenta de pesquisa, tendo em vista que fornecem desafios e restrições. Essa é uma característica semelhante ao quebra-cabeças estudado nesse trabalho, no entanto, o quebra-cabeças de Fisherman's Folly é espacial, o que significa que é composto por objetos que contém dimensões em 3D (como apresentado na seção 3.1).

O ASP também é utilizado para solucionar outros quebra-cabeças, como no trabalho de Mitra e Baral (2015), que é o primeiro sistema de aprendizagem a realizar uma solução inteiramente automática de *Logic Grid Puzzles*, que são quebra-cabeças lógicos onde as informações necessárias para a solução do problema são fornecidas na forma de texto. O trabalho desenvolvido realiza a interpretação do problema traduzindo o texto para a linguagem ASP (BARAL e DZIFICAK, 2012) e, assim, solucionando o quebra-cabeças com um solucionador. É visto neste trabalho uma importante aplicação do ASP, onde é possível a tradução da linguagem em forma de texto para a linguagem declarativa de ASP (seção 3.2.1).

El Khatib (2016) soluciona o *Pixel Puzzle* com a utilização da linguagem de programação ASP provando que, com a utilização do ASP, é possível encontrar resultados mais eficientes na solução desse quebra-cabeças, do que de outros programas específicos para o problema. O *Pixel Puzzle* também é um problema NP-

completo que é composto por uma tabela vazia com indicações numéricas que, ao serem preenchidas corretamente de acordo com as indicações, formam o desenho proposto no quebra-cabeças.

Conhecer a aplicação e eficiência do ASP é fundamental neste trabalho porque o algoritmo oASP(MDP), como dito anteriormente, utiliza essa linguagem de programação para encontrar a solução do quebra-cabeças estudado (Fisherman's Folly). Neste trabalho são aplicados os conceitos de raciocínio diagramático em uma prova por indução matemática, com o objetivo de realizar a prova de correção do algoritmo oASP(MDP). Na seção 3 encontram-se as descrições dos conceitos fundamentais para o desenvolvimento deste trabalho: quebra-cabeças Fisherman's Folly (seção 3.1); algoritmo oASP(MDP) (seção 3.2); Grafos Existenciais de Peirce e Grafos de Equilíbrio que serviram de inspiração para a representação diagramática e que servirão como base para trabalhos futuros (seção 3.3 e 3.4, respectivamente).

3 REVISÃO TEÓRICA

Nesta seção estão descritos os principais conceitos que serviram como base para este trabalho.

3.1 DOMÍNIO E FORMALIZAÇÃO

Foi escolhido o quebra-cabeças de Fisherman's Folly como domínio de investigação deste trabalho. Esse quebra-cabeças apresenta um domínio que contém objetos flexíveis e buracos, o que reproduz a complexidade de problemas espaciais tendo em vista que tanto as características dos objetos presentes no quebra-cabeça quanto a sua formalização apresentam desafios (CABALAR e SANTOS, 2011).

A solução deste quebra-cabeças consiste em separar o anel de um emaranhado de objetos rígidos, com buracos ou não. Assim, indo de um possível estado inicial (Figura 8) para um estado final (Figura 9). Os elementos que compõem o quebra-cabeças de Fisherman's Folly são: um poste perfurado ligado à uma base de madeira, uma corda, um anel, um par de esferas e um par de discos. Com relação à composição e mobilidade desses objetos, tem-se a corda cruzando o buraco do poste fazendo com que cada esfera e cada disco fiquem dispostos de um lado do poste. Os discos e as esferas são vinculados à corda, sendo que os discos são fixados em suas extremidades e as esferas são deslizantes. Os discos podem atravessar o buraco do poste, alternando as configurações do quebra-cabeças para que seja possível a retirada do anel. No entanto, o diâmetro da esfera não permite que ela possa atravessar para o outro lado do poste. Com relação ao anel, seu diâmetro permite que ele atravesse a esfera e o poste, tanto pelo buraco quanto por cima, porém não permite que atravesse o disco na tentativa de alcançar a solução representada pelo estado final (Figura 9). Há de se ponderar que as inúmeras possibilidades de movimentação entre os objetos podem gerar nós e complicar a convergência na solução.

As propriedades dos elementos que compõem esse quebra-cabeças são distintas, o que implica na possibilidade de classificá-los em três diferentes categorias:

objetos Regulares (discos), objetos com Buracos (anel, esferas e poste) e objetos Longos (corda e poste).

Os objetos do domínio são chamados de: *Disk1*, *Disk2* (referentes aos objetos regulares, isto é, os discos); *Base* (indicando a base do quebra-cabeças); *Ring*, *Sphere1*, *Sphere2*, *PostH* (representando os objetos com buracos, sendo, respectivamente: anel, esferas e poste); e *Str* e *Post* (referentes aos objetos longos, que são, respectivamente, a corda e o poste) (CABALAR e SANTOS, 2011).

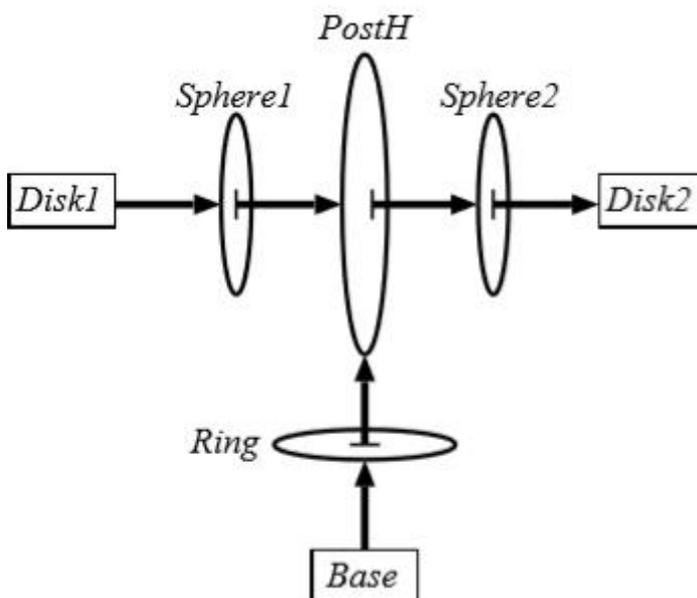
Cada estado necessita ser descrito para o entendimento e construção da representação diagramática para prova de correção por indução matemática. Com base na Figura 7 é possível descrever a formalização do quebra-cabeças de Fisherman's Folly. As setas são correspondentes à segmentos de um objeto longo, compreendidos entre pares de objetos com buracos ou um objeto com buraco de um disco ou base. As setas são direcionadas e percorrem, partindo da extremidade negativa, a corda (setas horizontais) e o poste (setas verticais), até alcançar a extremidade positiva. As elipses correspondem aos objetos com buracos, enquanto os retângulos representam os objetos regulares. A face dos objetos com buracos na qual a seta direcionada tem início é definida como a face positiva.

Partindo da ponta negativa de cada objeto longo, são descritas duas listas de dados que representam a sequência de cruzamento dos objetos. Cada lista é chamada de $chain(X)$, sendo X a variável para cada objeto longo. Tendo como exemplo a Figura 7, as duas listas são representadas como:

- $chain(Str) = [+Sphere1, +PostH, +Sphere2];$
- $chain(Post) = [+Ring].$

Em $chain(Str)$ é afirmado que a corda atravessa o buraco através da face positiva da esfera 1, através da face positiva do poste e também através da face positiva da esfera 2. Enquanto em $chain(Post)$, por sua vez, o poste atravessa o buraco do anel.

Figura 7 – Representação do estado inicial do quebra-cabeças de Fisherman's Folly



Fonte: Cabalar e Santos, 2011

A Figura 10 é meramente ilustrativa, porém representa a solução diagramática ótima do quebra-cabeças de Fisherman's Folly, isto é, a maneira mais eficiente de alcançar o objetivo de retirar o anel do conjunto de objetos. O primeiro diagrama (Figura 10a, estado S0) seria a representação do estado apresentado na Figura 8 enquanto o último diagrama (Figura 10f, estado S5) seria a representação da solução do quebra-cabeças. A Figura 10 também mostra outros quatro diagramas de estados intermediários (Figura 10b, Figura 10c, Figura 10d, Figura 10e) para alcançarmos a solução do quebra-cabeças de Fisherman's Folly.

O algoritmo oASP(MDP) (explicado na seção 3.2) descreve cada estado encontrado durante a busca e, partindo dessa descrição, os diagramas (como os intermediários da Figura 10) serão construídos como resultado do presente trabalho. A representação dos estados com diagramas é estruturada nos princípios dos Grafos Existenciais de Peirce (1882) (explicado na seção 3.3). Com essa representação diagramática é possível realizar a prova de correção por indução matemática.

Figura 8 – Estado inicial do quebra-cabeças de Fisherman's Folly



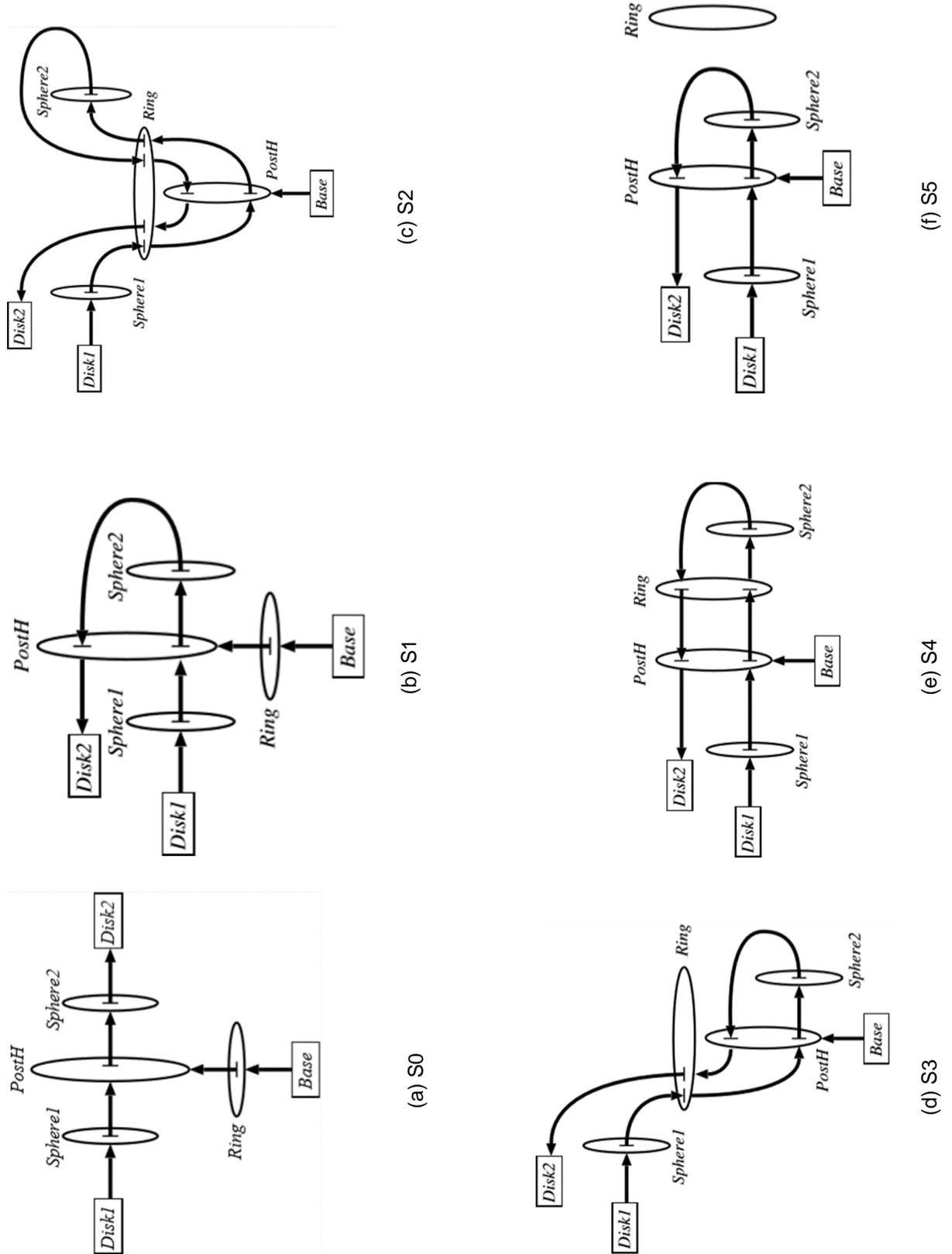
Fonte: Cabalar e Santos, 2009

Figura 9 – Estado Final do quebra-cabeças de Fisherman's Folly



Fonte: Cabalar e Santos, 2009

Figura 10 – Solução do quebra-cabeças de Fisherman’s Folly



Fonte: Autor “adaptado de” Cabalar e Santos, 2011

3.2 ONLINE ANSWER SET PROGRAMMING PARA PROCESSO DE DECISÃO DE MARKOV (oASP(MDP))

Utilizando os conceitos de *Answer Set Programming* (ASP) e Aprendizado por Reforço (*Reinforcement Learning*, RL), foi proposto por Ferreira, et al. (2018) a técnica de Online Answer Set Programming para processo de decisão de Markov (oASP(MDP)). Esses conceitos utilizados pelo oASP(MDP) estão descritos abaixo.

3.2.1 Answer Set Programming (ASP)

ASP é uma linguagem que utiliza o padrão declarativo de programação, baseado em programação lógica e raciocínio não-monotônico, para solução de problemas (GELFOND e LIFSCHITZ, 1988). Esse padrão declarativo faz com que essa linguagem seja útil para resolver problemas de busca, como o quebra-cabeças de Fisherman's Folly.

O ASP baseia-se em modelos estáveis, que são gerados por um solucionador para a realização da busca. Um exemplo de um programa em ASP está apresentado na Figura 11 abaixo, onde são declarados tipos de *bird* (*tweety* e *woody*), um tipo de *penguin* (*tweety*) e, em seguida, é dada a condição que indica se o *bird* (pássaro) voa (*flies*) ou não:

Figura 11 – Exemplo de Programa em ASP

```
bird(tweety).
bird(woody).
penguin(tweety).
flies(X) :- bird(X), not ab(X).
ab(X) :- penguin(X).
```

Fonte: Cabalar, 2017

Para este programa o ASP encontra duas soluções possíveis:

```
{bird(tweety). bird(woody). penguin(tweety).}
{ab(tweety). flies(woody).}
```

Como podemos perceber pelo exemplo da Figura 11, ao escrever um programa em ASP, são definidas cláusulas de Horn (LIFSCHITZ, 2008). O símbolo “:-“ representa uma implicação, à esquerda desse símbolo está um átomo, e à direita se encontra o conjunto de literais que são separados por vírgulas que, por sua vez, representam conjunção lógica. O símbolo “not” representa a negação por falha, também chamada de negação fraca. O símbolo “-“ (não utilizado pelo exemplo da Figura 11) representa a negação forte. A negação por falha é considerada fraca porque a aplicação da regra é permitida quando não se tem informações sobre a condição a ser negada, ou seja, quando queremos aplicar alguma regra apenas a partir da certeza de que a condição foi negada, devemos utilizar a negação forte.

Em ASP é possível a utilização das cláusulas de Horn sem a cabeça, utilizando essas fórmulas para representar restrições, isto é, quando deseja-se excluir da execução de um programa algumas soluções que poderiam ser encontradas. Também é importante ressaltar que a ordem para a escrita das regras em ASP é irrelevante, sendo que a regra será positiva nos casos onde não houverem negação.

Os programas em ASP, que não possuem negação, podem ser computados apenas com a aplicação da regra do operador de consequência imediata (Tp), ou seja, por iteração do programa são verificados os átomos que são provados dados os seus literais. Nos programas que não possuem negação, o operador de consequência imediata converge para um ponto fixo, que seria o conjunto de respostas (Answer Set) do programa. No exemplo da Figura 12 temos um programa positivo em ASP, onde pode ser realizada a inferência pelo operador de consequência imediata:

Figura 12 – Programa positivo em ASP

```

p
q
r ← p,s      s ← q      b ← s,a
              a ← b,p      a ← c

```

Fonte: Cabalar, 2017

Com o Tp inicial apenas conseguimos ter como verdade os átomos p e q :

$$Tp(\emptyset) = \{p, q\}.$$

Em um segundo momento, partindo da interpretação (conjunto de átomos) verdadeira, é possível provar s (consequência imediata de q):

$$Tp(\{p, q\}) = \{p, q, s\}.$$

Continuando a verificação, porém com a nova interpretação onde s também é verdadeiro, conseguimos provar r (consequência imediata de p e s):

$$Tp(\{p, q, s\}) = \{p, q, s, r\}.$$

Por fim, a última iteração leva a um ponto fixo (Answer Set do programa), tendo em vista que com esse modelo não há mais consequências imediatas nesse programa:

$$Tp(\{p, q, s, r\}) = \{p, q, s, r\}.$$

O Answer Set, obrigatoriamente, faz parte do ponto fixo do Tp .

Nos casos em que o programa contém uma parte negativa é necessário fazer a redução do ASP dada uma interpretação. O programa reduzido é positivo e chamado de P^I . Vide exemplo de programa com negação abaixo:

Figura 13 – Exemplo de programa ASP com negação

```

drinks ← german, not ab
german

```

Fonte: Cabalar, 2017

Dada as três possíveis interpretações que o programa da Figura 13 ($\{german, ab\}$; $\{german, ab, drinks\}$; $\{german, drinks\}$) é possível inferir o programa reduzido e o modelo estável:

Figura 14 – Relação entre Interpretação, Reduto e Modelo Proposicional da Figura 13

I	P^I	$LM(P^I)$
$\{german, ab\}$	<i>german</i>	$\{german\} \neq I$ not stable
$\{german, ab, drinks\}$	<i>german</i>	$\{german\} \neq I$ not stable
$\{german, drinks\}$	<i>drinks</i> ← <i>german</i> <i>german</i>	$\{german, drinks\}$ stable!

Fonte: Cabalar, 2017

Apenas a última interpretação da Figura 14 gera um modelo estável, porque por definição, a interpretação é um modelo estável de um programa caso seja igual ao modelo proposicional.

Em programas que possuem negação, o modelo estável só é considerado um Answer Set caso contenha a negação forte.

Com o ASP é possível realizar a modelagem de um Processo de Decisão de Markov (*Markov Decision Process*, MDP) (FERREIRA et al., 2018). Na seção 3.2.2 é detalhado o conceito que possibilita a solução do programa lógico em ASP e, com isso, as políticas que solucionam o MDP.

3.2.2 MDP e Aprendizado por Reforço

Um dos métodos de Aprendizado de Máquina é conhecido por Aprendizado por Reforço (RL) que pode ser definido como o problema de aprendizagem através de interações com o domínio a fim de atingir um determinado objetivo (SUTTON E BARTO, 2018).

O Agente é o componente responsável pelo aprendizado e pela execução de ações em diferentes instantes no Ambiente. O Ambiente, por sua vez, responde ao Agente com a recompensa das ações que foram tomadas e com estados futuros para que, em instantes subsequentes, o Agente adote novas ações.

No processo de tomada de decisões, o agente tende a optar por ações que, previamente executadas, resultaram em uma recompensa maior. No entanto, na busca por essas ações, o Agente deve explorar o ambiente e testar diversas ações. Faz-se fundamental a determinação de um período para que o Agente interaja com o Ambiente, uma sugestão é iniciar com alta taxa de exploração e, com o tempo, aumentar a utilização de ações que já se conhece o resultado.

Sutton e Barto (2018) também apresentam outros componentes que podem fazer parte de um sistema de RL: Função de Recompensa; Função Valor; Modelo do Ambiente; Política.

A Função de Recompensa é encarregada de mapear os estados observados pelo agente a fim de gerar um valor numérico, que indica o ganho do Agente em determinado estado.

A Função Valor estabelece as vantagens a longo prazo para o Agente, calculando o valor total das recompensas que o Agente, a partir de determinado estado, poderá obter em todos os estados futuros.

O Modelo do Ambiente simula o comportamento do ambiente, indicando estados e recompensas futuras baseando-se em um estado e uma ação.

A Política mapeia estados atingidos pelo agente dadas as ações que serão executadas nesse mesmo estado.

Esse sistema de decisão sequencial utilizado pelo Agente trata-se de um Processo de Decisão Markoviano (*Markov Decision Process*, MDP), onde há a utilização de um ambiente totalmente observável com um modelo de transição Markoviano. Um MDP pode ser representado pela tupla $\langle S, A, T, R \rangle$, onde:

- S: conjunto de estados observáveis no domínio;
- A: conjunto de ações executáveis por um Agente;

- T: função de transição que permite ao Agente atingir um estado futuro dado um estado atual e uma ação;
- R: função de recompensa que recompensa o Agente por alcançar um estado futuro, dado um estado atual e uma ação.

São discutidos por Sutton e Barto (2018) três métodos diferentes para aprender a política ótima de um MDP: Programação Dinâmica, Métodos de Monte Carlo e Aprendizado por Diferença Temporal. No entanto, para encontrar a política ótima do MDP do oASP(MDP) no trabalho de Ferreira et al. (2018), foi utilizado um método de Aprendizado de Diferença Temporal, mais especificamente, o algoritmo Q-Learning.

3.2.3 Algoritmo Q-Learning

Um dos mais populares algoritmos para a realização de aprendizagem por reforço, o Algoritmo Q-Learning foi escolhido porque ele faz com que o Agente aprenda de acordo com as interações com o Ambiente, fazendo atualização de valores a cada passo executado, sem necessitar aguardar o fim de um episódio. Com este algoritmo não há a necessidade de conhecer o modelo do ambiente, a recompensa e as distribuições de probabilidades dos estados futuros *a priori*.

A atualização da função correspondente aos valores das ações é feita a partir da fórmula da equação abaixo:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

$Q(S_t, A_t)$ corresponde à função valor-ação, onde é atualizado o estado s executando uma ação a . O fator de desconto (γ) é utilizado como garantia de que os valores de Q sejam finitos, e α representa a taxa de aprendizagem.

Na utilização do algoritmo Q-Learning só é necessário conhecer o conjunto de estados (S) e o conjunto de ações (A), não há necessidade de informar sobre a função de transição (T) e a função de recompensa (R). A combinação com o ASP se mostra

vantajosa porque, com essa programação, é possível encontrar o conjunto de estados e ações.

3.2.4 oASP(MDP)

A ideia do online Answer Set Programming para Processo de Decisão de Markov (FERREIRA et al., 2018) consiste na utilização da combinação das vantagens dos métodos ASP e Q-Learning apresentados nas seções 3.2.1 e 3.2.3.

Primeiro é feita a modelagem de um MDP utilizando um programa lógico (em ASP), para que seja possível a busca por um modelo estável durante a exploração do espaço de estados. Com isso, ao encontrar as soluções para um programa lógico, também se encontram as políticas que solucionam o MDP.

O ASP possibilita ao algoritmo de aprendizado a realização de revisões nos conjuntos de estados e ações, o que implica em uma alteração na formalização do MDP, sem que haja a necessidade de uma reinicialização no processo de aprendizado.

A combinação de Q-Learning com ASP possibilita a definição de regras de escolha para descrever uma transição $t(s, a, s')$ na sua forma lógica conforme a equação abaixo, que representa que, caso uma ação a seja executada, ocorrerá uma transição para somente um estado novo s' :

$$1 \{s'\} 1 : - a$$

Essa descrição pode ser aplicada para qualquer ação e estado que pertença ao domínio. Consequentemente, é possível a utilização de um solucionador ASP para a obtenção de um conjunto de ações e estados observáveis para que, a partir das interações entre agente e ambiente, seja definida a função valor-ação $Q(s, a)$.

No Algoritmo 1 está o pseudocódigo do algoritmo oASP(MDP) (FERREIRA, 2018).

Algoritmo 1 – Pseudocódigo do oASP(MDP)

```

1 Algoritmo: oASP(MDP)
2 Entrada: O conjunto de ações  $\mathcal{A}$ , um método de aproximação da função
   valor-ação  $M$  e o número de episódios  $n$ .
3 Saída: A função aproximada  $Q(s,a)$ .
4 Inicialize o conjunto de estados observados  $\mathcal{S}_o = \emptyset$ 
5 enquanto número de episódios for menor que  $n$  faça
6   repita
7     Observe o estado atual  $s$ 
8     se  $s \notin \mathcal{S}_o$  então
9       Adicione  $s$  ao conjunto de estados  $\mathcal{S}_o$ .
10      Escolha e execute uma ação aleatória  $a \in \mathcal{A}$ .
11      Observe o estado futuro  $s'$ .
12      Atualize o estado  $s$  adicionando ao programa lógico a transição observada
        através da regra de escolha correspondente.
13      Atualize a descrição de  $Q(s,a)$  encontrando cada answer set para cada
        estado  $s$  adicionado a  $\mathcal{S}_o$  neste episódio.
14     fim
15     senão
16       Escolha uma ação  $a \in \mathcal{A}$  definida por  $M$ .
17       Execute a ação escolhida  $a$ .
18       Observe o estado futuro  $s'$ .
19     fim
20     Atualize o valor de  $Q(s,a)$  a partir da definição encontrada por  $M$ .
21     Atualize o estado atual  $s \leftarrow s'$ .
22   até fim do episódio
23 fim

```

Fonte: Ferreira, 2018

O algoritmo oASP(MDP) recebe três parâmetros diferentes de entrada. O primeiro parâmetro corresponde ao conjunto de ações que o agente pode executar no estado inicial; o segundo parâmetro representa um método M de RL, que no caso seria o algoritmo Q-Learning, responsável pela aproximação da função valor-ação $Q(s,a)$; como último parâmetro de entrada, o algoritmo oASP(MDP) recebe uma quantidade pré-definida n de episódios que devem ser executados:

- conjunto de ações que podem ser executadas dado o estado inicial;
- método M de RL (Q-Learning), que pode aproximar a função valor-ação $Q(s, a)$;
- quantidade n de episódios que devem ser executados.

Dada as entradas, o algoritmo inicializa o conjunto de estados (S) como vazio, tendo em vista que o conjunto será construído conforme as interações do Agente com o Ambiente, ainda aproximando a função valor-ação $Q(s, a)$.

Subsequente as definições iniciais informadas como parâmetro de entrada, o algoritmo entrará em um laço de repetição que será executado n vezes. Em cada episódio o algoritmo toma uma decisão baseada no estado em que se encontra. Caso seja um estado que não está incluído em S , este passa a fazer parte do conjunto de estados e uma ação aleatória é tomada, retornando um estado futuro. Caso o estado já faça parte de S , então a escolha da ação é realizada pelo método M (Q-Learning) e não há atualização das regras de escolha. Finalmente, a função valor-ação $Q(s, a)$, baseando-se no método M , é n vezes atualizada. Em Ferreira (2017) é possível observar um exemplo prático aplicado ao mundo de grades.

3.2.5 oASP(MDP) para o Fisherman's Folly

Para a utilização do oASP(MDP) no quebra-cabeças de Fisherman's Folly foi realizada uma adaptação por Freitas (2018). Primeiro foram alterados os parâmetros: conjunto de ações; conjunto de estados composto por estados já conhecidos (o algoritmo base (oASP(MDP)) não recebe os estados conhecidos). A saída do algoritmo se mantém obtida com a aplicação do Q-Learning, no entanto, a parte do ASP foi alterada na representação de cada estado e ação em linguagem lógica. A alteração se deu na representação da tupla que, tendo em vista que as ações possíveis correspondem às passagens de objetos por buracos, passou a ser representada como $\langle HE, CE, HF \rangle$, onde:

HE *Host Element*: um elemento do domínio que possui buraco (identificado pelo próprio nome do objeto):

$$HE \in \{Sphere1, Sphere2, Post, Disk1, Disk2, Ring, Str\}$$

CE *Crossing Element*: um elemento do quebra-cabeças que atravessará um buraco de um elemento HE . A Str também faz parte desse conjunto porque suas pontas estão fixadas em $Disk1$ e $Disk2$, e a sua manipulação a faz atravessar elementos.

$CE \in \{Sphere1, Sphere2, Post, Disk1, Disk2, Ring, Str\}$

HF *Hole Face*: corresponde ao lado em que o objeto que está atravessando um buraco está indo em direção. A escolha do lado depende do agente durante a execução da ação. São duas faces possíveis: positivo (+) ou negativo (-).

A tupla $\langle HE, CE, HF \rangle$ é lida como: HE está atravessando CE em direção à face HF (FREITAS, 2018). A escolha de uma ação, considerando este cenário da tupla, determina a manipulação dos elementos pelo agente, que, neste ponto, ainda não tem conhecimento sobre quais objetos possuem buracos ou são regulares. A falta de conhecimento do agente, no estado inicial, faz com que haja 7 elementos manipuláveis nos conjuntos HE e CE, além de duas faces em HF, o que totaliza 98 ações possíveis.

O aprendizado sobre as características do domínio ocorre no decorrer das interações do agente com o ambiente através de recompensas negativas para ações que, pelas propriedades dos objetos do quebra-cabeças, são fisicamente impossíveis de serem realizadas, por exemplo, a *Sphere1* não consegue atravessar fisicamente o *PostH* e o *Disk1* não consegue atravessar fisicamente o *Ring*, entre outras restrições (seção 3.1). O oASP(MDP) aprende quais são as ações que não podem ser realizadas na prática e as exclui do conjunto de ações.

Os estados do domínio utilizam a formalização apresentada na seção 3.1, com a construção de uma lista para os elementos atravessados pela *Str* (corda) e uma lista para os elementos atravessados pelo *Post* (poste).

Após a obtenção da representação dos estados e das ações, o algoritmo oASP(MDP) recebe o conjunto de ações A , o conjunto de estados S e o número de episódios n para sua inicialização.

Em um primeiro momento, o estado inicial (s_0) não estará contido no conjunto de estados (S), mesmo assim o primeiro passo dado pelo algoritmo é essa verificação. Com o resultado negativo da presença do estado inicial no conjunto de estados, o agente executará uma ação aleatória e irá alcançar um estado s_1 , o estado s_0 poderá ser adicionado ao conjunto de estados e o agente recebe uma recompensa r_0 . Dado o primeiro passo, o agente passa a conhecer um possível estado e a transição que

levou até ele, sendo possível a criação de uma regra de escolha em ASP, onde cada ação executada resulta em somente um estado novo:

$$1\{s1\}1 : - a(AçãoExecutada)$$

O termo “AçãoExecutada” representa a ação executada pelo agente. Essa regra é importante porque, a partir dela, o ASP consegue encontrar os *answer sets*, possibilitando ao agente iniciar a função valor-ação $Q(s, a)$ e atualizar o valor inicial com a recompensa r_0 obtida.

Quando o agente se encontrar em um estado já conhecido, a ação executada será determinada pelo algoritmo Q-Learning. As interações são repetidas de acordo com a quantidade de episódios (n) determinada, atualizando o valor da função valor-ação $Q(s, a)$ com a recompensa oriunda do ambiente.

Fundamentado pelo oASP(MDP), neste trabalho é feita uma prova de correção por indução matemática a fim de garantir que os estados encontrados pelo algoritmo são, de fato, possíveis de serem construídos fisicamente no quebra-cabeças de Fisherman’s Folly.

A solução encontrada pelo oASP(MDP) equivale aos estados do quebra-cabeças, representados através de uma lista correspondente à disposição dos objetos do domínio (as listas são detalhadas nas seções 4 e 5). Os estados encontrados pelo oASP(MDP), como solução do quebra-cabeças, passam pelo método desenvolvido neste trabalho para que possam ser diagramaticamente representados para o embasamento da prova de correção. Este método consiste na tradução da lista das disposições dos objetos do domínio em diagramas.

Na seção 3.3 são descritos os Grafos Existenciais de Peirce (1882), que fundamentam o desenvolvimento dos diagramas utilizados para a realização da prova de correção pelo método da indução matemática (seção 5). Os Grafos Existenciais de Peirce também orientam o desenvolvimento de um solucionador automático para o Fisherman’s Folly que seja inteiramente diagramático, entretanto, esse tratamento será realizado em trabalhos futuros.

3.3 GRAFOS EXISTENCIAIS

Charles Sanders Peirce propôs em 1882 uma forma diagramática para expressões lógicas. Essa notação visual é um sistema que faz com que a lógica seja simples e intuitiva de ser compreendida.

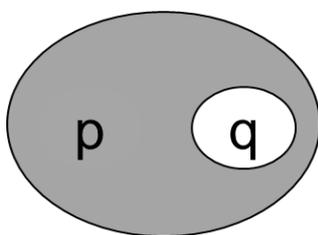
Grafos Existenciais de Peirce (1882) podem ser classificados em três categorias: *alfa*, *beta* e *gama*. Essas categorias correspondem respectivamente ao Cálculo Proposicional, Lógica de primeira ordem com igualdade e um tipo de lógica modal.

Um diagrama na categoria *alfa* pode representar:

- A verdade, quando for um diagrama vazio;
- Sentenças atômicas;
- A negação do que estiver envolto por um círculo fechado (também chamado de corte). Um corte vazio representa falsidade;
- Conjunções, que é a inserção de vários elementos dentro da mesma região.

Por exemplo, as figuras 15, 16 e 17 representam como podem ser utilizados os grafos *alfa*. O uso de conjunção e negação como operadores primitivos podem facilmente representar implicação e disjunção como, respectivamente, apresentados na Figura 15 e na Figura 16.

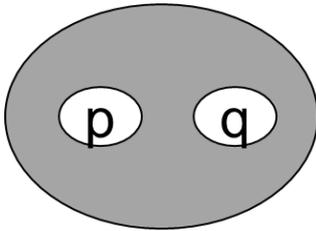
Figura 15 – Implicação em alfa



$$[p \rightarrow q] \equiv [\neg(p \wedge \neg q)]$$

Fonte: Autor "adaptado de" Pérez, 2017

Figura 16 – Disjunção em alfa



$$[p \vee q] \equiv [\neg p \wedge \neg q]$$

Fonte: Autor “adaptado de” Pérez, 2017

A Figura 17 representa a fórmula:

$$\neg(\text{chover} \wedge \neg\text{guarda-chuva} \wedge \neg\text{molhada})$$

Que também pode ser vista como a implicação:

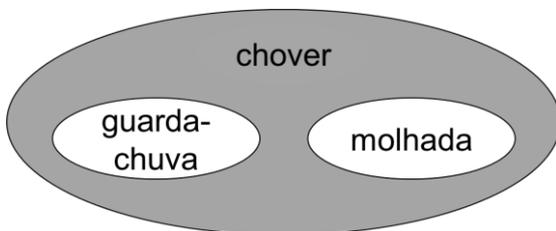
$$\text{chover} \vee \neg\text{guarda-chuva} \rightarrow \text{molhada} \text{ ou } \text{chover} \wedge \neg\text{molhada} \rightarrow \text{guarda-chuva}$$

Ou a disjunção:

$$\neg\text{chover} \vee \text{guarda-chuva} \vee \text{molhada}$$

Considerando que todas são equivalentes na lógica proposicional clássica.

Figura 17 – Exemplo grafo alfa



Fonte: Autor “adaptado de” Pérez, 2017

Para a representação de expressões de primeira ordem, os grafos *alfa* foram aprimorados para os grafos *beta* com a inclusão de um novo componente chamado de *linha de identidade*, que é representado por uma linha que conecta um ou mais nomes, criando quantos ramos forem necessários para fazer as ligações. A *linha de*

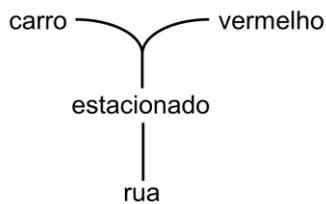
identidade é um quantificador existencial, portanto sua leitura deve ser feita associando que existe algo que está relacionado a algo.

As figuras a seguir exemplificam a utilização dos grafos *beta*.

A Figura 18 declara que existe um carro vermelho estacionado na rua:

$$\exists x \exists y(\text{carro}(x) \wedge \text{vermelho}(x) \wedge \text{estacionado}(x,y) \wedge \text{rua}(y))$$

Figura 18 – Exemplo 1 de grafo beta

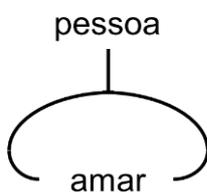


Fonte: Autor “adaptado de” Pérez, 2017

A Figura 19 diz que alguma pessoa ama a si própria:

$$\exists x(\text{pessoa}(x) \wedge \text{amar}(x,x))$$

Figura 19 – Exemplo 2 de grafo beta



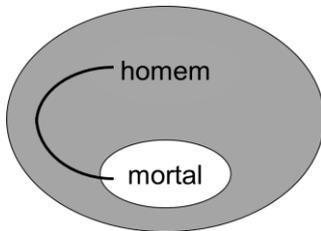
Fonte: Autor “adaptado de” Pérez, 2017

A Figura 20 afirma que todo homem é mortal, pode ser representado de duas formas:

$$\neg \exists x(\text{homem}(x) \wedge \neg \text{mortal}(x))$$

$$\forall x(\text{homem}(x) \rightarrow \text{mortal}(x))$$

Figura 20 – Exemplo 3 de grafo beta

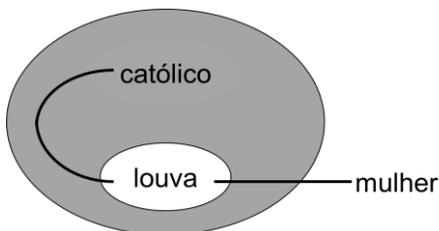


Fonte: Autor “adaptado de” Pérez, 2017

A Figura 21, por sua vez, especifica que existe uma mulher louvada por todos católicos:

$$\exists x(\text{mulher}(x) \wedge \forall y(\text{católico}(y) \rightarrow \text{louva}(y,x))).$$

Figura 21 – Exemplo 4 de grafo beta



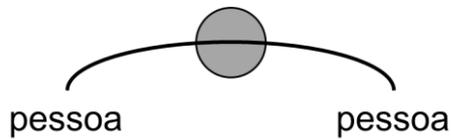
Fonte: Autor “adaptado de” Pérez, 2017

Uma observação importante é que os grafos *beta* não fornecem um método específico para a representação de constantes, por exemplo, não há uma maneira de expressar que “todo católico louva à Virgem Maria” que não seja utilizando o predicado para designar especificamente “Maria” ao invés de resumir à “mulher”.

Também pode-se ver as *linhas de identidade* como uma característica de igualdade, conseqüentemente, ao atravessar um corte com uma linha de identidade podemos representar desigualdade, como exemplificado pela Figura 22 abaixo, onde afirma-se que todas pessoas são diferentes, ou:

$$\exists x \exists y(\text{pessoa}(x) \wedge \text{pessoa}(y) \wedge x \neq y).$$

Figura 22 – Exemplo de desigualdade em grafos beta

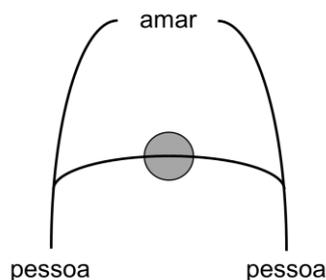


Fonte: Autor “adaptado de” Pérez, 2017

Ainda não há um consenso sobre como devem ser tratadas as desigualdades quando uma *linha de identidade* apresenta ramificações. Porém, para melhor aplicabilidade, neste trabalho, será seguida a linha de pensamento defendida por Shin (2002), e para exemplificar, a Figura 23 declara que duas pessoas diferentes se amam, ou:

$$\exists x \exists y(\text{pessoa}(x) \wedge \text{pessoa}(y) \wedge \text{amar}(x,y) \wedge x \neq y).$$

Figura 23 – Exemplo de desigualdade na visão de Shin (2002).



Fonte: Autor “adaptado de” Shin, 2002

Com essas representações é possível introduzir uma extensão dos Grafos Existenciais chamada de Grafos de Equilíbrio proposta por Cabalar, Pérez e Pérez (2017).

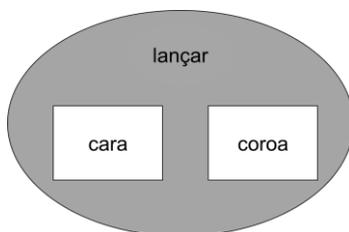
3.4 GRAFOS DE EQUILÍBRIO

Os Grafos de Equilíbrio são uma adaptação dos Grafos Existenciais, porém para modelagem de teorias em ASP. No entanto, a implicação é um operador primitivo e não pode ser representada em termos de conjunção e disjunção (PÉREZ, PÉREZ e CABALAR, 2017). Para lidar com esse problema, na extensão dos Grafos Existenciais proposta por Peirce (1882), foi substituído o componente corte por uma nova representação diagramática que é chamada de *condicional*, essa representação possui o formato de um círculo fechado e pode conter ou não retângulos chamados de *consequentes*.

As figuras 24, 25 e 26 apresentam exemplos dos Grafos de Equilíbrio *alfa* representando condições. Na Figura 24 está representada a implicação:

$$\textit{lançar} \rightarrow \textit{cara} \vee \textit{coroa}$$

Figura 24 – Exemplo 1 com condicionais



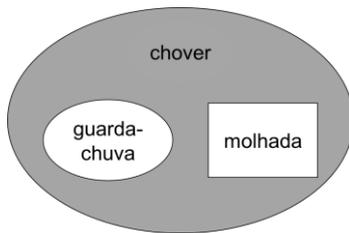
Fonte: Autor “adaptado de” Pérez, 2017

Para os casos em que não existirem retângulos será utilizada uma disjunção vazia (\perp), ou seja, uma regra *condicional* sem *consequentes* é lida da mesma maneira que nos Grafos Existenciais, como uma negação. A Figura 25 representa a implicação:

$$\textit{chover} \wedge \neg \textit{guarda-chuva} \rightarrow \textit{molhada}$$

$$\textit{chover} \wedge (\textit{guarda-chuva} \rightarrow \perp) \rightarrow \textit{molhada}$$

Figura 25 – Exemplo 2 com condicionais



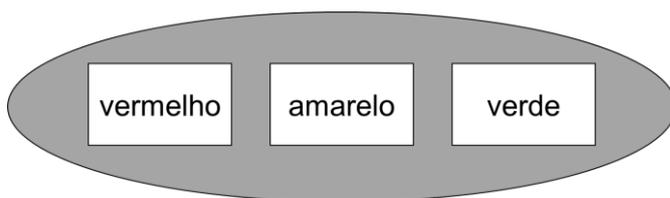
Fonte: Autor “adaptado de” Pérez, 2017

Inclusive, com a presença da nova representação diagramática *consequente*, é possível notar uma diferença com a Figura 17, onde não havia uma maneira de diferenciar entre uma condição negativa no antecessor e uma condição positiva no *consequente* (*molhada* e *guarda-chuva* desempenhavam o mesmo papel).

A Figura 26 representa uma disjunção possível para as cores de um semáforo, exemplificando uma situação na qual um *condicional* não apresenta antecedentes:

$$\text{vermelho} \vee \text{amarelo} \vee \text{verde}$$

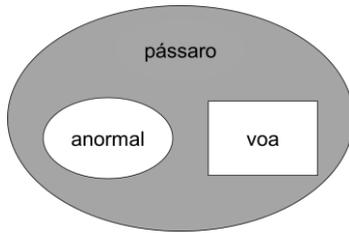
Figura 26 – Exemplo 3 com condicionais



Fonte: Autor “adaptado de” Pérez, 2017

Para finalizar os Grafos de Equilíbrio *alfa*, os exemplos das figuras Figura 27 e Figura 28 são importantes porque ilustram exemplos de Raciocínio Não-Monotônico, que serão análogos às situações presentes neste trabalho. A Figura 27 representa que *pássaros* que não são *anormais* são capazes de *voar* e a Figura 28 representa os *pinguins*, que por não voarem podem ser considerados *pássaros* e *anormais*.

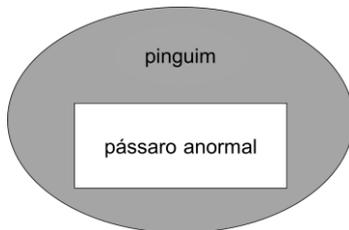
Figura 27 – Exemplo 1 de Raciocínio Não-Monotônico



$$pássaro \wedge \neg anormal \rightarrow voa$$

Fonte: Autor “adaptado de” Pérez, 2017

Figura 28 – Exemplo 2 de Raciocínio Não-Monotônico



$$pinguim \rightarrow anormal \wedge pássaro$$

Fonte: Autor “adaptado de” Pérez, 2017

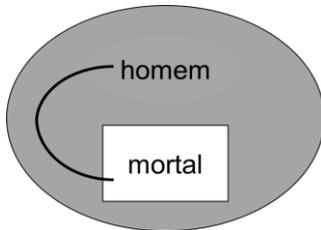
Os Grafos de Equilíbrio *beta* podem satisfazer condições para que as linhas de identidade correspondam a um quantificador universal sem a necessidade de criar novos diagramas, apenas aproveitando os *consequentes* dos Grafos de Equilíbrio alfa. Dada as condições de que a linha de identidade é envolta por um *condicional*, tem uma parte dentro de um *consequente* ou tem uma parte fora de um *consequente*, pode-se representar combinações entre *condicionais* e linhas de identidade.

As figuras 29, 30 e 31 apresentam exemplos dessas aplicações.

A Figura 29 corresponde a um quantificador universal que afirma que todo homem é mortal:

$$\forall x(homem(x) \rightarrow mortal(x))$$

Figura 29 – Exemplo 1 de condicionais com linhas de identidade



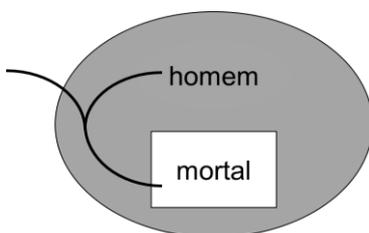
Fonte: Autor “adaptado de” Pérez, 2017

Na Figura 30 temos uma linha de identidade que não é envolta pelo *condicional*, ou seja, está aplicada na condição de que contém uma parte fora, enquanto a Figura 31 está envolta por um *condicional*, porém não contém partes dentro do *consequente*. De toda forma, um caso como o da Figura 30 pode ser lido como um quantificador existencial, tendo em vista que há equivalência entre as relações abaixo:

$$\forall x(\text{homem}(x) \rightarrow \text{mortal})$$

$$\exists x (\text{homem}(x) \rightarrow \text{mortal}(x))$$

Figura 30 – Exemplo 2 de condicionais com linhas de identidade



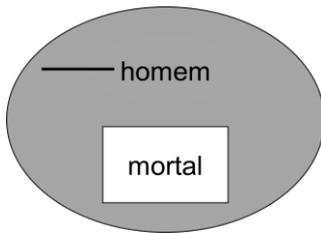
Fonte: Autor “adaptado de” Pérez, 2017

A Figura 31 também representa um caso que pode ser lido como um quantificador existencial, dada a equivalência entre as relações abaixo:

$$\forall x(\text{homem}(x) \rightarrow \text{mortal})$$

$$(\exists x (\text{homem}(x)) \rightarrow \text{mortal})$$

Figura 31 – Exemplo 3 de condicionais com linhas de identidade



Fonte: Autor “adaptado de” Pérez, 2017

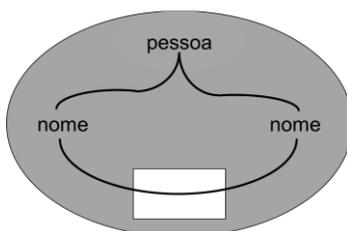
Nos Grafos de Equilíbrio as linhas de identidade também podem representar um papel implícito de igualdade. No caso onde uma linha de identidade atravessa um retângulo, a leitura deve ser feita como:

$$G \rightarrow (x = y) \wedge F$$

No caso acima G é a conjunção do átomo *antecedente* e F a disjunção do átomo *consequente*, como exemplificado pela Figura 32, onde temos declarado que uma pessoa pode ter somente um único nome, ou:

$$\forall x \forall y (\exists z \text{ pessoa}(z) \wedge \text{nome}(z,x) \wedge \text{nome}(z,y) \rightarrow (x = y))$$

Figura 32 – Exemplo de igualdade com linhas de identidade



Fonte: Autor “adaptado de” Pérez, 2017

Os Grafos Existenciais de Peirce provam que é possível a utilização de diagramas como representação de conhecimento. O raciocínio desenvolvido diagramaticamente por Peirce serve, neste trabalho, como inspiração para a investigação de uma representação de conhecimento, também com a utilização de diagramas, que fundamenta o método de indução matemática (seção 5.1).

Em trabalhos futuros, a pesquisa desenvolvida neste trabalho, conjuntamente com a utilização de diagramas para descrições lógicas, servirá como base para o desenvolvimento de um solucionador de problemas espaciais inteiramente composto por diagramas. No entanto, neste trabalho os diagramas serão utilizados como ferramenta fundamental para a prova de correção por indução matemática (seção 5). Para a construção dos diagramas foi desenvolvido um software, apresentado na seção 4.

4 SOFTWARE DESENVOLVIDO PARA CONSTRUIR DIAGRAMAS

Para a construção dos diagramas, que são fundamentais para a elaboração da prova de correção pelo método da indução matemática, foi desenvolvido um software utilizando a linguagem de programação Python. O código fonte desenvolvido neste trabalho para a construção dos diagramas está disponível em: https://github.com/AlexSaturno/MSC/blob/master/Diagrammatic_Correction_Proof_o ASPMDP.xz

No entanto, antes da explicação sobre o software é necessário entender a saída fornecida pelo algoritmo oASP(MDP), que como citado na seção 3.2.4, sabe-se que o algoritmo representa os estados por meio de listas. Nesta seção essas listas serão detalhadas e mais exemplos podem ser vistos na seção 5.2.

O algoritmo oASP(MDP) encontra listas que são formadas pela união das listas responsáveis por descrever a disposição dos elementos que são atravessados pela corda ($chain(Str)$) e pelo poste ($chain(Post)$) (seção 3.1). Essa união é representada por uma descrição que contém as faces dos elementos, que podem ser n (negativa) ou p (positiva), e pelos elementos pertencentes ao quebra-cabeças, que são numerados de 0 a 6:

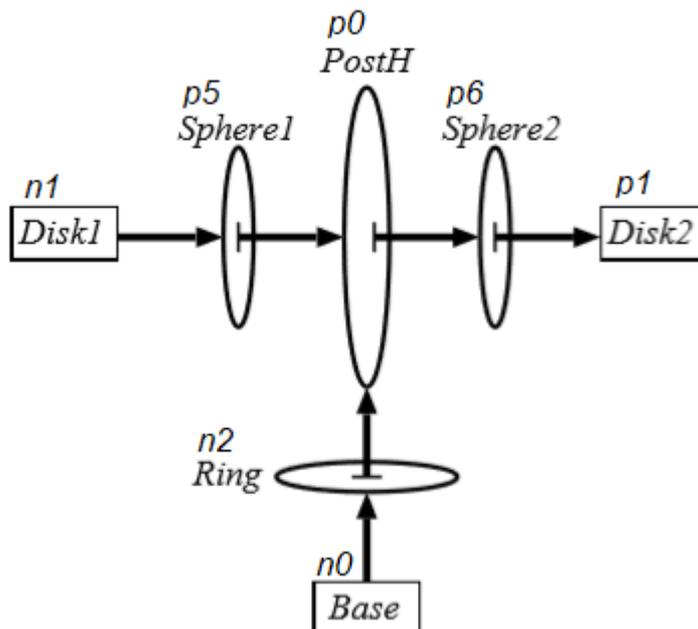
0. *PostH*
1. *String*
2. *Ring*
3. *Disk1*
4. *Disk2*
5. *Sphere1*
6. *Sphere2*

A lista da corda ($chain(Str)$) é separada da lista do poste ($chain(Post)$) pela letra b , que só tem a função de diferenciação das duas listas. O Estado Inicial (Figura 8), por exemplo, é descrito pela lista: $n1p5p0p6p1bn0n2p0$. A leitura dessa lista é feita da esquerda para a direita, enumerando os elementos que são atravessados pela corda ou pelo poste. O exemplo da lista do Estado Inicial é lido como: $n1$ (face negativa do elemento *String*, isto é, extremidade esquerda da corda); $p5$ (corda atravessando a face positiva da *Sphere1*); $p0$ (corda atravessando a face positiva do *PostH*); $p6$ (corda

atravessando a face positiva da *Sphere2*); *p1* (face positiva da *String*, isto é, extremidade direita da corda); *b* (fim da lista da corda (*chain(Str)*) e início da lista do poste (*chain(Post)*); *n0* (base do Poste); *n2* (poste atravessando a face negativa do *Ring*); *p0* (ponta do Poste).

A Figura 33 exemplifica a leitura de uma lista baseando-se na Figura 7. Seguindo as setas horizontais é possível visualizar a lista da corda. Devido à característica do domínio, os elementos *Disk1* e *Disk2* sempre estarão fixos nas extremidades da corda, portanto, nas descrições da lista feita pelo oASP(MDP), a extremidade negativa da corda (*n1*) representa o *Disk1*, enquanto a extremidade positiva da corda (*p1*) representa o *Disk2*, sendo assim, não haverá, nas listas, a utilização dos elementos 3 e 4. A lista do poste é verificada seguindo as setas verticais, onde *n0* seria a base do poste e *p0* a ponta do poste.

Figura 33 – Leitura da lista com base na Figura 7



Fonte: Autor "adaptado de" Cabalar e Santos, 2011

No total são 12 as ações possíveis de serem realizadas (tal qual definido no algoritmo oASP(MDP)):

0. Passar *Post* pelo *Ring*, atravessando a face +;
1. Passar *Post* pelo *Ring*, atravessando a face -;

2. Passar *Disk1* pelo *Post*, atravessando a face +;
3. Passar *Disk1* pelo *Post*, atravessando a face -;
4. Passar *Disk2* pelo *Post*, atravessando a face +;
5. Passar *Disk2* pelo *Post*, atravessando a face -;
6. Passar *Sphere1* pelo *Ring*, atravessando a face +;
7. Passar *Sphere1* pelo *Ring*, atravessando a face -;
8. Passar *Sphere2* pelo *Ring*, atravessando a face +;
9. Passar *Sphere2* pelo *Ring*, atravessando a face -;
10. Passar *Ring* pelo *Post*, atravessando a face +;
11. Passar *Ring* pelo *Post*, atravessando a face -;

As ações acima são responsáveis pela alteração de estados no quebra-cabeças, no entanto, utilizando os conceitos de raciocínio sobre diagramas para a simplificação de problemas (SLOMAN, 1998), para a aplicação do método de indução matemática foi considerada uma versão simplificada do quebra-cabeças de Fisherman's Folly. Por ser um quebra-cabeças simétrico, os elementos *Disk1* e *Disk2*, idem *Sphere1* e *Sphere2*, sofrem ações que têm por consequência resultados análogos, ou seja, as ações 2 e 3 são correspondentes às ações 4 e 5, da mesma forma que as ações 6 e 7 quando comparadas com as ações 8 e 9. Assim sendo, para evitar a repetição de diversos estados simétricos, foram consideradas apenas as ações 0, 1, 2, 3, 6, 7, 10 e 11. Com base nisso, para a representação diagramática, foi realizada uma substituição dos elementos *Disk2* e *Sphere2* por um elemento figurativo chamado de *Wall*.

Conforme descrito na seção 3.2.4, durante a busca, o algoritmo oASP(MDP) salva o conjunto de estados e ações que encontrou em sua exploração. No entanto, o algoritmo encontra a solução do quebra-cabeças antes de explorar todos os estados existentes e impõe limites em ações que gerariam muitos nós desnecessários na corda, como por exemplo realizar a mesma ação várias vezes seguidas, o que geraria um aumento significativo da quantidade de elementos na lista e à um distanciamento da solução do problema. Em casos como esse, o algoritmo se mantém no estado atual para que ações mais convenientes para a solução possam ser tomadas. Essas características do algoritmo fazem com que nem todos os estados recebam todas as ações, mesmo as que são possíveis de serem aplicadas. Isso será visto em alguns

estados da prova (seção 5), entretanto não significa que determinada ação não gere um estado fisicamente possível de ser construído, significa apenas que, em determinado estado, seria uma ação que o algoritmo de busca considera que o resultado encontrado se distanciaria da solução do problema.

Dada a explicação sobre as listas fornecidas pelo oASP(MDP), é possível explicar o software desenvolvido, que, em suma, recebe um arquivo contendo os estados previamente mapeados após a execução do algoritmo oASP(MDP) e em seguida, partindo da interpretação da representação dos estados por meio de listas, os diagramas são desenhados. Não há uma interface gráfica que permita interação entre o usuário e o software. A função do software desenvolvido é realizar a tradução das listas descritas em diagramas.

Sabendo que o algoritmo oASP(MDP) automaticamente impõe as restrições da quantidade de ações aplicadas, que poderiam levar a estados em que deveríamos considerar uma possível restrição imposta pelo tamanho da corda, é possível desenhar diagramas com as posições dos objetos pré-definidas. Com isso não há necessidade de limitar o tamanho da corda diagramaticamente, o que facilita a verificação do método da indução matemática (seção 5.2) nos estados mais complexos.

A interpretação das listas é realizada em duas partes. Inicialmente é feita uma análise na lista do poste ($chain(Post)$) para verificar se o objeto *Ring* está contido ou não no conjunto do poste, essa verificação é importante porque define a posição em que o objeto *Ring* deve ser desenhado. Após esta etapa, é feita uma varredura da lista da corda ($chain(Str)$) de forma similar à leitura, porém o software desenhará automaticamente a corda através dos objetos listados.

Uma vez que a lista mapeada pelo oASP(MDP) correspondente ao Estado Inicial seja: $n1p5p0p6p1bn0p2p0$, tem-se a Figura 34 que corresponde ao desenho realizado pelo software da representação diagramática do Estado Inicial. No entanto, cabe lembrar que com a simplificação realizada do Fisherman's Folly para o desenvolvimento da prova faz com que o software desenvolvido desenhe o elemento figurativo *Wall* quando se deparar com o que seria a *Sphere2* ($p6$) e *Disk2* ($p1$). Na Figura 35 é possível ver a representação do mesmo estado no quebra-cabeças real.

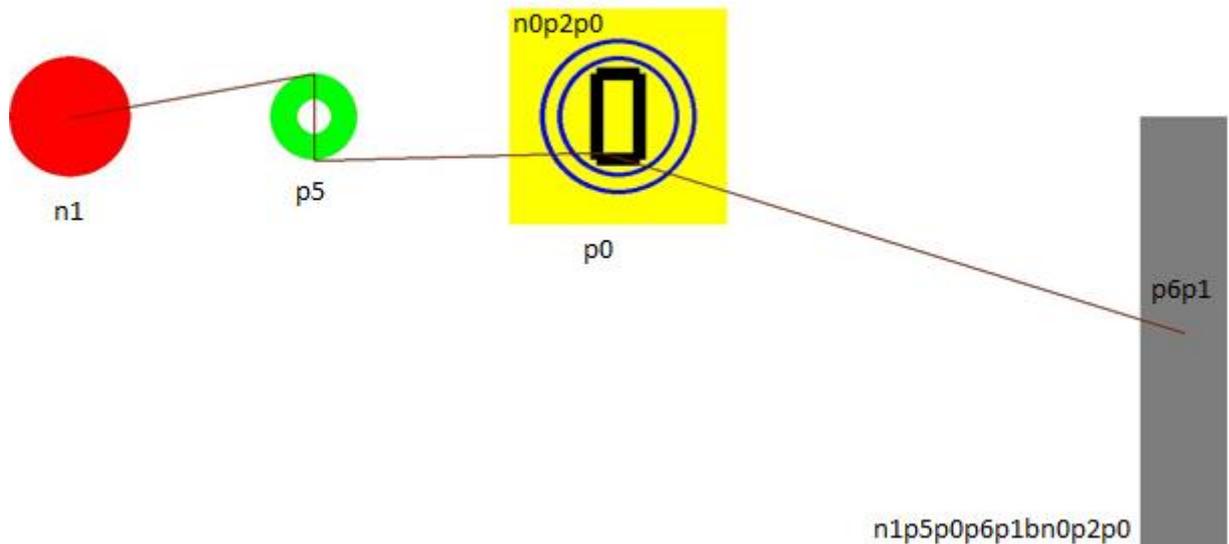
Os elementos identificados na Figura 35 são indicados na Tabela 1:

Tabela 1 – Representação dos elementos dos diagramas

<i>Disk1</i>	Forma circular fechada em vermelho
<i>Sphere1</i>	Forma circular vazada em verde
<i>Post</i>	Forma retangular vazada em preto
<i>Ring</i>	Formas circulares vazadas em azul
<i>Post Base</i>	Forma retangular fechada em amarelo
<i>Wall</i>	Forma retangular fechada em cinza
<i>String</i>	Linha marrom

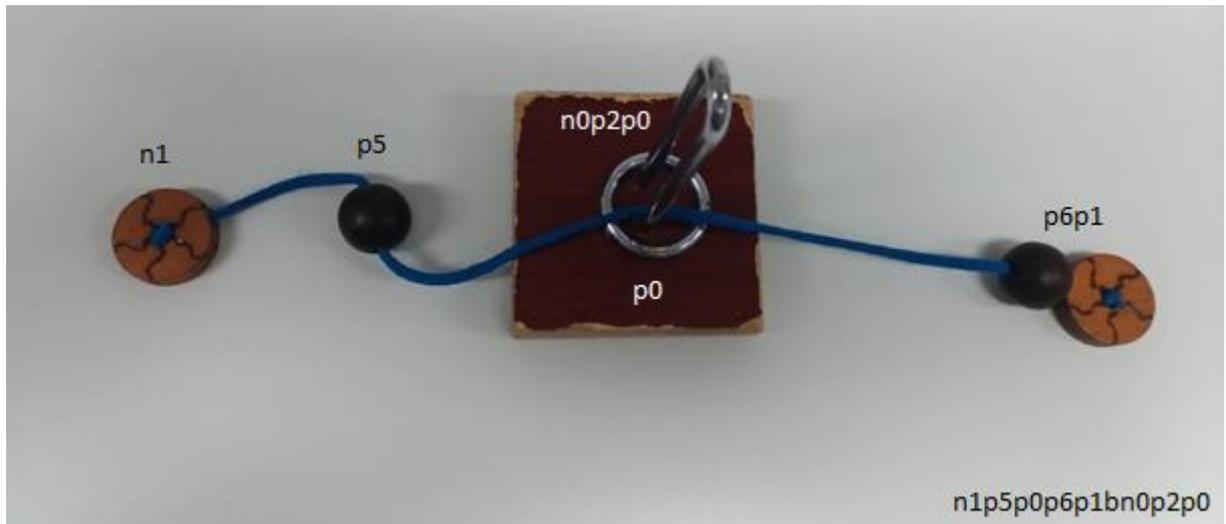
Fonte: Autor

Figura 34 – Representação Diagramática do Estado Inicial desenhada pelo Software desenvolvido



Fonte: Autor

Figura 35 – Estado Inicial no quebra-cabeças de Fisherman’s Folly real



Fonte: Autor

Como visto na seção 3.1, os objetos com Buracos podem ser atravessados por outros objetos, no entanto, é importante saber que, dada as características do quebra-cabeça, apenas os elementos *Post* e *Ring* possuem a chance de serem atravessados pelas faces positiva e negativa. Nos diagramas, a corda atravessando face positiva do objeto *Post* é representada à esquerda da forma retangular vazada em preto (Tabela 1) e a face negativa é representada à direita. Também nos diagramas, porém quando se trata do objeto *Ring*, as faces positiva e negativa são representadas, respectivamente, à direita e esquerda das formas circulares vazadas em azul (Tabela 1).

Saber diferenciar as faces que são atravessadas pela corda é fundamental para poder realizar a análise correta dos diagramas e, com isso, entender a prova de correção do algoritmo oASP(MDP) pelo método da indução matemática com a utilização de diagramas, apresentado na seção 5.

5 PROVA DE CORREÇÃO DO oASP(MDP) POR INDUÇÃO MATEMÁTICA

Baseando-se nos fundamentos dos Grafos Existenciais de Peirce (1882) e dos exemplos de provas de Nelsen (1993, 2015) e Brown (2008), nesta seção está descrita a prova de correção por indução matemática que foi desenvolvida a partir de uma representação diagramática dos estados gerados pelo algoritmo oASP(MDP) (seção 3.2.4), que foram desenhados pelo software desenvolvido neste trabalho (seção 4).

5.1 MÉTODO DE INDUÇÃO MATEMÁTICA

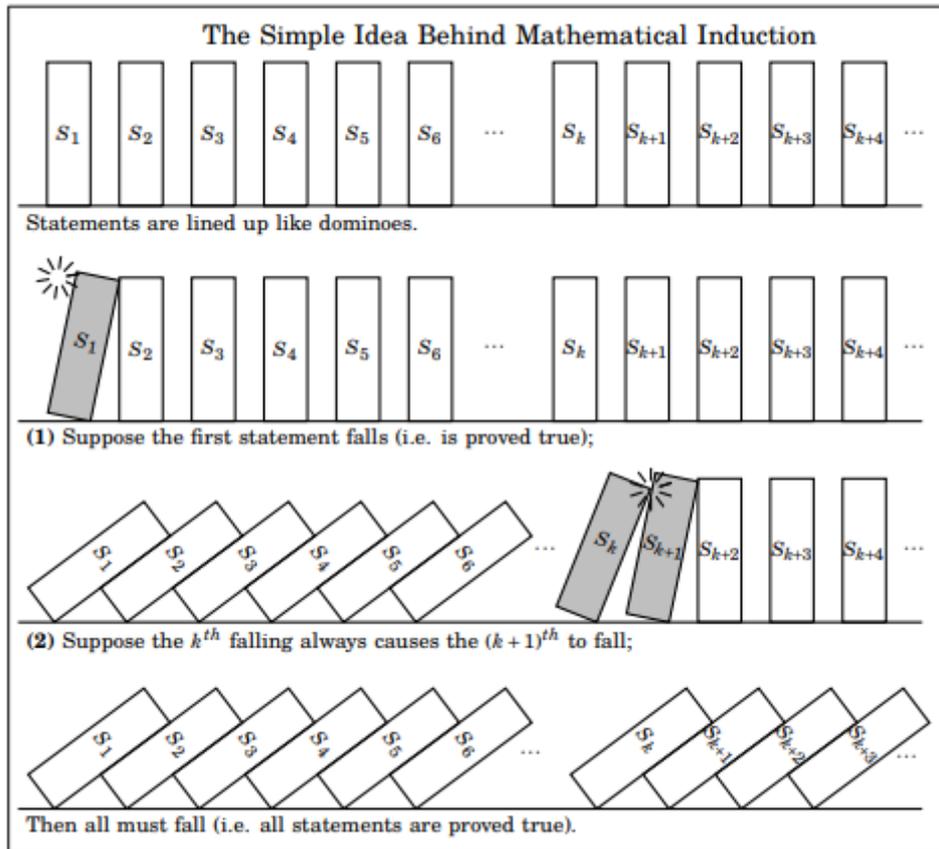
O método de indução matemática consiste na prova de que todos os elementos de um conjunto se comportam da mesma maneira, desde que uma mesma propriedade P seja considerada no conjunto.

Na Figura 36 está representada a ideia do método de indução matemática com o exemplo do Efeito Dominó, onde são enfileiradas peças do jogo Dominó a uma distância na qual, se uma peça for derrubada, ela atingirá a outra e, conseqüentemente, todas as peças enfileiradas devem cair.

A queda de uma peça tem por consequência a queda das peças subsequentes, essa dedução pode ser feita analisando apenas o resultado de derrubar a primeira peça e verificando se o mesmo resultado mantêm-se ao derrubar qualquer outra peça. Com base nesse raciocínio, dada uma propriedade P , para a confirmação desse método é necessária a prova de dois passos:

1. Caso Base: a propriedade P deve ser verdadeira para o valor inicial, isto é, nesse passo é provado $P(0)$;
2. Passo Indutivo: assumindo que a propriedade P é verdadeira para um valor n pertencente à um determinado conjunto (hipótese indutiva), a propriedade P deve manter-se verdadeira para um valor $n+1$ pertencente ao mesmo conjunto. Em termos matemáticos, provamos que $\forall n \in \mathbb{N} P(n) \rightarrow P(n+1)$.

Figura 36 – Simples ideia por trás da Indução Matemática (Efeito Dominó)



Fonte: Hammack, 2013

Um exemplo do método da indução matemática (VELLEMAN, 1994) é a prova de que, dado o conjunto dos números naturais \mathbb{N} , $2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$.

Seguindo a estratégia da prova do Caso Base e do Passo Indutivo, o objetivo é a prova da declaração $\forall n \in \mathbb{N} P(n)$ onde, nesse caso, $P(n)$ é equivalente à hipótese indutiva que será provada: $2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$.

No Caso Base é definido o valor inicial $n = 0$. Realizando a substituição do n por 0, tem-se:

$$2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$$

$$2^0 = 2^{0+1} - 1$$

$$2^0 = 2^1 - 1$$

$$2^0 = 1 = 2^1 - 1$$

Conclui-se que $P(0)$ equivale à $2^0 = 1 = 2^1 - 1$ e, como a propriedade se mostra verdadeira para o valor inicial determinado, o Caso Base está provado.

O Passo Indutivo requer a prova de que $\forall n \in \mathbb{N} P(n) \rightarrow P(n+1)$. Assume-se um valor arbitrário n que corresponde ao conjunto dos números naturais \mathbb{N} e, em seguida, assumindo que a propriedade $P(n)$ é verdadeira, é possível provar que $P(n+1)$ também é verdadeira. Tendo como $P(n)$ a equação $2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$, é possível obter que $P(n+1)$ corresponde à $2^0 + 2^1 + \dots + 2^{n+1} = 2^{n+2} - 1$. A solução para essa prova se dá pelo desenvolvimento matemático a seguir:

$$\begin{aligned} 2^0 + 2^1 + \dots + 2^{n+1} &= (2^0 + 2^1 + \dots + 2^n) + 2^{n+1} \\ &= (2^{n+1} - 1) + 2^{n+1} \\ &= 2 \cdot 2^{n+1} - 1 \\ &= 2^{n+2} - 1 \end{aligned}$$

Portanto, $2^0 + 2^1 + \dots + 2^{n+1} = 2^{n+2} - 1$.

Com isso, temos a prova de que o comportamento matemático, para essa propriedade, será o mesmo dado qualquer número natural. Esse raciocínio da prova será aplicado para a solução do quebra-cabeças de Fisherman's Folly dada pelo algoritmo oASP(MDP), porém com a utilização de diagramas para a realização da indução.

5.2 MÉTODO DE INDUÇÃO MATEMÁTICA PARA O oASP(MDP)

O método de indução matemática foi aplicado para provar que todos os estados encontrados pelo algoritmo oASP(MDP), na busca por uma solução do quebra-cabeças de Fisherman's Folly, são fisicamente possíveis de serem construídos. As propriedades P são características do domínio apresentado na seção 3.1, portanto, as diferenças entre trabalhar com números ou com elementos de um quebra-cabeças espacial são encontradas apenas no comportamento do domínio, ou seja, ao invés da utilização das regras matemáticas, são utilizadas as restrições do domínio do quebra-cabeças (seção 3.1).

Baseando-se no fato de que o quebra-cabeças de Fisherman's Folly não é um conjunto numérico, porém um conjunto de objetos, foi realizada uma analogia entre os estados do quebra-cabeças e os números, além das ações e operações matemáticas. Para essa analogia, foi considerada uma corda infinita, ou seja, os resultados possíveis obtidos não dependem das restrições físicas impostas por uma possível quantidade de excesso de nós consequentes de repetidas ações. Sem essa consideração, haveria a possibilidade de uma ação não gerar o resultado esperado por causa da limitação da corda.

Os estados do quebra-cabeças são representados por meio de diagramas. Cada diagrama representa uma configuração dos elementos do domínio e todos são desenhados automaticamente a partir das listas encontradas pelo algoritmo oASP(MDP) (como apresentado na seção 4).

O método de indução matemática é aplicado para que seja provado que os estados que são encontrados pelo algoritmo oASP(MDP), em sua busca, são fisicamente possíveis de serem construídos, como formalizado pelo Teorema I, descrito abaixo:

Teorema I: *Seja A o algoritmo oASP(MDP) e F o quebra-cabeças de Fisherman's Folly, toda solução π gerada por A para solucionar F é fisicamente factível, isto é, as ações aplicadas pelo algoritmo oASP(MDP) em estados fisicamente possíveis de serem construídos no quebra-cabeças de Fisherman's Folly gera estados que, por sua vez, também são fisicamente possíveis de serem construídos.*

São considerados estados fisicamente possíveis de serem construídos aqueles onde não existam irregularidades quanto às restrições do domínio, por exemplo, o objeto *Sphere1* não pode atravessar o buraco do *PostH*. Se alguma ação gerasse tal disposição dos objetos, seria alcançado pelo oASP(MDP) um estado fisicamente impossível de ser construído.

A Tabela 2 foi construída após a realização de testes empíricos no quebra-cabeças de Fisherman's Folly e apresenta os possíveis resultados que as ações geram no quebra-cabeças. Partindo dos testes empíricos, foi verificado que cada ação realizada altera a quantidade de cruzamentos entre determinados elementos

pertencentes à *chain(Str)* (seção 3.1). As ações 0 e 1 alteram o posicionamento do elemento *Ring* que pode pertencer à *chain(Post)* (seção 3.1) ou não, a *chain(Post)* só determina, diagramaticamente, se o elemento *Ring* é desenhado dentro ou fora do conjunto formado por *Base* e *Post*.

Quando comparados dois estados, um de origem (qualquer estado fisicamente possível de construir) e outro um estado alcançado como consequência de uma ação, caso houvesse alguma ação que gerasse algum cruzamento diferente dos evidenciados na Tabela 2, estaria sendo gerado um estado que não fosse fisicamente possível de ser construído, o que pode ser visto nos diagramas quando comparada a quantidade de cruzamentos do estado atual com a do estado gerado pela ação.

Na Tabela 2 também é possível perceber a presença do símbolo “=”, esse símbolo representa que a aplicação das ações pode ter mantido a busca no mesmo estado, isso ocorre por causa das restrições impostas ao oASP(MDP) de não explorar estados onde haja a repetição de muitas ações, o que, como dito anteriormente, geraria muitos nós e um distanciamento da solução. Outro símbolo que aparece na parte dos resultados é o “0”, nesse caso houve alteração na configuração da lista, no entanto, a quantidade de cruzamentos se manteve a mesma. Onde está descrito como resultado “Ring position” a alteração ocorreu apenas na lista do poste, conseqüentemente, na posição do elemento *Ring*. Os demais resultados são descritos como a quantidade de cruzamentos nos buracos de Ring ou PostH (faces positivas e negativas).

Tabela 2 – Ações e Resultados Possíveis

	AÇÕES	RESULTADOS POSSÍVEIS				
<i>Post</i> pelo <i>Ring</i>	0	Ring position	+1 [+Ring, -Ring]	+2 [+Ring, -Ring]	=	
	1	Ring position	+1 [+Ring, -Ring]	+2 [+Ring, -Ring]	=	
<i>Disk1</i> pelo <i>Post</i>	2	+1 [-PostH]	-1 [+PostH]	=		
	3	-1 [-PostH]	+1 [+PostH]	=		
<i>Sphere1</i> pelo <i>Ring</i>	6	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	0	=	
	7	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	0	=	
<i>Ring</i> pelo <i>Post</i>	10	+1 [+PostH, -PostH]	-1 [+PostH, -PostH]	+2 [+PostH, -PostH]	0	=
	11	+1 [+PostH, -PostH]	-1 [+PostH, -PostH]	+2 [+PostH, -PostH]	0	=

Fonte: Autor

Lema I: Os resultados das ações representados na Tabela 2 são únicos.

Prova do Lema I: Por *reductio ad absurdum*, assumindo que não sejam únicos, i.e., que existam distintos efeitos partindo das ações, atinge-se estados que não são resultados das ações.

Lema II: Os resultados das ações representados na Tabela 2 são fisicamente possíveis de serem construídos.

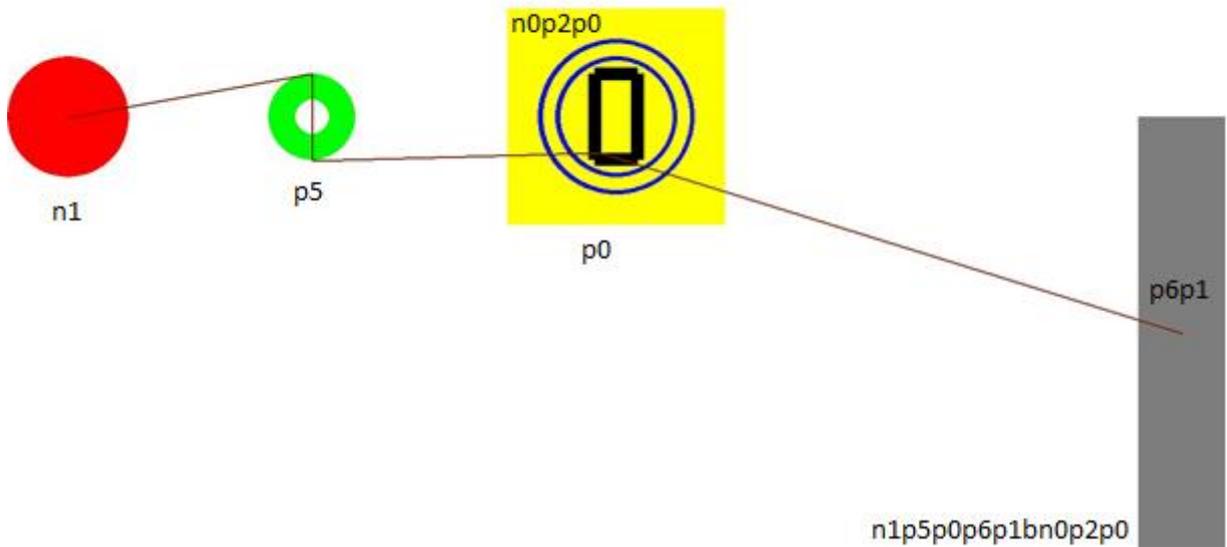
Prova do Lema II: Prova-se esse Lema por verificação empírica no quebra-cabeças de Fisherman's Folly real.

Prova do Teorema I: Utilizando a Indução Matemática, seja $P(n)$ a hipótese indutiva de que os estados são fisicamente possíveis de serem construídos, a aplicação de ações gera estados que também são fisicamente possíveis de serem construídos no quebra-cabeças de Fisherman's Folly. A prova procederá em dois passos: Primeiro o Caso Base e, em seguida, o Passo Indutivo.

O Caso Base consiste na aplicação das ações no Estado Inicial do quebra-cabeças (Figura 8). Na Figura 37 é possível ver a configuração inicial do quebra-cabeças de Fisherman's Folly por meio de uma vista superior. Essa configuração corresponde à representação diagramática do Estado Inicial que foi construída pelo software desenvolvido (conforme descrito na Seção 4), nesse software é realizado o desenho do diagrama, dos estados mapeados pelo oASP(MDP) em sua busca, a partir da lista dos elementos do quebra-cabeças.

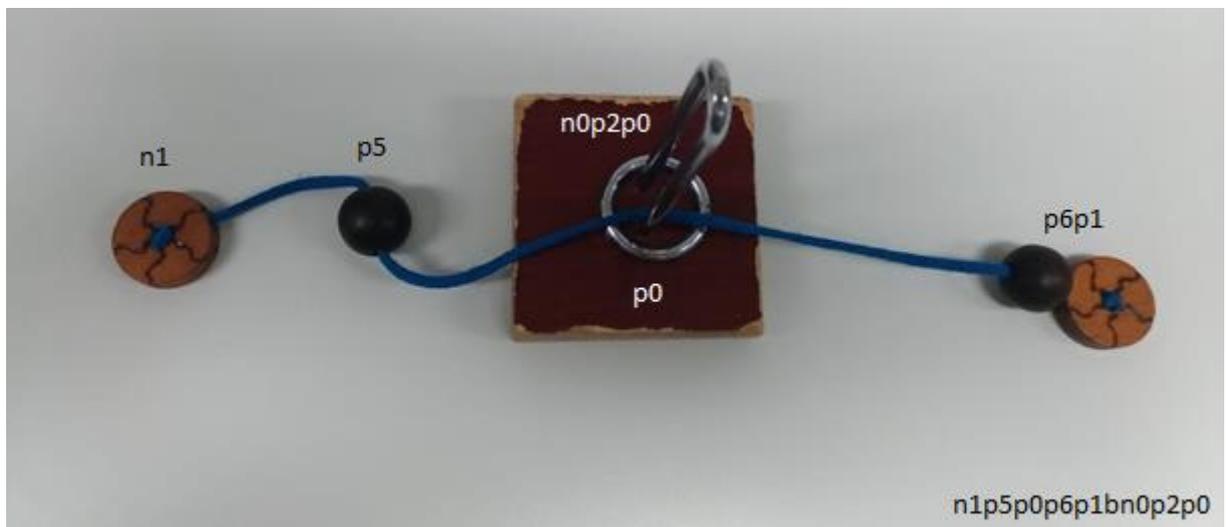
Visto que a lista mapeada pelo oASP(MDP) que corresponde ao Estado Inicial é: $n1p5p0p6p1bn0p2p0$, e dadas as simplificações realizadas para o desenvolvimento da prova (vide seção 4), a Figura 37 corresponde ao desenho realizado automaticamente pelo software da representação diagramática do Estado Inicial. A Figura 38 representa o mesmo estado, porém no quebra-cabeças real.

Figura 37 – Representação Diagramática do Estado Inicial



Fonte: Autor

Figura 38 – Estado Inicial no quebra-cabeças de Fisherman's Folly



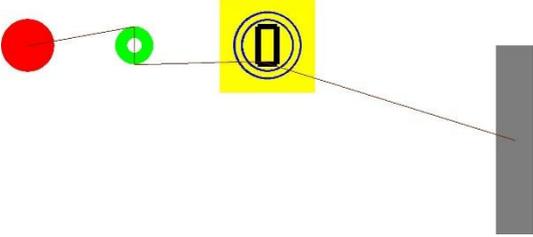
Fonte: Autor

Caso Base:

A prova do Caso Base consiste na aplicação das ações básicas sobre o estado inicial quebra-cabeças, verificando se é possível chegar em novos estados que, por sua vez, também são possíveis de serem construídos. Analogamente ao método de

indução matemática descrito na seção 5.1, a indução matemática diagramática está exemplificada na Tabela 3:

Tabela 3 – Analogia entre Indução Matemática Numérica e Diagramática

Indução Matemática Numérica	Indução Matemática Diagramática
Provar: $2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$	Provar: Os estados encontrados pelo oASP(MDP) são fisicamente possíveis de serem construídos.
P(0): $2^0 = 2^1 - 1$	P(0): 
$2^0 = 1 = 2^1 - 1$	Aplicam-se ações possíveis

Fonte: Autor

Aplicando as ações possíveis no Estado Inicial são encontrados novos estados que, de acordo com a Tabela 2 (detalhada anteriormente e respeitando os Lemas I e II), são fisicamente possíveis de serem construídos no quebra-cabeças de Fisherman's Folly. O algoritmo oASP(MDP), em sua busca por uma solução, aplica as ações 1, 2 e 3, e tem por resultado, respectivamente, a Figura 39, a Figura 41 e a Figura 43:

A lista encontrada pelo oASP(MDP) para a Figura 39 é:
 $n1p5n2p0p2p6p1bn0p0$.

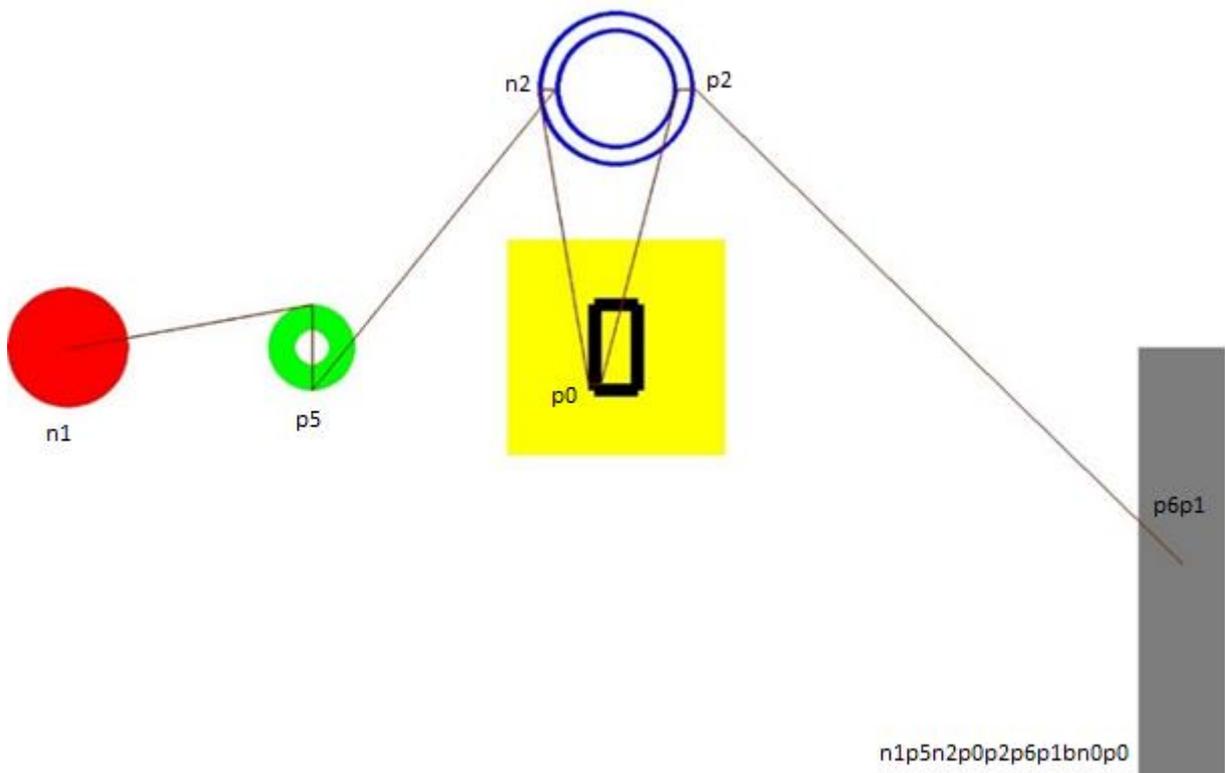
Ao analisar a Figura 39 é possível perceber que o *Ring* está fora do conjunto *Base* e *Post*, o que comprova diagramaticamente a lista do poste ($n0p0$), onde está descrito apenas a base ($n0$) e o topo do poste ($p0$), sem a presença do elemento *Ring*.

Percorrendo a *String* (linha marrom, vide Tabela 1) é possível verificar no diagrama da Figura 39 a lista da corda ($n1p5n2p0p2p6p1$). Inicialmente partindo do *Disk1* ($n1$), a corda atravessa a *Sphere1* pela face positiva ($p5$), em seguida o *Ring*

pela face negativa ($n2$), o *PostH* pela face positiva ($p0$), o *Ring* novamente, agora pela face positiva ($p2$) e, por fim, o elemento figurativo *Wall* ($p6p1$).

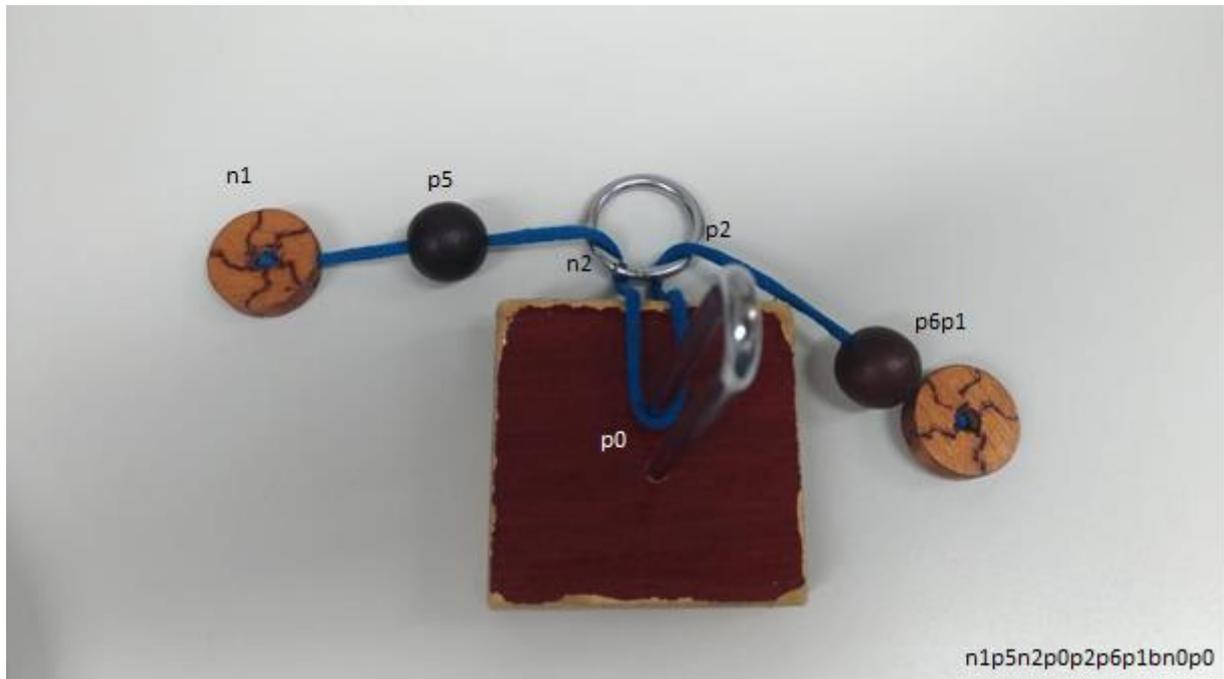
Conforme descrito na Tabela 2, a realização da Ação 1 deve alterar a posição do *Ring* e pode, ou não, aumentar a quantidade de cruzamentos no *Ring*. Percebe-se ao comparar a Figura 37 com a Figura 39 que o *Ring* não faz mais parte da chain(*Post*) (seção 3.1), por isso é desenhado afastado do conjunto *Base* e *Post*. Já na chain(*Str*) (seção 3.1) há a inclusão de dois cruzamentos $n2$ e $p2$, o que pode ser visto no diagrama ao analisar a corda atravessando o elemento *Ring* duas vezes. A Figura 40 representa o mesmo estado resultante da Ação 1 no Estado Inicial feita no quebra-cabeças real.

Figura 39 – Resultado da Ação 1 no Estado Inicial – Representação Diagramática



Fonte: Autor

Figura 40 – Resultado da Ação 1 no Estado Inicial - Fisherman's Folly



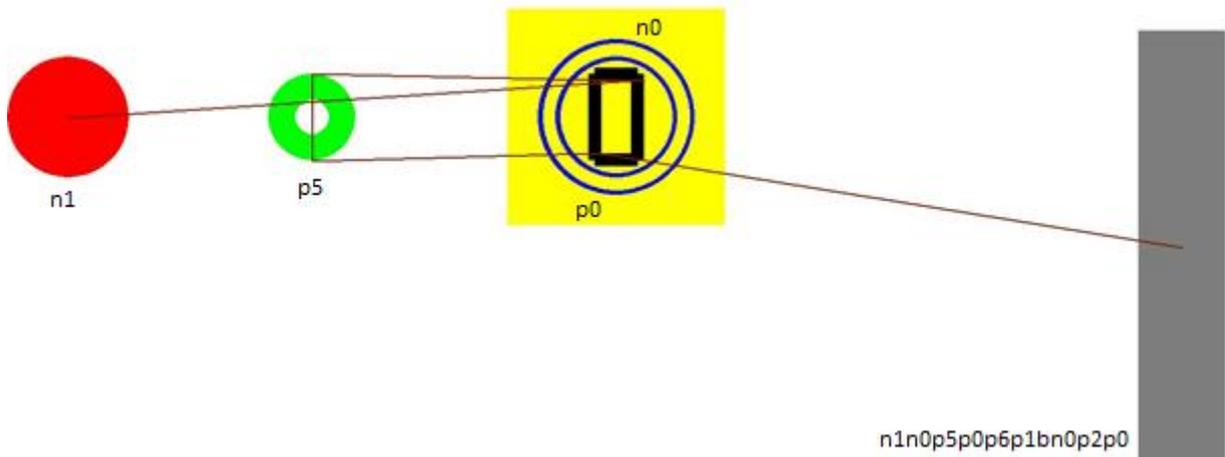
Fonte: Autor

A lista encontrada pelo oASP(MDP) para a Figura 41 é:
 $n1n0p5p0p6p1bn0p2p0$.

Ao analisar a Figura 41 é perceptível que o *Ring* faz parte do conjunto *Base* e *Post*, como descrito na lista do poste ($n0p2p0$). Quanto à lista da corda, partindo do *Disk1* fixado na extremidade negativa da *String* ($n1$) e seguindo a *String*, a corda atravessa o *PostH* pela face negativa ($n0$), a *Sphere1* pela face positiva ($p5$), o *PostH* novamente, porém pela face positiva ($p0$), e para finalizar no elemento figurativo *Wall* ($p6p1$).

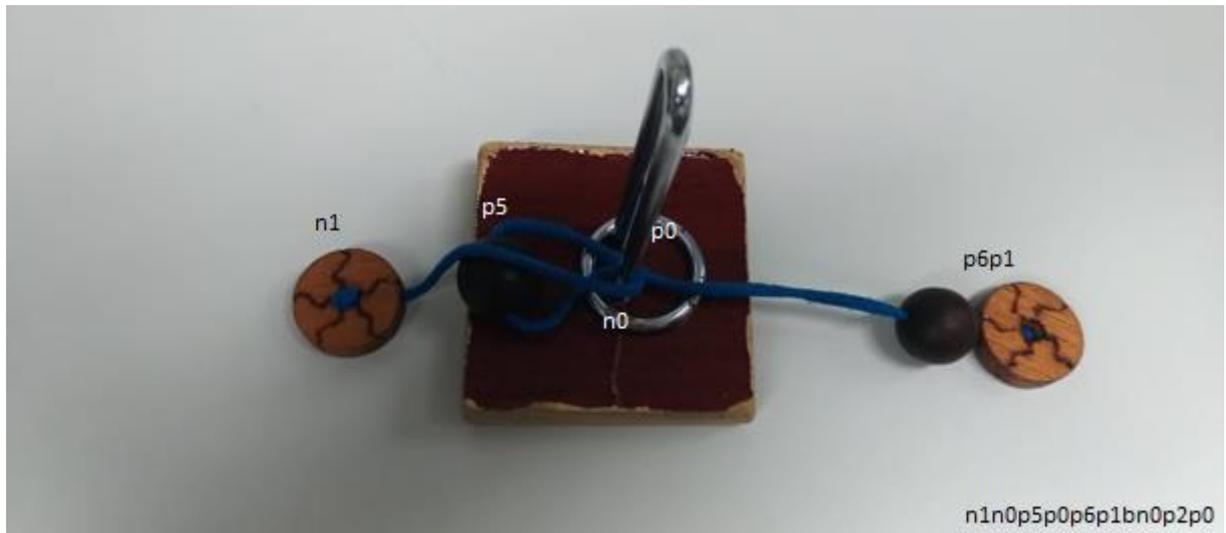
Conforme previsto pela Tabela 2, a Ação 2 aumenta a quantidade de cruzamentos em -PostH, que é única diferença entre a Figura 37 e a Figura 41. A Figura 42 representa o mesmo resultado da Ação 2 no Estado Inicial com a visualização no quebra-cabeças real.

Figura 41 – Resultado da Ação 2 no Estado Inicial – Representação Diagramática



Fonte: Autor

Figura 42 – Resultado da Ação 2 no Estado Inicial - Fisherman's Folly



Fonte: Autor

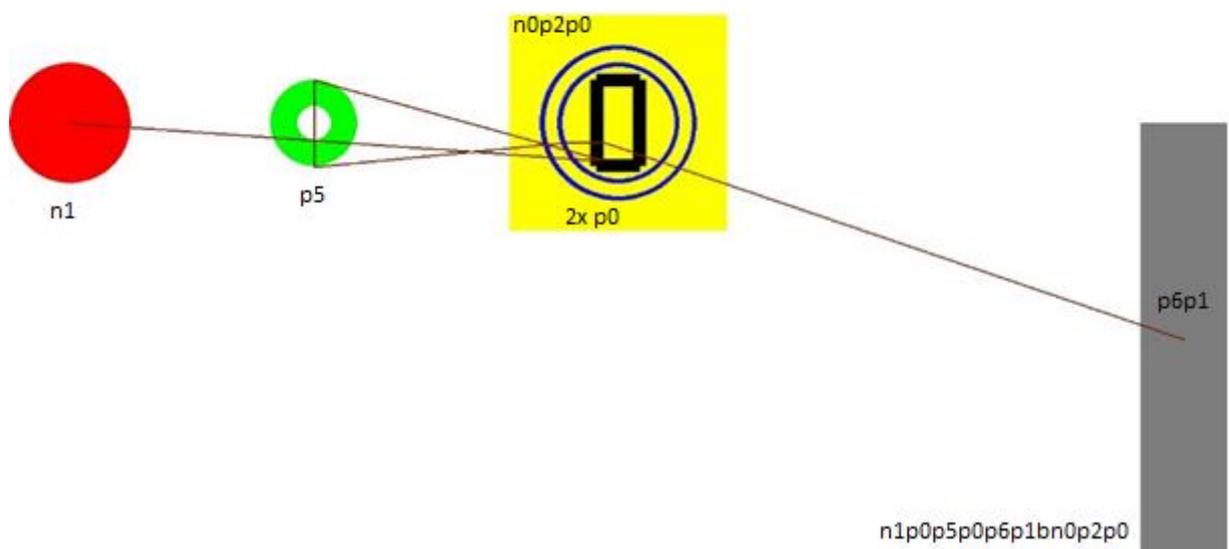
A lista encontrada pelo oASP(MDP) para a Figura 43 é:
 $n1p0p5p0p6p1bn0p2p0$.

Ao analisar a Figura 43 é possível ver que o *Ring* faz parte do conjunto *Base* e *Post*, como descrito na lista do poste ($n0p2p0$). Quanto à lista da corda, partindo do *Disk1* fixado na extremidade negativa da *String* ($n1$) e seguindo a *String*, a corda atravessa o *PostH* pela face positiva ($p0$), a *Sphere1* pela face positiva ($p5$),

novamente atravessa o *PostH* pela face positiva ($p0$), para finalizar no elemento figurativo *Wall* ($p6p1$).

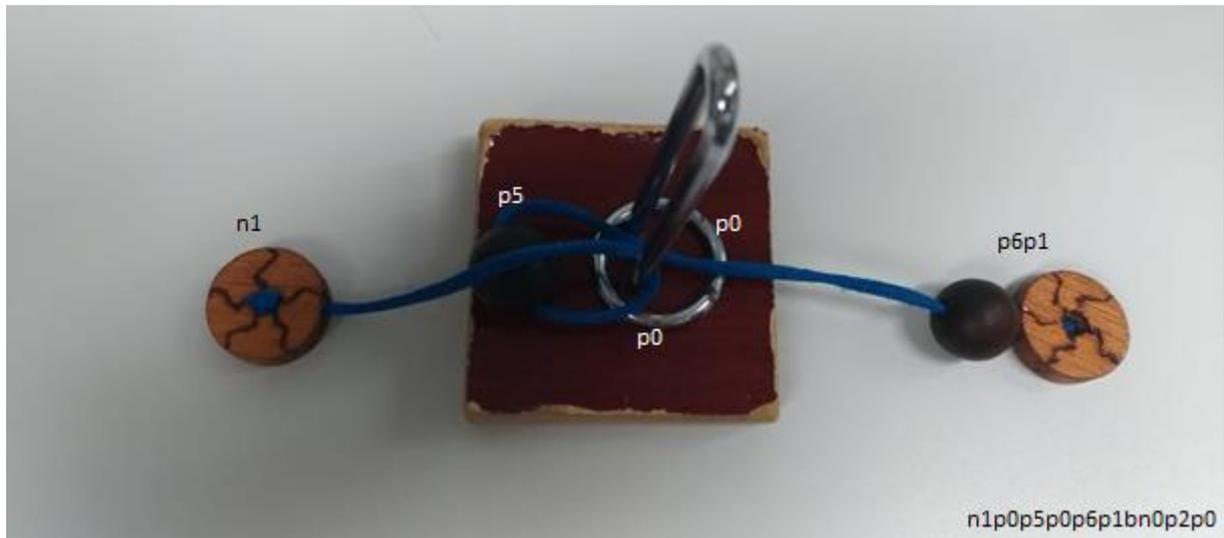
De acordo com a Tabela 2, a Ação 3 aumenta a quantidade de cruzamentos em +PostH, o que é perceptível analisando o diagrama da Figura 43 por ser a única diferença apresentada quando comparado com a Figura 37. Na Figura 44 temos a representação dessa mesma Ação 3 no quebra-cabeças real.

Figura 43 – Resultado da Ação 3 no Estado Inicial – Representação Diagramática



Fonte: Autor

Figura 44 – Resultado da Ação 3 no Estado Inicial – Fisherman's Folly



Fonte: Autor

As ações 0, 6, 7, 10 e 11 não alteram o Estado Inicial (Figura 37 e Figura 38) por serem ações que gerariam estados que não são fisicamente possíveis de serem construídos porque romperiam restrições físicas específicas do quebra-cabeças. A ação 0 não altera o estado porque não há como atravessar novamente o anel pela face positiva sendo que já está atravessado. A tentativa de atravessar o *Ring* pelo *PostH* (ações 10 e 11) é uma ação possível desde que o elemento *Ring* não esteja contido na lista do poste, já a tentativa de atravessar a *Sphere1* pelo *Ring* (ações 6 e 7) também é inválida nessa mesma condição devido a espessura do poste.

Da mesma maneira que na indução matemática numérica, a solução do Caso Base na indução matemática diagramática do Fisherman's Folly tende a ser mais simples que o Passo Indutivo, que apresenta maior complexidade de análise e solução.

Passo Indutivo:

Para o Passo Indutivo é necessário provar que, partindo de qualquer estado n do quebra-cabeças F , assumindo-o fisicamente possível de ser construído ($P(n)$), a aplicação de ações gera estados que são possíveis de serem encontrados no quebra-cabeças físico, ou seja, $\forall (n \in F) P(n) \rightarrow P(n+1)$. Portanto, foram selecionados estados

originadores, i.e., estados que englobam todas as configurações iniciais possíveis e servem como base para a originar outros estados mais complexos.

Para calcular a quantidade de *estados originadores* fisicamente possíveis de serem construídos, analisou-se empiricamente o quebra-cabeças realizando variações entre os elementos que são móveis, como por exemplo o *Disk1*, a *Sphere1* e o *Ring*, e construiu-se a Tabela 4, que contém todas as verificações de todas configurações possíveis:

Tabela 4 – Todas possibilidades de configurações dos elementos

<i>Disk1</i>	<i>Sphere1</i>	<i>Ring</i>
Fixo	Fixo	Fixo
Fixo	Fixo	Variando
Fixo	Variando	Fixo
Fixo	Variando	Variando
Variando	Fixo	Fixo
Variando	Fixo	Variando
Variando	Variando	Fixo
Variando	Variando	Variando

Fonte: Autor

Após a realização de todas as configurações básicas possíveis foram encontrados 30 *estados originadores* diferentes, que foram submetidos à comparação com os estados resultantes da aplicação das ações possíveis, como feito no Caso Base, porém para cada um dos 30 estados. Os 30 *estados originadores* descritos na forma de listas, que são geradas pelo oASP(MDP), são:

- S1: n1p5p0p6p1bn0p2p0
- S2: n1p5n2p0p2p6p1bn0p0
- S3: n1p5p0n2p0p2n0p6p1bn0p0
- S4: n1p5n0n2p0p2p0p6p1bn0p0
- S5: n1n2p5p0p2p6p1bn0p0
- S6: n1p2p5n2n2p0p2p6p1bn0p0
- S7: n1n2p5p2p0n2p0p2n0p6p1bn0p0

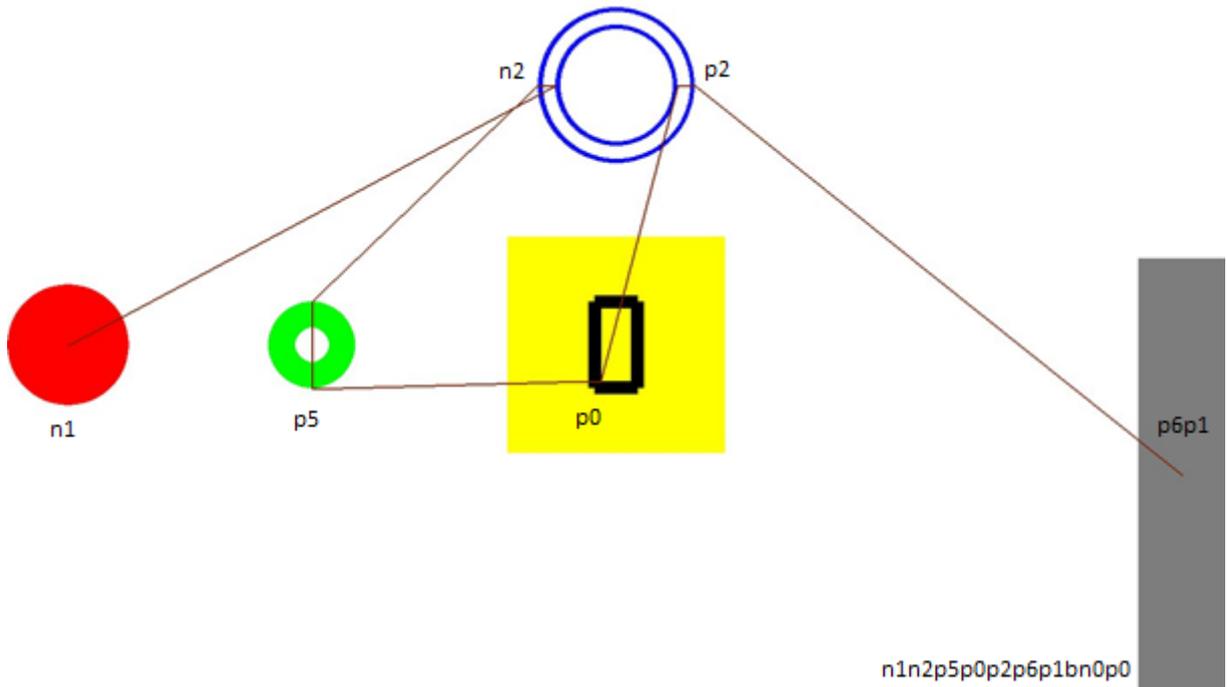
S8: n1p2p5n2p0n2p0p2n0p6p1bn0p0
 S9: n1n2p5p2n0n2p0p2p0p6p1bn0p0
 S10: n1p2p5n2n0n2p0p2p0p6p1bn0p0
 S11: n1n0p5p0p6p1bn0p2p0
 S12: n1p0p5p0p6p1bn0p2p0
 S13: n1p0p5n2p0p2p6p1bn0p0
 S14: n1n0p5n2p0p2p6p1bn0p0
 S15: n1p0p5p0n2p0p2n0p6p1bn0p0
 S16: n1n0p5p0n2p0p2n0p6p1bn0p0
 S17: n1p0p5n0n2p0p2p0p6p1bn0p0
 S18: n1n0p5n0n2p0p2p0p6p1bn0p0
 S19: n1p0n2p5p0p2p6p1bn0p0
 S20: n1n0n2p5p0p2p6p1bn0p0
 S21: n1p0p2p5n2n2p0p2p6p1bn0p0
 S22: n1n0p2p5n2n2p0p2p6p1bn0p0
 S23: n1p0n2p5p2p0n2p0p2n0p6p1bn0p0
 S24: n1n0n2p5p2p0n2p0p2n0p6p1bn0p0
 S25: n1p0p2p5n2p0n2p0p2n0p6p1bn0p0
 S26: n1n0p2p5n2p0n2p0p2n0p6p1bn0p0
 S27: n1p0n2p5p2n0n2p0p2p0p6p1bn0p0
 S28: n1n0n2p5p2n0n2p0p2p0p6p1bn0p0
 S29: n1p0p2p5n2n0n2p0p2p0p6p1bn0p0
 S30: n1n0p2p5n2n0n2p0p2p0p6p1bn0p0

Comparando a quantidade de cruzamentos que a corda realiza em cada elemento, é possível perceber que as ações realizadas alteram apenas as quantidades de cruzamentos da corda nos elementos. Portanto, um estado que não fosse fisicamente possível de ser construído, só existiria se não correspondesse aos padrões de cruzamentos característicos dos elementos que compõe o quebra-cabeças (Tabela 2).

Na Tabela 2 é possível ver que as ações 0 e 1 (Passar o *Post* pelo *Ring*) interferem diretamente apenas nos cruzamentos da corda no *Ring*, alterando apenas sua posição ou gerando cruzamentos pelas faces +Ring e -Ring. Os únicos casos em

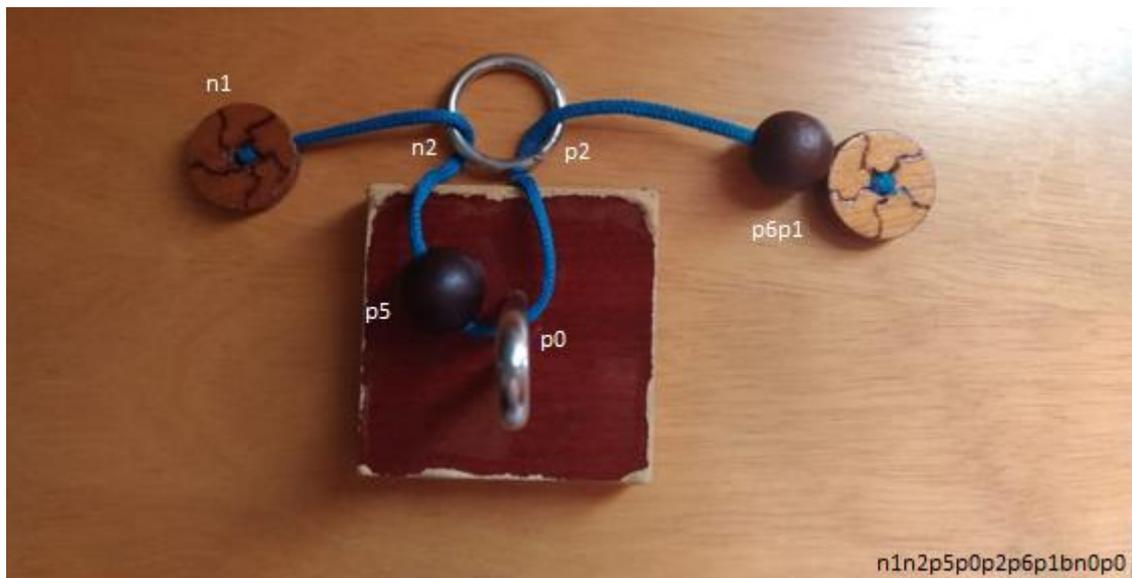
que haverá o acréscimo de dois cruzamentos pelas faces destacadas serão onde, em ações prévias, o *Ring* atravessou o *Post* (Ações 10 e 11). Um exemplo é o Estado originador 5, dado pela lista: $n1n2p5p0p2p6p1bn0p0$ e apresentado na Figura 45:

Figura 45 – Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 46 – Estado originador 5 – Fisherman's Folly



Fonte: Autor

A lista $n1n2p5p0p2p6p1bn0p0$ representa as duas listas da corda e do poste, separadas pela letra b . A lista da corda é descrita como $n1n2p5p0p2p6p1$ e é lida como $n1$ (extremidade esquerda da corda, onde está o elemento $Disk1$), $n2$ (a corda atravessa o elemento $Ring$ saindo pela face negativa), $p5$ (a corda atravessa o elemento $Sphere1$ saindo pela face positiva), $p0$ (a corda atravessa o elemento $PostH$ saindo pela face positiva), $p2$ (a corda atravessa o elemento $Ring$, dessa vez saindo pela face positiva), $p6p1$ (elemento figurativo $Wall$). A lista do poste descreve apenas a Base ($n0$) e o topo do poste ($p0$), o que indica que não há cruzamentos com o elemento $Ring$, o que pode ser percebido analisando o diagrama da Figura 45.

O diagrama do novo estado gerado é formado a partir da lista: $n1n2p5p2p0p6p1bn0p2p0$. A aplicação da ação 0 no Estado originador 5 gera apenas uma mudança no posicionamento do $Ring$, que agora passa a fazer parte do conjunto $Base$ e $Post$, e mantém-se a quantidade de vezes que a corda atravessa o $Ring$, como é possível verificar quando comparada a Figura 45 com a Figura 47. Nesse caso, apenas a $chain(Post)$ é alterada, com o $Ring$ sendo atravessado pela face positiva.

Lista Estado originador 5:

$$n1n2p5p0p2p6p1bn0p0$$

Lista Efeito da Ação 0 no Estado originador 5:

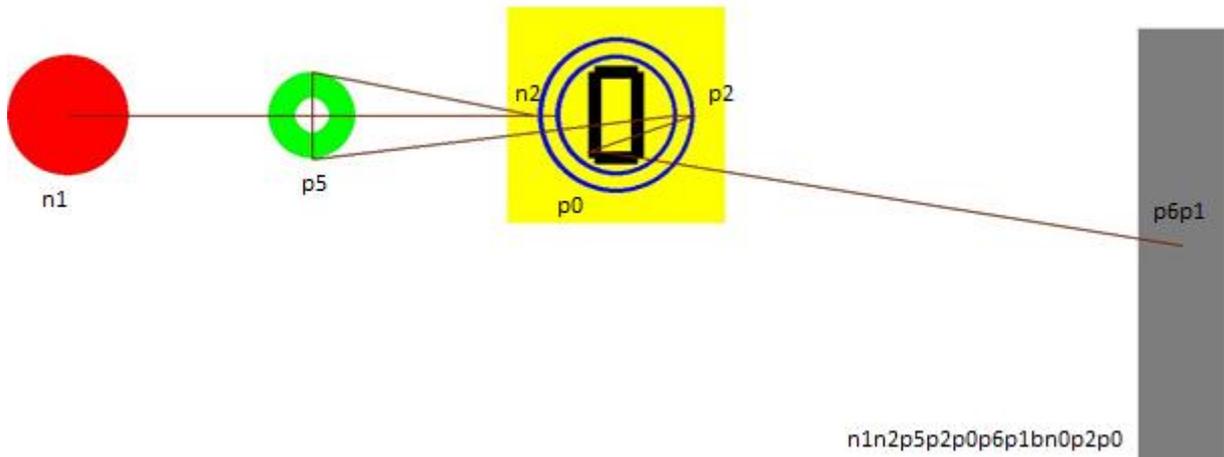
$$n1n2p5p2p0p6p1bn0p2p0$$

Quando em comparação com a lista do Estado originador 5, fica mais complexo e menos intuitivo determinar a quantidade de cruzamentos alterados do que quando a comparação é feita pelos diagramas.

Segundo a Tabela 2, a realização da ação 0 pode alterar apenas o posicionamento do elemento $Ring$, exatamente o que acontece quando aplicada ao Estado originador 5, onde a $chain(Str)$ (seção 3.1) se mantém com a mesma quantidade de cruzamentos e a $chain(Post)$ (seção 3.1) passa a conter o elemento $Ring$.

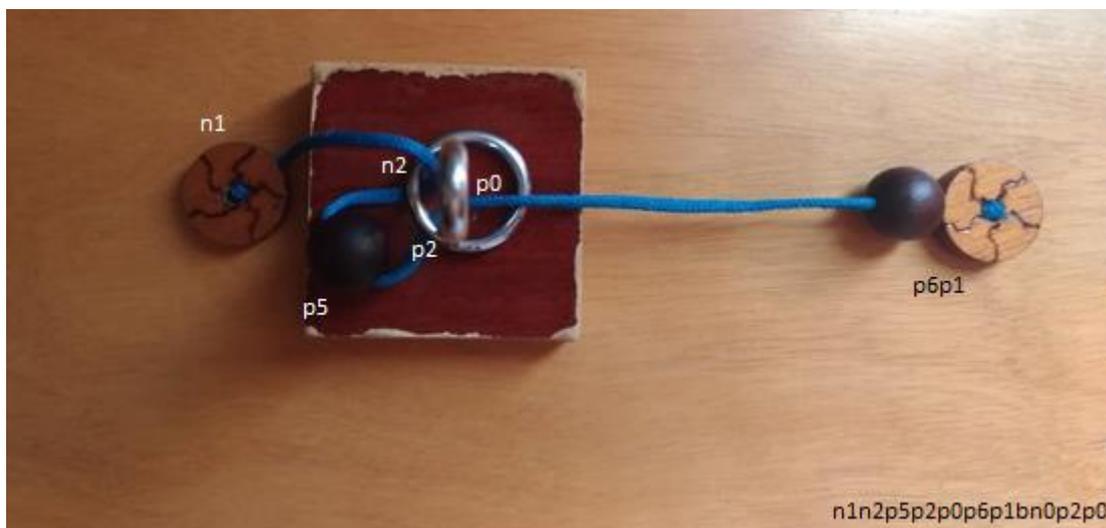
A Figura 48 mostra o efeito da ação 0 no Estado originador 5 quando realizada no quebra-cabeças real.

Figura 47 – Efeito da Ação 0 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 48 – Efeito da Ação 0 no Estado originador 5 – Fisherman's Folly



Fonte: Autor

A aplicação da ação 1 no Estado originador 5, por sua vez, gera mais um cruzamento nas faces +Ring ($p2$) e -Ring ($n2$), o que é possível de acontecer segundo a Tabela 2. Também há alteração no posicionamento fazendo, mais uma vez, parte do conjunto *Base* e *Post*, como é possível verificar quando comparada a Figura 45 (Estado originador 5) com a Figura 49 (resultado da ação 1 no estado originador 5). O

diagrama do novo estado gerado é formado a partir da lista:
 $n1n2p5n2p0p2p2p6p1bn0n2p0$.

Lista Estado originador 5:

$n1n2p5p0p2p6p1bn0p0$

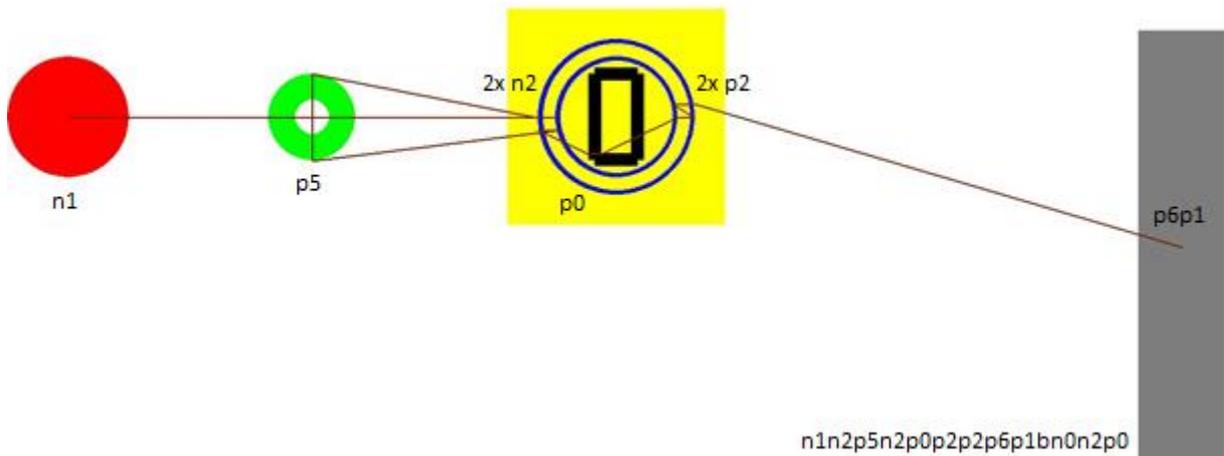
Lista Efeito da Ação 1 no Estado originador 5:

$n1n2p5n2p0p2p2p6p1bn0n2p0$

Nesse caso, a diferença para as listas se dá tanto na *chain(Str)* quanto na *chain(Post)*. Na primeira, temos o acréscimo de dois novos cruzamentos ($p2$ e $n2$) e na segunda a presença do *Ring*, dessa vez atravessado pela face negativa.

Na Figura 50 está representado o efeito da ação 1 no Estado originador 5 quando realizada no quebra-cabeças real.

Figura 49 – Efeito da Ação 1 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 50 – Efeito da Ação 1 no Estado originador 5 – Fisherman’s Folly



Fonte: Autor

Voltando à Tabela 2, é possível verificar que as ações 2 e 3 (Passar o *Disk1* pelo *Post*) alteram apenas a quantidade de faces *PostH*, sendo que, no caso da ação 2, as únicas possibilidades são: acréscimo de cruzamento em *-PostH* ($n0$) e decréscimo em *+PostH* ($p0$). No caso da ação 3, as possibilidades são contrárias: decréscimo de cruzamento em *-PostH* ($n0$) e acréscimo em *+PostH* ($p0$). Essas ações apresentam a solução mais intuitiva porque as variações dos cruzamentos do *Disk1* pelo *Post* não interferem nas demais configurações do diagrama. Tendo como base o mesmo Estado originador 5 (Figura 45) utilizado para exemplificar as ações 0 e 1, nas Figura 51 e Figura 53 é possível ver o resultado das ações 2 e 3, respectivamente.

A ação 2 no Estado originador 5 apenas gera um cruzamento a mais em *-PostH*, o que é possível perceber pela inclusão de um cruzamento $n0$ na lista gerada para o resultado dessa ação, que é representada por: $n1n0n2p5p0p2p6p1bn0p0$. A representação diagramática desse resultado está na Figura 51 e a representação no quebra-cabeças real está na Figura 52.

Enquanto isso, a ação 3 no Estado originador 5 gera apenas um cruzamento a mais em *+PostH* (inclusão de $p0$ na lista), a lista gerada para o resultado dessa ação é: $n1p0n2p5p0p2p6p1bn0p0$. A representação diagramática do resultado da ação 3

está na Figura 53, enquanto a representação no quebra-cabeças real está na Figura 54.

Lista Estado originador 5:

$n1n2p5p0p2p6p1bn0p0$

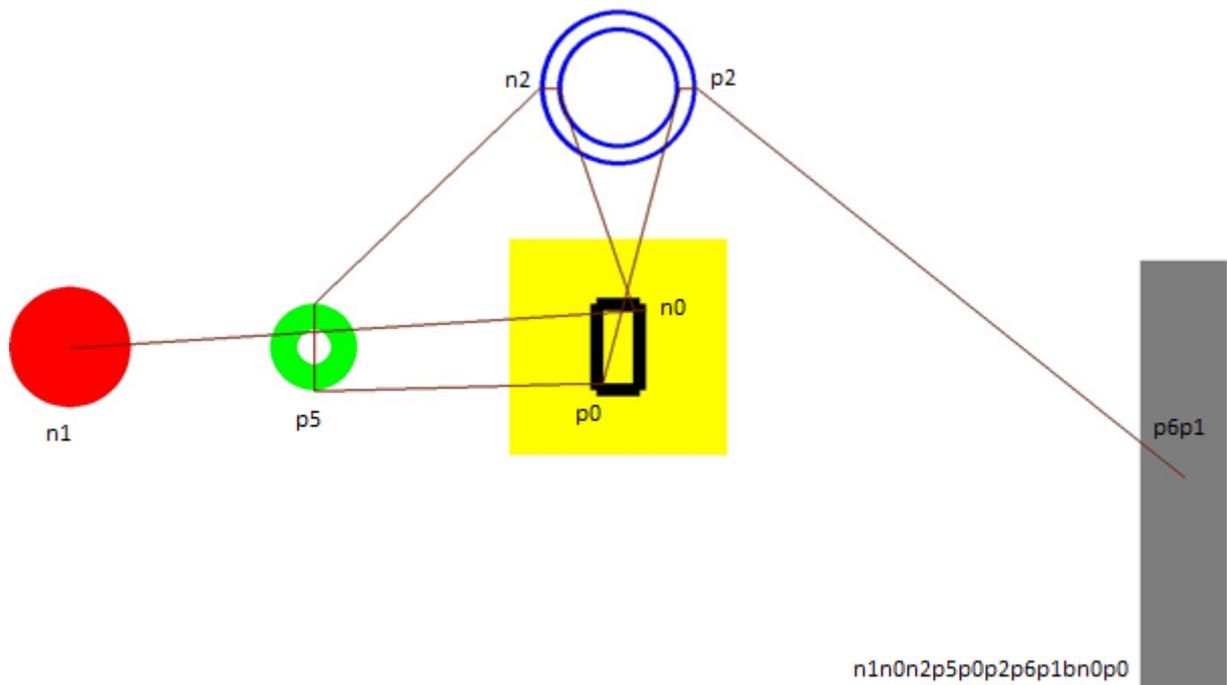
Lista Efeito da Ação 2 no Estado originador 5:

$n1n0n2p5p0p2p6p1bn0p0$

Lista Efeito da Ação 3 no Estado originador 5:

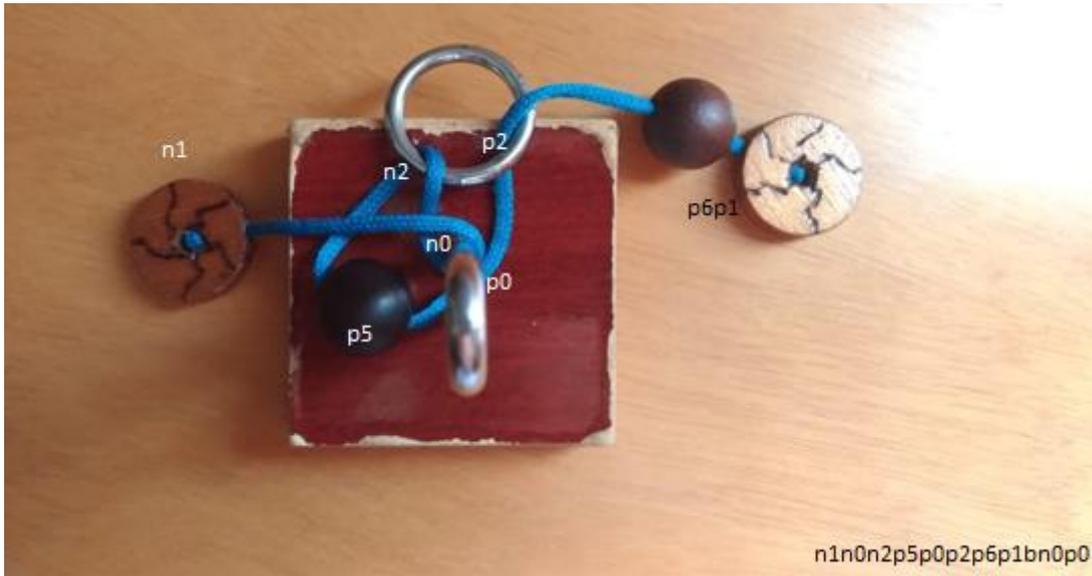
$n1p0n2p5p0p2p6p1bn0p0$

Figura 51 – Efeito da Ação 2 no Estado originador 5 – Representação Diagramática



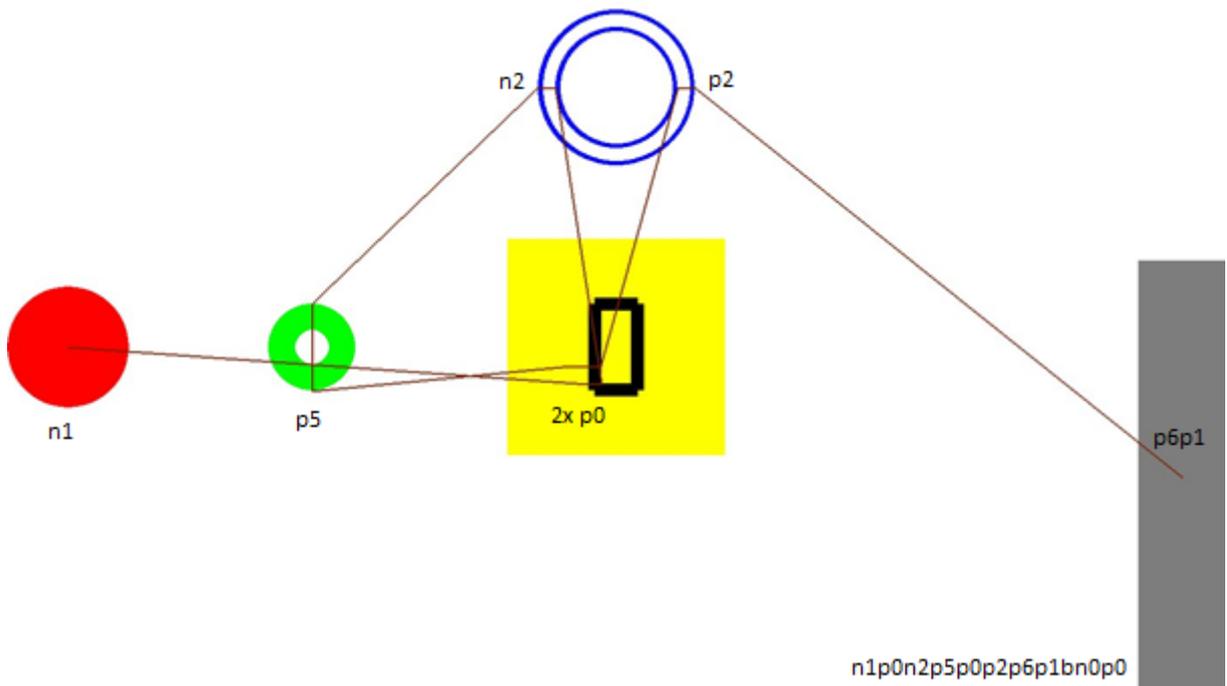
Fonte: Autor

Figura 52 – Efeito da Ação 2 no Estado originador 5 – Fisherman's Folly



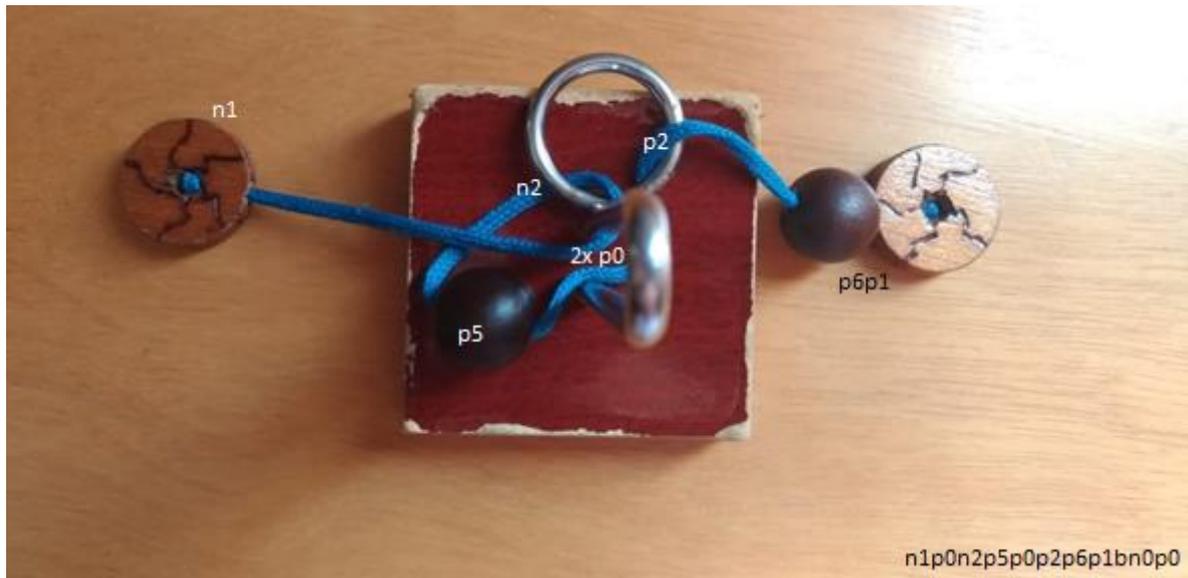
Fonte: Autor

Figura 53 – Efeito da Ação 3 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 54 – Efeito da Ação 3 no Estado originador 5 – Fisherman’s Folly



Fonte: Autor

As ações 6 e 7 (Passar *Sphere1* pelo *Ring*) (Tabela 2) podem aumentar ou diminuir apenas um cruzamento pelas faces +Ring ($p2$) e -Ring ($n2$), como mostrado na Tabela 2. Nas situações onde já houver algum cruzamento entre *Sphere1* e *Ring* no sentido contrário à ação, haverá diminuição da quantidade de faces, nos demais aumento. Exceção ao caso particular onde a ação 6 realizada no Estado originador 5 não gera nenhuma variação em cruzamento pelas faces, apenas no posicionamento da *Sphere1* com relação ao *Ring*. Esse caso particular é uma característica da configuração que permite a *Sphere1* a correr livremente pela *String* apenas atravessando o buraco do *Ring*. Esse caso particular se repete pela ação 6 nos Estados 19 e 20, que são derivados do Estado originador 5. Utilizando como exemplo o mesmo Estado originador 5 (Figura 45), verifica-se na Figura 55 o efeito da ação 6 (caso particular), cujo a lista que descreve a disposição dos elementos é: $n1p5n2p0p2p6p1bn0p0$, já na Figura 57, verifica-se o efeito da ação 7, cujo a lista que descreve a disposição dos elementos é: $n1n2n2p5p2p0p2p6p1bn0p0$. A ação 7 aumenta um cruzamento pelas faces +Ring ($p2$) e -Ring ($n2$).

A Figura 56 e a Figura 58 são as representações dos efeitos, respectivamente, das ações 6 e 7 no Estado originador 5.

Lista Estado originador 5:

$n1n2p5p0p2p6p1bn0p0$

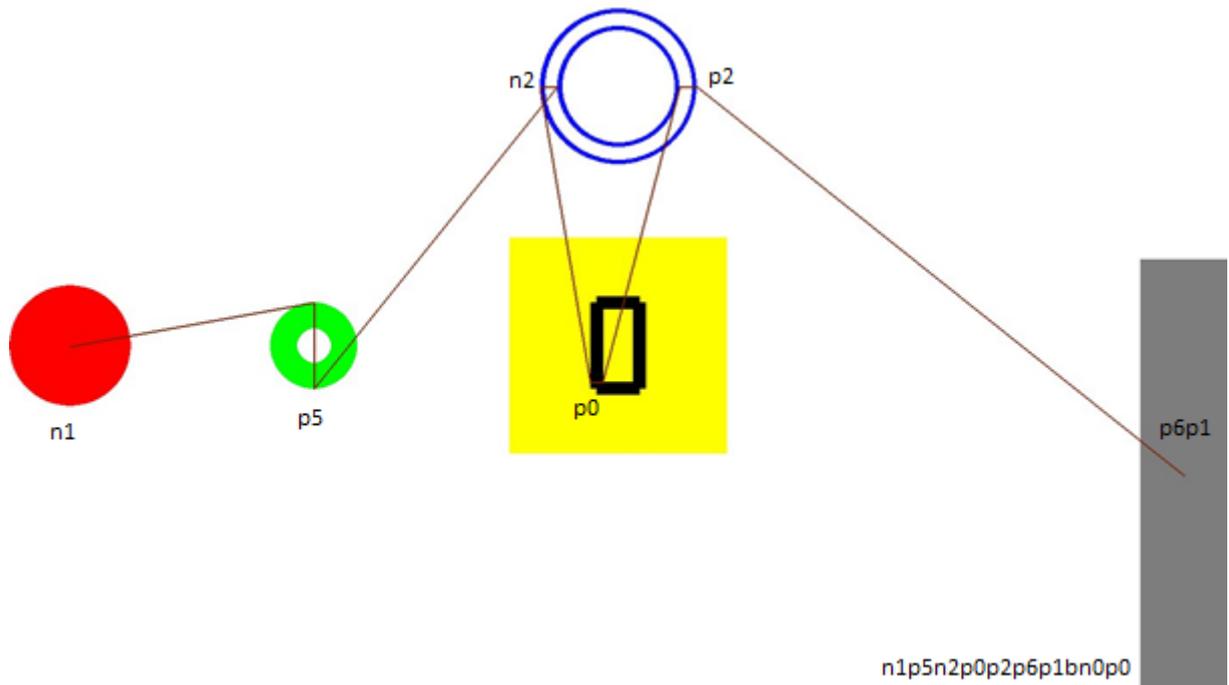
Lista Efeito da Ação 6 no Estado originador 5:

$n1p5n2p0p2p6p1bn0p0$

Lista Efeito da Ação 7 no Estado originador 5:

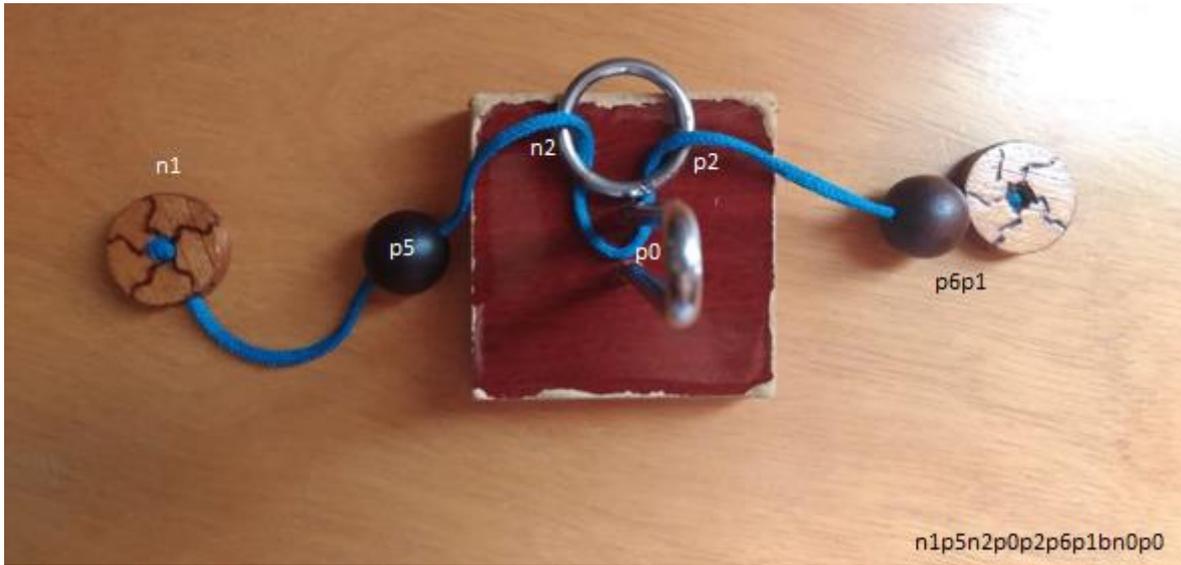
$n1n2n2p5p2p0p2p6p1bn0p0$

Figura 55 – Efeito da Ação 6 no Estado originador 5 – Representação Diagramática



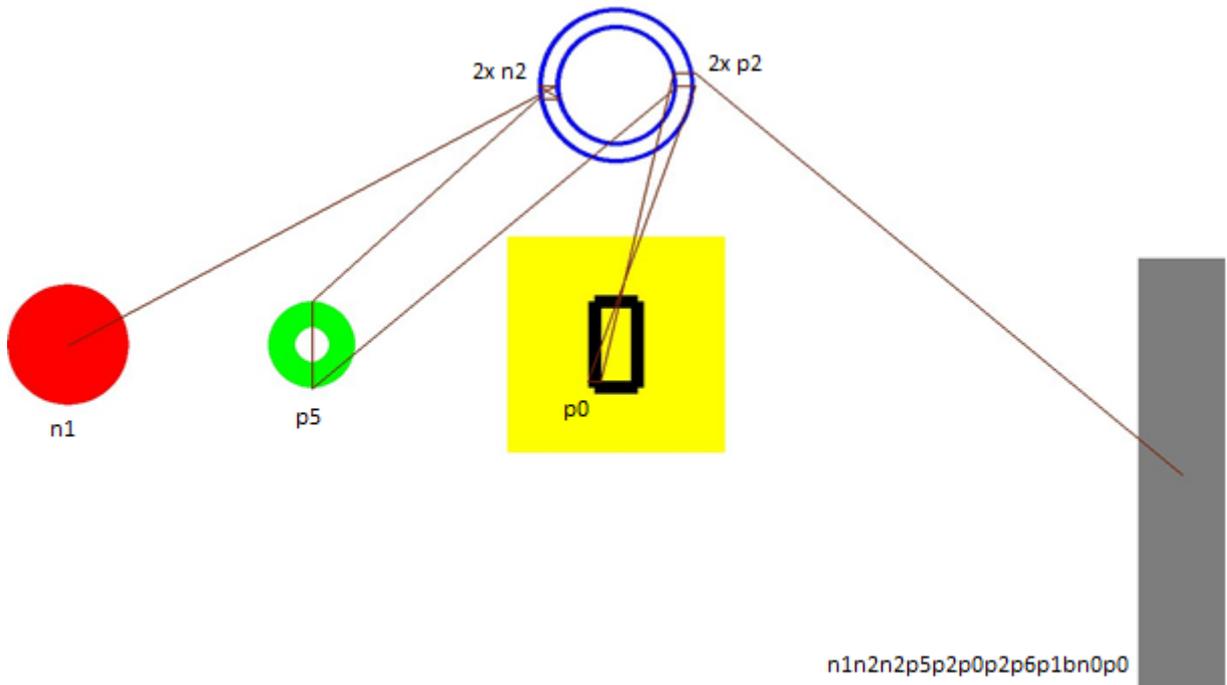
Fonte: Autor

Figura 56 – Efeito da Ação 6 no Estado originador 5 – Fisherman's Folly



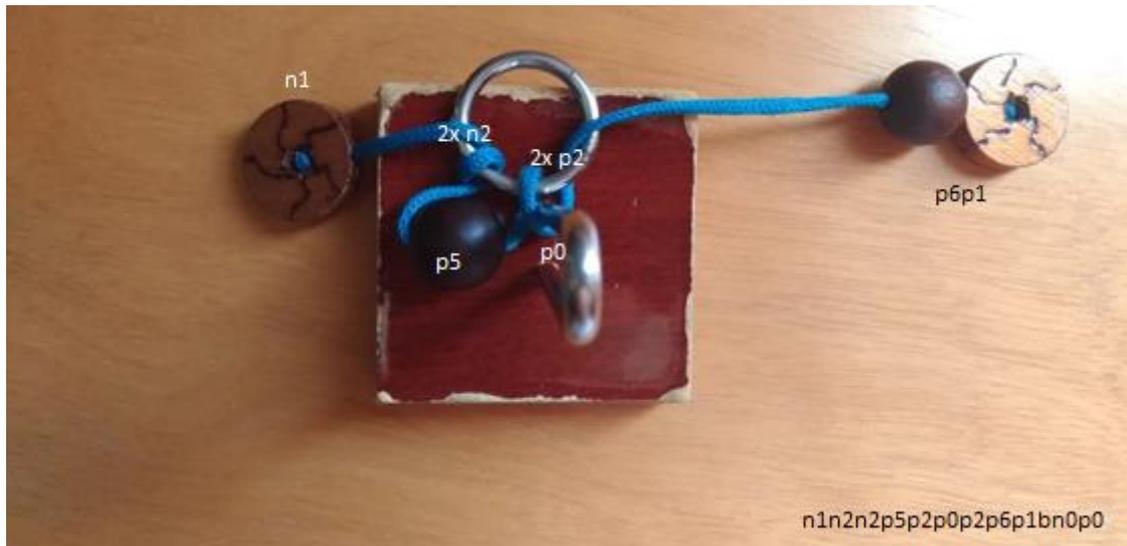
Fonte: Autor

Figura 57 – Efeito da Ação 7 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 58 – Efeito da Ação 7 no Estado originador 5 – Fisherman's Folly



Fonte: Autor

Segundo a Tabela 2, também verificamos que as ações 10 e 11 (Passar *Ring* pelo *Post*) gera acréscimo e decréscimo simultâneo na quantidade de faces +PostH ($p0$) e -PostH ($n0$). Irá diminuir a quantidade apenas quando, em ações prévias, houve uma configuração na qual o *Ring* ficou situado em posição contrária à ação atual, no restante das configurações irá aumentar. As situações onde haverá o aumento de dois cruzamentos ou nenhum aumento dependem das configurações anteriores da *Sphere1* com o *Ring*, que pode gerar dois cruzamentos dependendo do lado para qual o *Ring* está atravessando, devido a *Sphere1* não poder atravessar o PostH. Partindo do mesmo Estado originador 5 (Figura 45) utilizado nas exemplificações anteriores, o oASP(MDP), em sua busca, realizou as ações 10 e 11.

A ação 10 gera a lista: $n1p0n2n0p5p0p0p2n0p6p1bn0p0$ e está representada diagramaticamente na Figura 59, enquanto a Figura 60 representa o efeito da ação 10 no quebra-cabeças real.

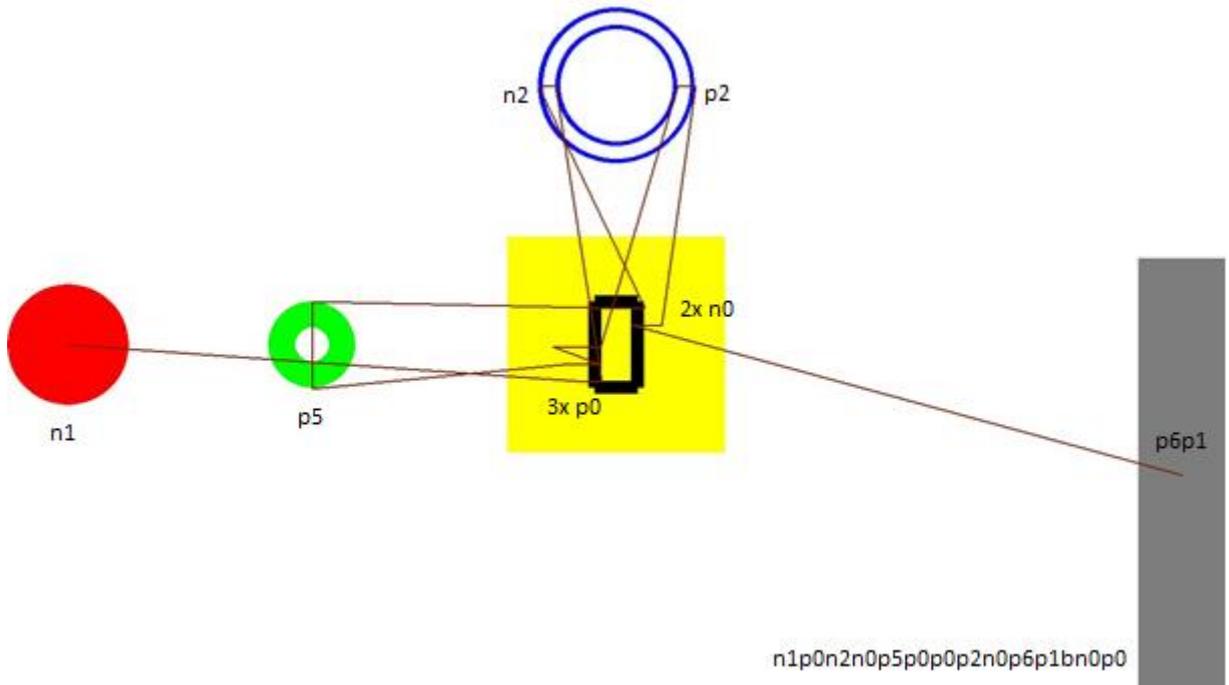
Lista Estado originador 5:

$$n1n2p5p0p2p6p1bn0p0$$

Lista Efeito da Ação 10 no Estado originador 5:

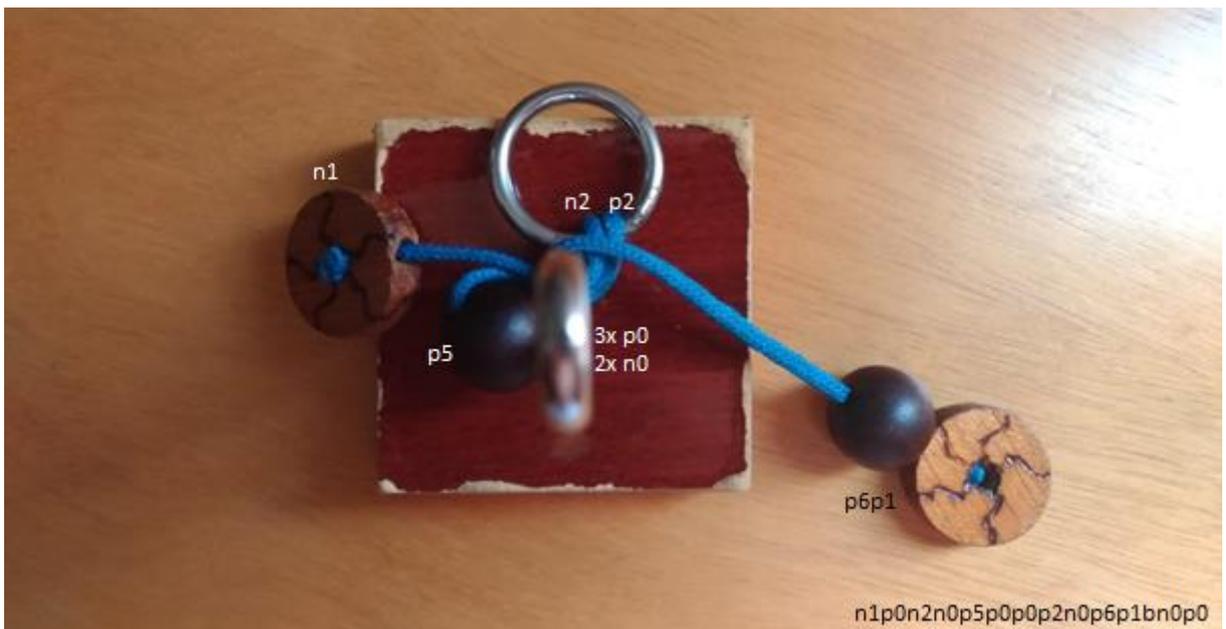
$$n1p0n2n0p5p0p0p2n0p6p1bn0p0$$

Figura 59 – Efeito da Ação 10 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 60 – Efeito da Ação 10 no Estado originador 5 – Fisherman's Folly



Fonte: Autor

A ação 11 gera a lista: $n1n0n2p0p5p2p0p6p1bn0p0$ e está representada diagramaticamente na Figura 61, enquanto a Figura 62 apresenta a representação do quebra-cabeças real.

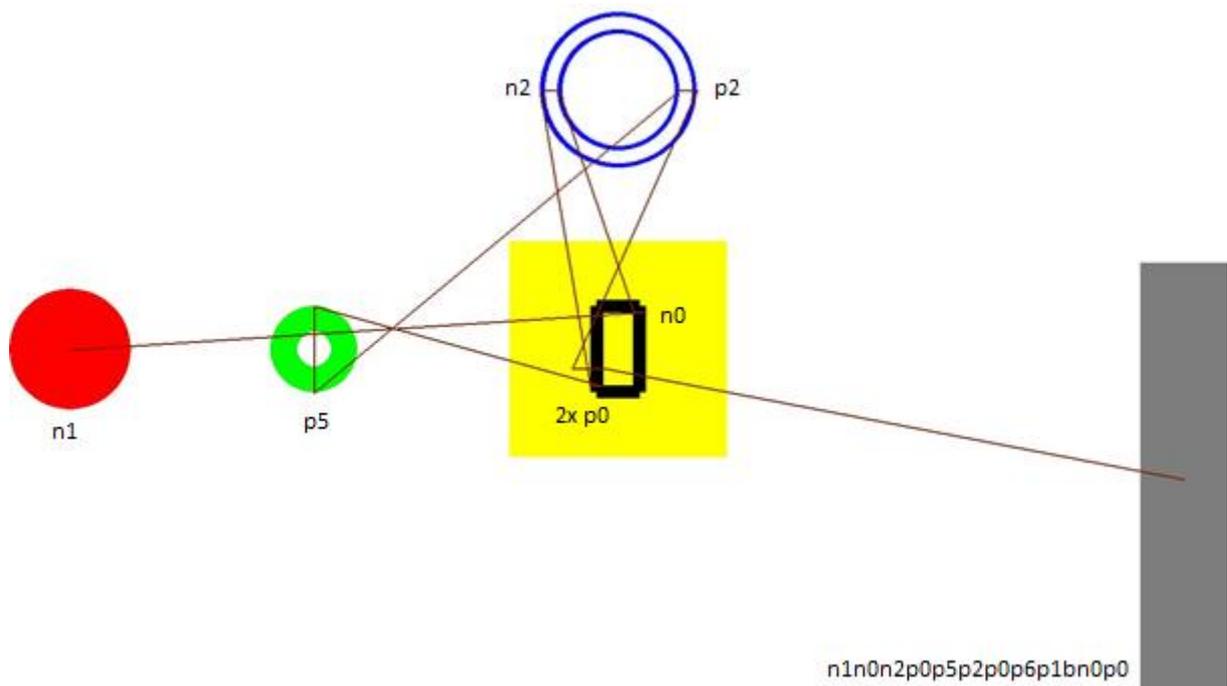
Lista Estado originador 5:

$n1n2p5p0p2p6p1bn0p0$

Lista Efeito da Ação 10 no Estado originador 5:

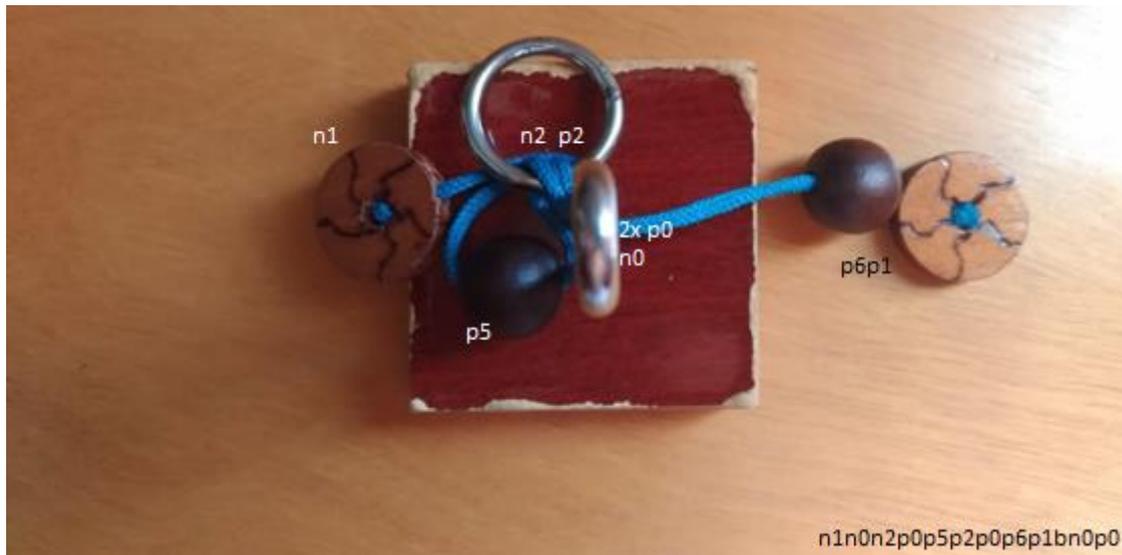
$n1n0n2p0p5p2p0p6p1bn0p0$

Figura 61 – Efeito da Ação 11 no Estado originador 5 – Representação Diagramática



Fonte: Autor

Figura 62 – Efeito da Ação 11 no Estado originador 5 – Fisherman’s Folly



Fonte: Autor

Observando as Figuras que representam o quebra-cabeças real é possível ver que, conforme a quantidade de ações aumenta, a quantidade de cruzamentos também tende a aumentar e isso faz com que a distância entre os objetos no quebra-cabeças real diminua por causa da restrição física da corda, o que dificulta a visualização dos cruzamentos. O diagrama se destaca nesse ponto por ter uma distância fixada entre os objetos, focando apenas na quantidade de cruzamentos e deixando a limitação da corda ser determinada pelo algoritmo oASP(MDP) que decide se a quantidade de nós que se formará com a aplicação de determinada ação é factível.

Em resumo, o Passo Indutivo consiste primeiro na aplicação de todas as ações para cada um dos 30 *estados originadores*, em seguida a comparação entre os estados básicos com seus respectivos estados subsequentes. Considerando o quebra-cabeças de Fisherman’s Folly simplificado, são 8 ações para cada um dos 30 estados, o que totaliza 270 diagramas, sendo 240 comparações entre estados. Foi verificado empiricamente que só há a possibilidade do aumento de cruzamentos pelas faces dos elementos em +Ring e -Ring (pelas ações 0, 1, 6 e 7) e em +PostH e -PostH (pelas ações 2, 3, 10 e 11), conforme mostrado na Tabela 2, ou seja, apesar da *Sphere1* também ser um elemento com buraco é fisicamente impossível que a corda a atravesse mais de uma vez.

Observando os 30 *estados originadores* listados, conclui-se que os 10 primeiros são estados geradores dos outros 20 por causa da diferença entre eles ser apenas a movimentação do *Disk1*, o que varia somente o início da lista. Isto é, as ações 2 e 3 são responsáveis por gerar os Estados 11 e 12 quando aplicadas no Estado 1, também geram os Estados 13 e 14 quando aplicadas no Estado 2, e assim sucessivamente. Com base nessa observação, foi montada a Tabela 5 com todos os resultados encontrados pelo oASP(MDP) para os 10 primeiros *estados originadores*, que são lidos da mesma maneira que os da Tabela 2.

As quantidades de cruzamentos gerados pelas ações podem aumentar de acordo com a quantidade de ações ou nós que o estado atual possua, como é o caso das ações 0 e 1 gerarem +2 cruzamentos nos buracos +Ring e -Ring (conforme visto na Tabela 2), no entanto, essa situação não é observada nos 10 primeiros *estados originadores* (Tabela 5).

A Tabela 5 apresenta os efeitos de cada ação realizada pelo algoritmo oASP(MDP) nos 10 primeiros *estados originadores* que, como visto anteriormente, são responsáveis por gerar todas as outras configurações que são fisicamente possíveis de serem construídas no quebra-cabeças real de Fisherman's Folly.

Analisando os efeitos das ações aplicadas pelo oASP(MDP) nos 10 primeiros *estados originadores* (Tabela 5), é possível ver que os resultados obtidos são correspondentes aos previstos pela Tabela 2 que, de acordo com os Lemas I e II, garante que todos os estados encontrados pelo oASP(MDP) são possíveis de serem fisicamente construídos.

Partindo das conclusões acima, é possível provar o Passo Indutivo com base nos cruzamentos da corda com os elementos, que são gerados a partir da aplicação das ações. Isto é, as soluções π encontradas pelo algoritmo A solucionam o quebra-cabeças F buscando apenas estados factíveis fisicamente e, com isso, também é possível garantir a prova de correção do algoritmo oASP(MDP) utilizando o método da indução matemática com diagramas.

Os diagramas que representam os 10 principais *estados originadores* podem ser encontrados no Apêndice A.

Tabela 5 – Estados Originadores e os efeitos de cada ação

		ESTADOS				
AÇÕES:	1	2	3	4	5	
0	=	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	+1 [+Ring, -Ring]	Ring Position	
1	+1 [+Ring, -Ring]					
2	+1 [-PostH]					
3	+1 [+PostH]					
6	=	+1 [+Ring, -Ring]	+1 [+Ring, -Ring]	+1 [+Ring, -Ring]	0	
7	=	0	+1 [+Ring, -Ring]	+1 [+Ring, -Ring]	+1 [+Ring, -Ring]	
10	=	+1 [+PostH, -PostH]	=	-1 [+PostH, -PostH]	+2 [+PostH, -PostH]	
11	=	+1 [+PostH, -PostH]	-1 [+PostH, -PostH]	=	+1 [+PostH, -PostH]	
	6	7	8	9	10	
0	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	Ring Position	+1 [+Ring, -Ring]	Ring Position	
1	+1 [+Ring, -Ring]	Ring Position	+1 [+Ring, -Ring]	Ring Position	+1 [+Ring, -Ring]	
2	+1 [-PostH]					
3	+1 [+PostH]					
6	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	
7	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	+1 [+Ring, -Ring]	-1 [+Ring, -Ring]	
10	=	=	=	+1 [+PostH, -PostH]	+1 [+PostH, -PostH]	
11	+2 [+PostH, -PostH]	+1 [+PostH, -PostH]	+1 [+PostH, -PostH]	=	=	

Fonte: Autor

6 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo deste trabalho foi a investigação de representações diagramáticas para a construção de uma prova de correção por indução matemática do oASP(MDP). Esta prova tem como objetivo a verificação de que os estados alcançados pelo algoritmo de solução do Fisherman's Folly apresentam estados fisicamente possíveis.

A maioria dos esforços em Representação de Conhecimento tem focado na manipulação simbólica e na formulação lógica. O uso de uma representação formal é conveniente para raciocínio automático, tendo em vista que as linguagens de computação promovem ferramentas excelentes para processamento e representações simbólicas.

Embora representações simbólicas sejam mais simples para uma interpretação automática por um computador, representações diagramáticas são mais intuitivas para o entendimento do ser humano. Diagramas possuem a vantagem de substituir nomes de componentes e identificadores por estruturas gráficas que não exigem enumeração. As formalizações simbólicas usadas para os quebra-cabeças são acompanhadas por representações diagramáticas, como por exemplo a Figura 10, que apresenta a solução do quebra-cabeças de Fisherman Folly. A princípio, essa representação é meramente ilustrativa, isto é, ainda não havia sido utilizada uma representação formal e não há tarefas computacionais associadas.

Este trabalho explorou o tratamento desses diagramas como opções às formalizações simbólicas de domínios para solucionadores automáticos de problemas. Foi tomada como perspectiva para visualização do quebra-cabeças uma vista de cima (planta), por ter sido empiricamente verificado, quando comparada com uma vista frontal, que facilitaria a visualização dos estados conforme a aplicação de ações. Com a utilização desses diagramas foi possível a análise da prova de correção por indução matemática das soluções encontradas pelo algoritmo oASP(MDP).

Essa prova de correção traz a garantia de que o solucionador automático realiza a busca pela solução apenas em estados que são fisicamente possíveis de serem construídos, durante a manipulação do quebra-cabeças real. Além disso, esta pesquisa será base para desenvolvimento de um solucionador automático de quebra-cabeças inteiramente baseado em representações diagramáticas. Esta tarefa,

entretanto, será considerada em trabalhos futuros, podendo também serem incluídas visualizações em 3D, o que permite abranger todas as vistas do quebra-cabeças, por consequência melhorando a estética da representação diagramática.

A utilização de diagramas no método da indução matemática garantiu a visualização de que o algoritmo oASP(MDP) alcança estados que são fisicamente possíveis de serem construídos. Um dos trabalhos futuros possíveis seria a utilização da mesma representação diagramática como prova de correção de outros quebra-cabeças espaciais, como por exemplo o Rope Ladder (CABALAR e SANTOS, 2011) que também contém objetos semelhantes (como cordas e buracos) porém em um domínio mais completo. Também é possível garantir o funcionamento de outros algoritmos que também solucionem o mesmo domínio. Para a aplicação deste método em outros quebra-cabeças espaciais ou outros algoritmos seria suficiente uma adaptação do código fonte (seção 4).

Entretanto, a representação diagramática é uma ferramenta que pode ser utilizada também para o desenvolvimento de um software que lide inteiramente com diagramas, onde haja interpretações dos diagramas e um possível aprendizado partindo da representação diagramática, sem que haja a necessidade de um algoritmo específico.

REFERÊNCIAS

ABDEL-RAOUF, O.; ABDEL-BASET, M.; HENAWY, I. An improved flower pollination algorithm with chaos. **IJEME**, v. 4, n. 2, p. 1-8, Ago. 2014.

BARAL, C.; DZIFCAK, J. Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation. In: Principles of Knowledge Representation and Reasoning, 2012, Roma. **Proceedings...** Roma: 2012. p. 10-14.

BARAL, C.; MITRA, A. Learning to automatically solve logic grid puzzles. In: EMNLP, 2015, Lisboa. **Proceedings...** Lisboa: 2015. p. 1023-1033.

BENNETT, Brandon et al. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. **Applied Intelligence**, v. 17, n. 3, p. 239-251, Nov. 2002.

BROWN, James. **Philosophy of mathematics: a contemporary introduction to the world of proofs and pictures**. 2. ed. Nova York e Londres: Routledge, 2008.

CHANDRASEKARAN, B.; KURUP, U.; BANARJEE, B. A diagrammatic reasoning architecture: design, implementation and experiments. In: AAAI SPRING SYMPOSIUM, 2005, Califórnia. **Proceedings...** Califórnia: 2005. p. 108-113.

CABALAR, P.; SANTOS, P. Strings and holes: an exercise on spatial reasoning. In: IBERAMIA, 10. e SBIA, 18., 2006, Ribeirão Preto. **Proceedings...** Ribeirão Preto: Springer, 2006. p. 419-429.

CABALAR, P.; SANTOS, P. Formalising the fisherman's folly puzzle. **Artificial Intelligence**, v. 175, n. 1, p. 346–377, Jan. 2011.

CABALAR, P.; SANTOS, P. A qualitative spatial representation of string loops as holes. **Artificial Intelligence**, v. 238, p. 1-10, Maio 2016.

CABALAR, P. Notas de Aula. Corunna University, 2017.

CAYLI, Merve et al. Solving challenging grid puzzles with answer set programming. In: ASP, 2007. **Proceedings...** p. 175-190.

COHN, A. G.; RENZ, J. Qualitative spatial representation and reasoning. In: VAN HARMELEN, F.; LIFSCHITZ, V.; PORTER B. **Handbook of Knowledge Representation**. Amsterdã e Oxford: Elsevier, 2008. p. 551–596.

DE TOFFOLI, S.; GIARDINO, V. Forms and roles of diagrams in knot theory. **Erkenntnis**, v. 79, n. 3, p. 829-842, Nov. 2014.

DE TOFFOLI, S.; GIARDINO, V. An inquiry into the practice of proving in low-dimensional topology. In: LOLLI, G.; PANZA, M.; VENTURI, G. **From Logic to Practice**. Berlim: Springer, 2015, p. 315-336.

DOHERTY, Patrick et al. (TAL) temporal action logics: language specification and tutorial. **Electronic Transactions on Artificial Intelligence**, v. 3-4, p. 273–306, 1998. Disponível em: <<http://www.ep.liu.se/ej/etai/1998/009/>>. Acesso em: 26, fev. 2019.

DROGOUL, A.; DUBREUIL C. Eco-problem-solving model: results of the n-puzzle. In: DAI Workshop, 13., 1993, Seattle. **Proceedings...** Seattle: 1993.

EITER, Thomas et al. Answer set programming: a primer. In: TESSARIS, S.; EITER, T.; GUTIERREZ, C.; HANDSCHUH, S.; ROUSSET, M.; SCHMIDT, R. **Lecture Notes in Computer Science**. 5689. ed. Springer, 2009, p. 40-110.

EI-KHATIB, O. Solving the pixel puzzle under answer set programming. **IJCS**, v. 3, n. 3, p. 200-206, Mar. 2016.

FERREIRA, Leonardo A. et al. A method for the online construction of the set of states of a Markov Decision Process using Answer Set Programming. In: MOUHOU M.; SADAQUI S.; AIT MOHAMED O.; ALI M. In: **Recent Trends and Future Technology in Applied Intelligence**. Cham: Springer, 2018. p. 3-15.

FERREIRA, Leonardo A. et al. Answer set programming for non-stationary Markov decision processes. **Applied Intelligence**, v. 47, n. 4, p. 993-1007, Dez. 2017.

FREITAS, T. et al. Solving a spatial puzzle using Answer Set Programming integrated with Markov Decision Process. In: BRACIS, 7., 2018, São Paulo. **Proceedings...** São Paulo: CPS, 2018. p. 528-533.

GELFOND, M.; LIFSCHITZ, V. The stable models semantics for logic programming. In: ICLP, 5., 1988, Cambridge. **Proceedings...** Cambridge: MIT Press, 1988. p. 1070–1080.

GELFOND, M.; LIFSCHITZ, V. Action languages. **Electronic Transactions on Artificial Intelligence**, v. 3-4, p. 193–210, 1998. Disponível em: <<http://www.ep.liu.se/ej/etai/1998/007/>>. Acesso em: 27, fev. 2019.

HAMMACK, Richard. **Book of Proof**. 2. ed. Richmond: Richard Hammack, 2013.

JAMNIK, M.; BUNDY, A.; GREEN, I. On automating diagrammatic proof of arithmetic arguments. **Journal of Logic, Language and Information**, v. 8, n. 3, p. 297-321, Jul. 1999.

JAMNIK, M.; STAPLETON, G.; SHIMOJIMA, A. Effective representation of information: generalizing free rides. In: JAMNIK, M.; UESAKA, Y.; SCHWARTZ, S. **Lecture Notes in Artificial Intelligence**. 9781. ed. Springer, 2016. p. 296-299.

KENDALL, G.; PARKES, A.; SPOERER, K. A survey of NP-complete puzzles. **ICGA**, v. 31, n. 1, p. 13-34, Mar. 2008.

KOWALSKI, R.; SERGOT, M. A logic-based calculus of events. **New Generation Computing**, v. 4, p. 67–95, 1986.

LEVESQUE, H.; PIRRI, F.; REITER, R. Foundations for the situation calculus. **Electronic Transactions on Artificial Intelligence**, v. 3-4, p.159–178, 1998. Disponível em <<http://www.ep.liu.se/ej/etai/1998/005/>>. Acesso em 27, fev. 2019.

LIGOZAT, Gérard. **Qualitative spatial and temporal reasoning**. Paris: ISTE, 2012.

LIFSCHITZ, Vladimir. What Is Answer Set Programming?. In: AAAI, 23., 2008, Chicago. **Proceedings...** Chicago: AAAI Press, 2008. p. 1594-1597

MCCARTHY, John. AI as sport. **Science**. Nova York, v. 276, n. 5318, p.1518 – 1519, Jun. 1997.

MCCARTHY, John. Elaboration tolerance. In: Common Sense 98, 4., 1998, Londres. **Proceedings...** Londres: 1998. p. 198-217.

MCCARTHY, J.; HAYES, P. Some philosophical problems from the standpoint of artificial intelligence. **Machine Intelligence Journal**, v. 4, p. 463–512, 1969.

NELSEN, Roger B. **Proofs without words: exercises in visual thinking**. 1. ed. Washington: Mathematical Association of America, 1993.

NELSEN, Roger B. **Proofs without words: further exercises in visual thinking**. 3. ed. Washington: Mathematical Association of America, 2015.

NEWELL, A.; SHAW, J.; SIMON, H. **Report on a general problem-solving program**. 1959.

NILSSON, Nils. **Artificial Intelligence: a new synthesis**. São Francisco: Morgan Kaufmann, 1998.

PEARCE, David. A new logical characterisation of stable models and answer sets. In: NMELP, 2., 1996, Bad Honnef. **Proceedings...** Bad Honnef: 1996. p. 57-70.

PEARCE, D.; VALVERDE, A. Quantified equilibrium logic and foundations for answer set programs. In: ICLP, 24., 2008, Udine. **Proceedings...** Udine: Springer, 2008. p. 546-560.

PEIRCE, Charles S. Manuscripts on existential graphs. **Collected Papers of Charles Sanders Peirce**, v. 4, p. 320-410, 1906.

PÉREZ, C.; PÉREZ, G.; CABALAR, P. **A diagrammatic reasoning tool for logic programming**. 2017. 93 f. Trabalho de Fin de Grao (Enxeñaría Informática) — Universidade da Coruña, Espanha, 2017.

REIDEMEISTER, Kurt. **Knoten Theorie**. Germany: Springer, 1932. Tradução BCS Associates, 1983.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence: a modern approach**. 3. ed. Nova Jersey: Pearson, 2009.

SANTOS, P.; CABALAR, P. Passing through holes and getting entangled by strings: an automated solution for a spatial puzzle. In: ECAI, 17., 2006, Riva del Garda. **Proceedings...** Riva del Garda: Editora, 2006. p.

SANTOS, P.; CABALAR, P. Holes, knots and shapes: a spatial ontology of a puzzle. In: Common Sense 07, 8., 2007, Stanford. **Proceedings...** Stanford: 2007.

SANTOS, P.; CABALAR, P. The space within fisherman's folly: playing with a puzzle in mereotopology. **Spatial Cognition & Computation**, v. 8, n. 1-2, p. 47–64, 2008.

SANTOS, P.; CABALAR, P. Knots world: an investigation of actions, change and space. In: ECAI, 20., 2012, Montpellier. **Proceedings...** Montpellier: 2012. p. 36-44.

SANTOS, P.; CABALAR, P. An investigation of actions, change, space. In: ICAPS, 23., 2013, Roma. **Proceedings...** Roma: 2013.

SANTOS, P.; CABALAR, P. An investigation of actions, change, space within a hole-loop dichotomy. In: Common Sense 13, 11., 2013. Ayia Napa. **Proceedings...** Ayia Napa: 2013.

SANTOS, P.; CABALAR, P. Framing holes within a loop hierarchy. **Spatial Cognition & Computation**, v. 16, n. 1, p. 54–95, 2016.

SCHEUTZ, M.; SLOMAN, A.; LOGAN, B. Emotional states and realistic agent behaviour. In: GAME-ON, 1., 2000, Londres. **Proceedings...** Londres: 2000.

SIMON, H.; SCHAEFFER, J. The game of chess. In: AUMANN, H., **Handbook of Game Theory with Economic applications**, 1 ed. North Holland, 1992. p. 1–17.

SHIN, Sun-Joo. **The iconic logic of Peirce's graphs**. Massachussets: Bradford Book, 2002.

SOWA, J. F. Peirce's tutorial on existential graphs. **Semiotica**, v. 186, n. 1-4, p. 345-394, 2011.

STOCK, Oliviero. **Spatial and temporal reasoning**. Massachussets: Kluwer Academic, 1997.

SLOMAN, A. The empirical case for two systems of reasoning. **Psychological Bulletin**, v. 119, n. 1, p. 3-22, 1996.

SLOMAN, A. Diagrams in the mind. In: Thinking with Diagrams Conference, 1998. Aberystwyth. **Proceedings...** Aberystwyth: 1998.

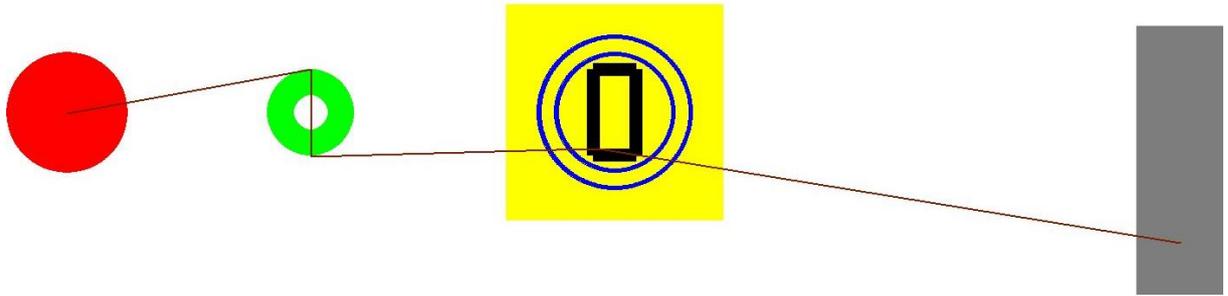
THIELSCHER, M. Introduction to the fluent calculus. **Electronic Transactions on Artificial Intelligence**, v. 2, n. 3-4, p. 179–192, 1998.

VELLEMAN, D. **How to prove it: a structured approach**. Cambridge: Cambridge University Press, p. 245-283. 1994.

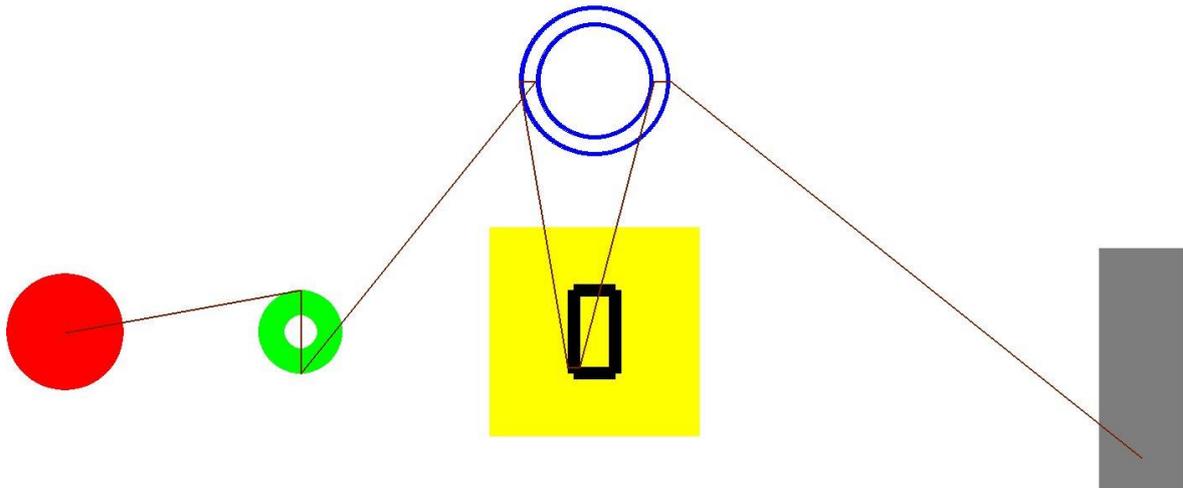
APÊNDICE A

Conforme destacado na seção 5.2, abaixo estão os diagramas que representam os 10 *estados originadores*:

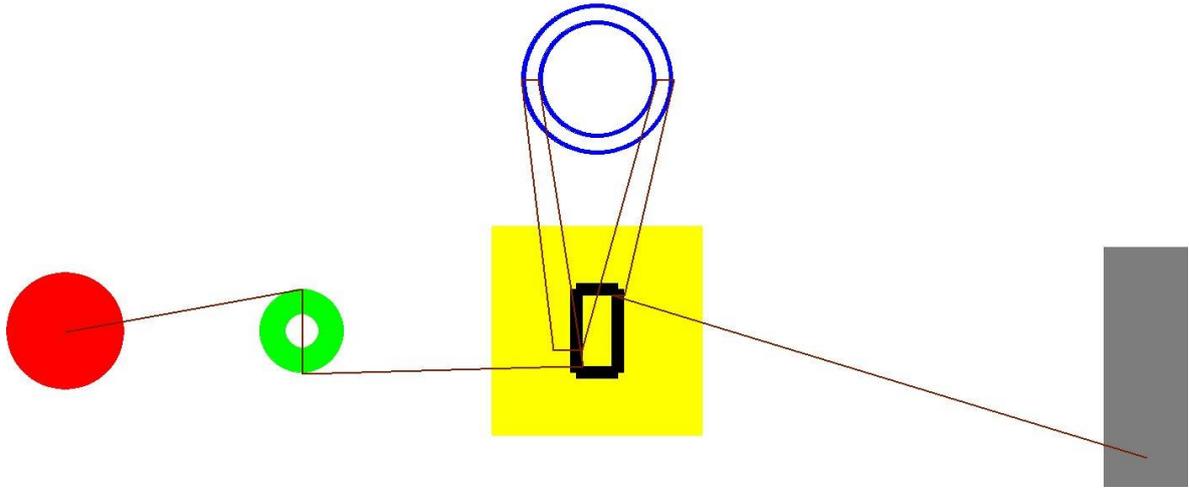
Estado S1 (n1p5p0p6p1bn0p2p0):



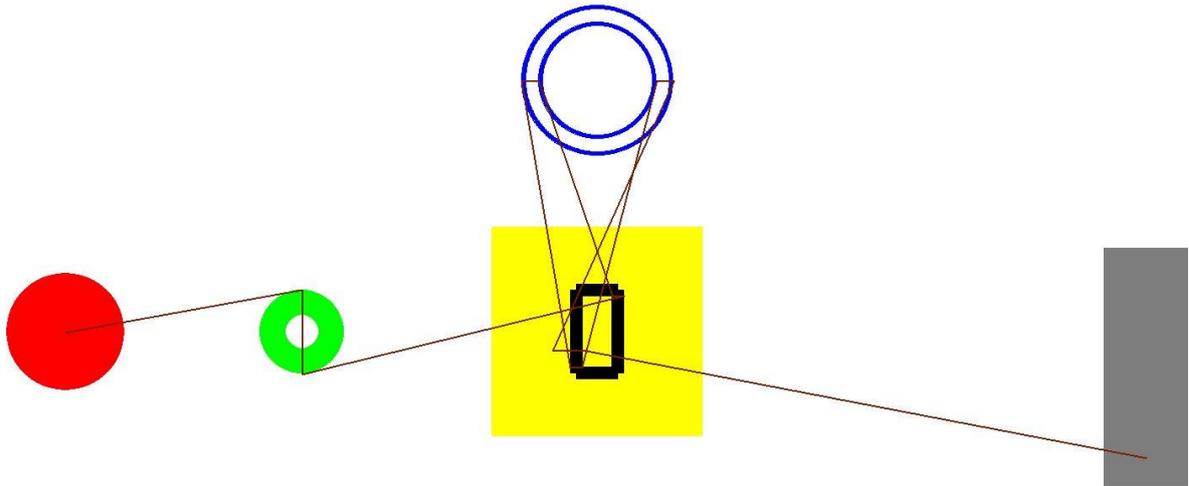
Estado S2 (n1p5n2p0p2p6p1bn0p0):



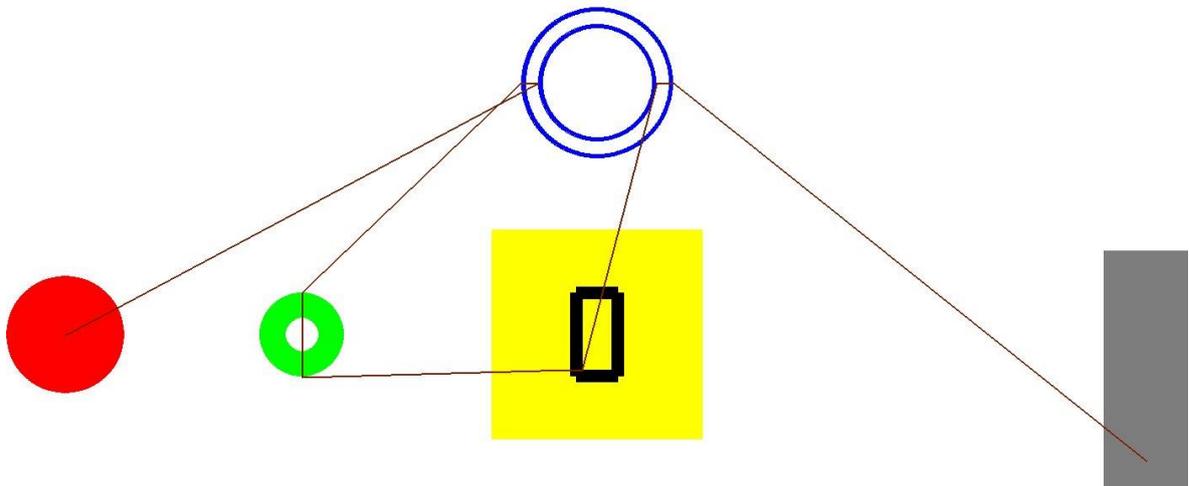
Estado S3 (n1p5p0n2p0p2n0p6p1bn0p0):



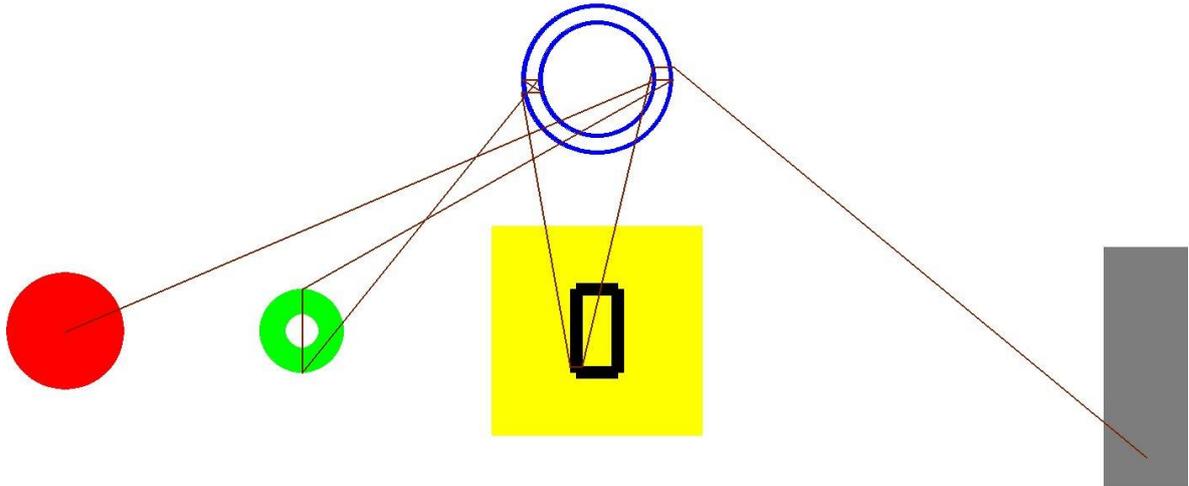
Estado S4 (n1p5n0n2p0p2p0p6p1bn0p0):



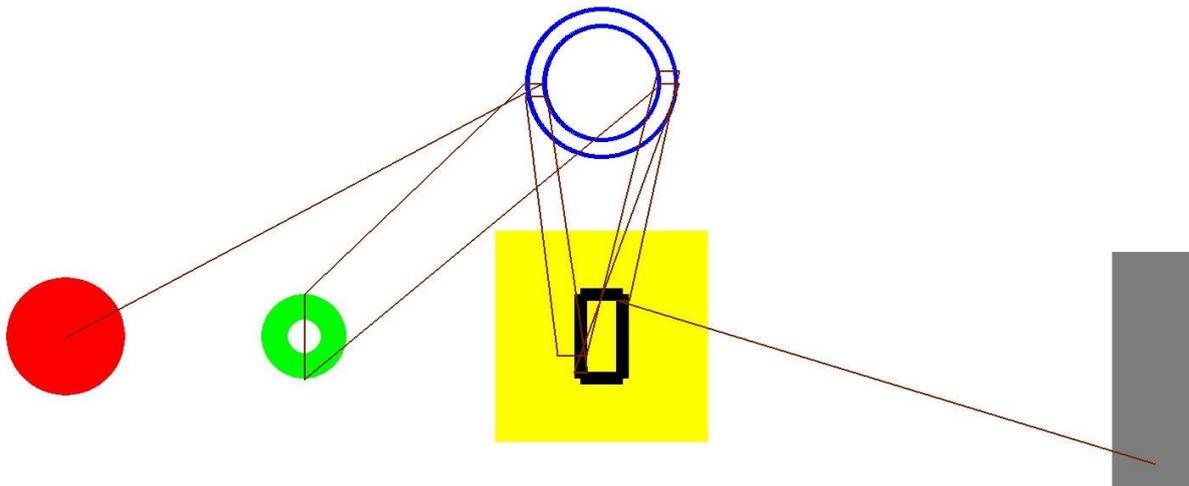
Estado S5 (n1n2p5p0p2p6p1bn0p0):



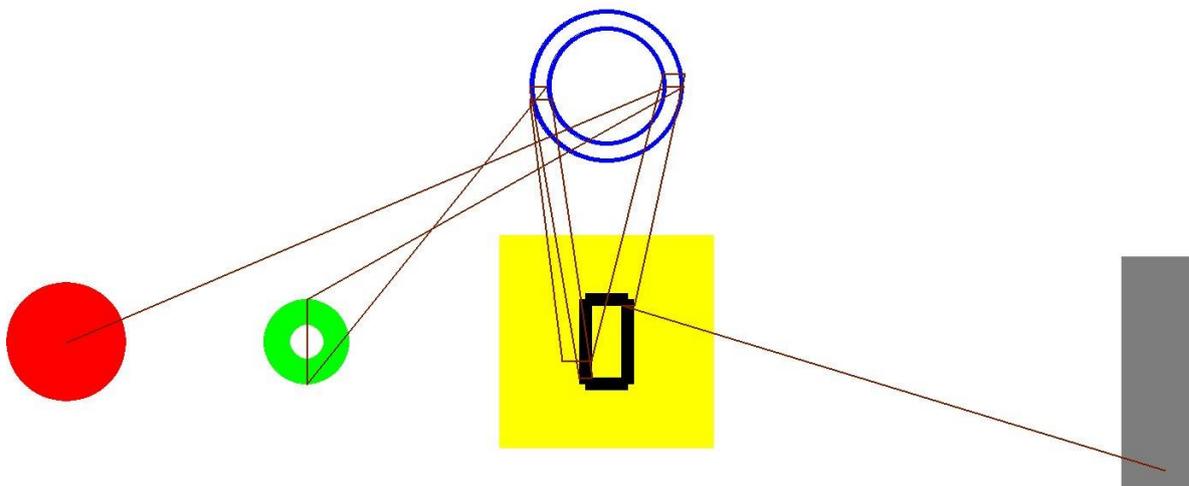
Estado S6 (n1p2p5n2n2p0p2p6p1bn0p0):



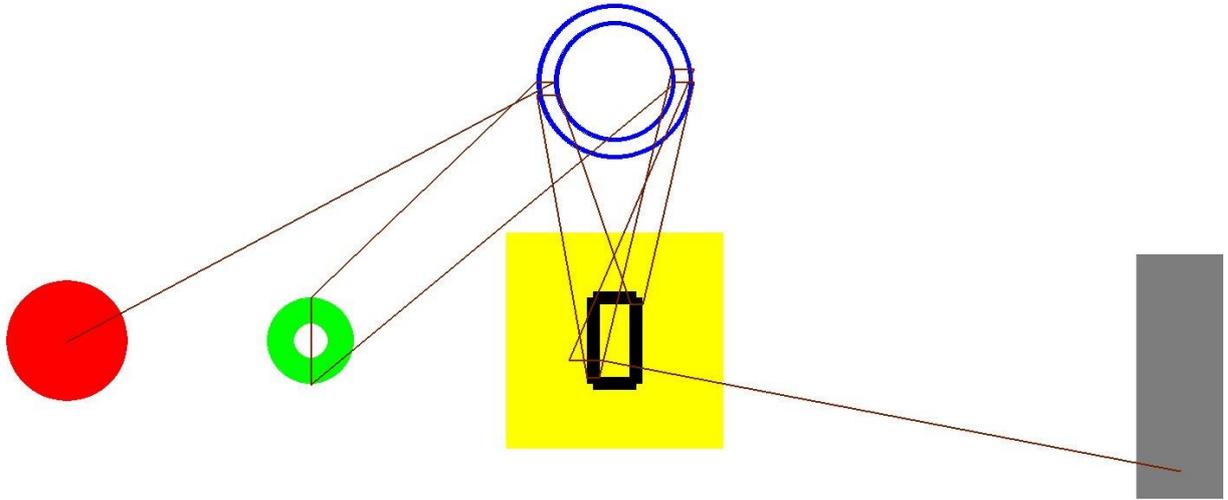
Estado S7 (n1n2p5p2p0n2p0p2n0p6p1bn0p0):



Estado S8 (n1p2p5n2p0n2p0p2n0p6p1bn0p0):



Estado S9 (n1n2p5p2n0n2p0p2p0p6p1bn0p0):



Estado S10 (n1p2p5n2n0n2p0p2p0p6p1bn0p0):

