UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VINICIUS TEIXEIRA DE MELO

APLICAÇÃO DE BUSCA BIDIRECIONAL EM PLANEJAMENTO COMO VERIFICAÇÃO SIMBÓLICA DE MODELOS

QUIXADÁ

2018

VINICIUS TEIXEIRA DE MELO

APLICAÇÃO DE BUSCA BIDIRECIONAL EM PLANEJAMENTO COMO VERIFICAÇÃO SIMBÓLICA DE MODELOS

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Orientadora: Profa. Dra. Maria Viviane de Menezes

Coorientador: Prof. Dr. Paulo de Tarso Guerra Oliveira

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M486a Melo, Vinicius Teixeira de.
Aplicação de Busca Bidirecional em Planejamento como Verificação Simbólica de Modelos / Vinicius Teixeira de Melo. – 2018.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2018.
Orientação: Profa. Dra. Maria Viviane de Menezes.
Coorientação: Prof. Dr. Paulo de Tarso Guerra Oliveira.

1. Planejamento Automatizado. 2. Modelo de Árvore de Decisão. 3. Métodos Formais (Computação). I. Título.

CDD 004

VINICIUS TEIXEIRA DE MELO

APLICAÇÃO DE BUSCA BIDIRECIONAL EM PLANEJAMENTO COMO VERIFICAÇÃO SIMBÓLICA DE MODELOS

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Aprovada em: ___/___/___.

BANCA EXAMINADORA

Profa. Dra. Maria Viviane de Menezes (Orientadora)
Universidade Federal do Ceará – UFC

Prof. Dr. Paulo de Tarso Guerra Oliveira (Coorientador)
Universidade Federal do Ceará - UFC

Prof. Dr. Davi Romero de Vasconcelos
Universidade Federal do Ceará - UFC

Prof. Dr. Criston Pereira de Souza
Universidade Federal do Ceará - UFC

À minha mãe, Valdelice Teixeira Maciel.

AGRADECIMENTOS

À minha família, em especial a minha mãe, Valdelice Teixeira Maciel, que sempre me deu apoio e fez o máximo para que eu pudesse concluir a minha graduação. Ao meu pai, Antônio Edinaldo de Melo, e o meu irmão, Vicente Almeida Maciel Neto, que sempre me deram apoio e me incentivaram para que pudesse ser a pessoa que tenho orgulho de ser hoje.

Aos meus amigos desde o ensino fundamental e médio, Beatriz, Jêsca, Yuri, Sávio, Segundo e Robert, que estão sempre a disposição para conversar, dar apoio e se divertir, vocês são demais. Aos amigos que conheci na graduação, Juliana <3, Pedro Henrique, Bora Jon, Enoque, Leomessi, Leonigga, Alex, Lucas, Fransa, Olímpico, Robson, Mamá, Sérgio, Rodrigol, Décio e Gabriel, com vocês essa graduação foi muito divertida e cheia de meninagens, muito obrigado por tudo, menineira.

À professora, orientadora e amiga, Maria Viviane de Menezes, por todo o conhecimento e experiências que pude adquirir com ela, sempre me apoiando para que eu pudesse obter ótimos resultados. Ao professor e amigo Paulo de Tarso Guerra Oliveira, por estar sempre a disposição para ajudar e conversar, e por ser uma pessoa fantástica. Sou muito fã de vocês, Viviane e Paulo.

Aos professores da Universidade Federal do Ceará, Campus de Quixadá, que contribuíram para a minha formação, em especial aos professores presentes nas bancas do meu trabalho de conclusão de curso 1 e 2, Wladimir Araujo Tavares, Criston Pereira de Souza e Davi Romero de Vasconcelos, muito obrigado pelo tempo, colaborações e sugestões.

"Se você quiser. Se você se esforçar. Se você treinar. Se você entrar de cabeça. Se você se concentrar. Nada garante que você vai conseguir."

(Craque Daniel)

RESUMO

Planejamento Automatizado é uma subárea da Inteligência Artificial que tem como objetivo a escolha de ações de um agente inteligente para alcançar uma meta. Uma solução para um problema de planejamento é uma sequência de ações (plano), que leva o agente de um estado inicial para um estado que satisfaz a meta. Quando não existe um plano que alcançe um estado meta, é dito que esse problema não possui solução. A busca por um plano pode ser feita de três maneiras: (i) progressiva, a partir do estado inicial, tentando alcançar algum estado que satisfaz a meta; (ii) regressiva, a partir do conjunto de estados que satisfazem a meta, tentando alcançar o estado inicial e; por fim, (iii) bidirecional, executando simultaneamente as buscas progressiva e regressiva e tentando alcançar um ponto em comum entre estas buscas. Os algoritmos raciocinam sobre modelos simbólicos e foram implementados utilizando Diagramas de Decisão Binária. Este trabalho propõe a implementação e incorporação da busca simbólica bidirecional no arcabouço de planejamento com verificação simbólica de modelos, e a realização de testes em domínios de planejamento *benchmarks* do *track clássico* e do *track unsolvability* (problemas sem solução).

Palavras-chave: Planejamento Automatizado. Diagrama de Decisão Binária. Busca Simbólica Bidirecional. Métodos Formais.

ABSTRACT

Automated Planning is a subarea of Artificial Intelligence that aims to choose actions from an intelligent agent to achieve a goal. One solution to a planning problem is a sequence of actions (plan), which takes the agent from an initial state to a state that satisfies the goal. When there is no plan that achieves a goal state, it is said that this problem has no solution. The search for a plan can be done in three ways: (i) progressive, from the initial state, trying to reach some state that meets the goal; (ii) regressive, from the set of states that satisfy the goal, trying to reach the initial state e; finally, (iii) bidirectional, simultaneously executing the progressive and regressive searches and trying to reach a common point between these searches. The algorithms reasoned about symbolic models and were implemented using Binary Decision Diagrams. This work proposes the implementation and incorporation of the bidirectional symbolic search in the planning framework with symbolic verification of models, and the performance of tests in planning domains of classic track and track unsolvability.

Keywords: Automated Planning. Binary Decision Diagram. Bidirectional Symbolic Search. Formal Methods.

LISTA DE FIGURAS

Figura 1 – Subgrafo do grafo de transição de estados para o domínio de Logística. 15
Figura 2 – Trecho de um domínio de Logística descrito em PDDL. 16
Figura 3 – Planejador. 17
Figura 4 – Tipos de representações possíveis de estados e domínios de planejamento. . . 22
Figura 5 – Trecho de um domínio de Logística instanciado com objetos (*grounded domain*). 24
Figura 6 – Verificador de Modelos. 24
Figura 7 – Exemplo de estrutura de Kripke e sua árvore de computação correspondente. 25
Figura 8 – Semântica dos operadores temporais da lógica CTL. 27
Figura 9 – Árvores de decisão binária. 31
Figura 10 – Compartilhamento de subgrafos isomorfos e remoção de testes redundantes. . 31
Figura 11 – Exemplo de um problema no domínio de Logística. 49
Figura 12 – Exemplo de um problema no domínio *sliding-tiles*. 50
Figura 13 – Exemplo de um problema no domínio *bottleneck*. 51
Figura 14 – Gráfico de comparação da profundidade de camadas alcançada pelas buscas no domínio de Logística. 54
Figura 15 – Gráfico de comparação do tempo das buscas no domínio *sliding-tiles*. 55
Figura 16 – Gráfico de comparação do tempo das buscas no domínio *bottleneck*. 56

LISTA DE TABELAS

Tabela 1 – Comparação entre os trabalhos relacionados e esta pesquisa. 44
Tabela 2 – Resultado da busca simbólica regressiva no domínio de Logística. 52
Tabela 3 – Resultado da busca simbólica progressiva no domínio de Logística. 52
Tabela 4 – Resultado da busca simbólica bidirecional no domínio de Logística. 53
Tabela 5 – Resultado das buscas simbólicas no domínio *sliding-tiles*. 53
Tabela 6 – Resultado das buscas simbólicas no domínio *bottleneck*. 54

LISTA DE ALGORITMOS

Algoritmo 1 – VERIFICADOR DE MODELOS 29
Algoritmo 2 – VERIFICADOR SIMBÓLICO DE MODELOS 33
Algoritmo 3 – VERIFICADOR PLANEX 37
Algoritmo 4 – VERIFICADOR SIMBÓLICO PLANEX 40
Algoritmo 5 – VERIFICADOR SIMBÓLICO PLANEX BIDIRECIONAL 46

LISTA DE ABREVIATURAS E SIGLAS

IPC	Competição Internacionial de Planejamento
UIPC	Competição Internacional de Planejamento para Problemas sem Solução
CTL	Lógica de Árvore Computacional
PDDL	Linguagem de Definição de Domínios de Planejamento

SUMÁRIO

1	**INTRODUÇÃO**	14
1.1	**Planejamento Clássico**	14
1.2	**Planejamento como Verificação de Modelos**	17
1.3	**Problemas de Planejamento sem Solução**	19
1.4	**Motivação**	20
1.5	**Objetivos**	20
1.5.1	***Objetivo Geral***	20
1.5.2	***Objetivos Específicos***	20
1.6	**Organização**	21
2	**FUNDAMENTAÇÃO TEÓRICA**	22
2.1	**Representação da Dinâmica do Ambiente**	22
2.2	**Verificação de Modelos**	24
2.2.1	***Estrutura de Kripke***	25
2.2.2	***Lógica CTL***	26
2.2.3	***Imagem***	28
2.2.4	***Pré-Imagem***	28
2.2.5	***Algoritmo***	28
2.2.6	***Verificação Simbólica de Modelos***	29
2.2.7	***Codificação proposicional para estados e transições***	30
2.2.8	***Diagrama de Decisão Binária***	30
2.2.9	***Cálculo da Imagem e Pré-Imagem Simbólica***	32
2.2.10	***Algoritmo Simbólico***	33
2.3	**Planejamento como Verificação de Modelos**	34
2.3.1	***Lógica α-CTL***	34
2.3.2	***Progressão de Ações***	35
2.3.3	***Regressão de Ações***	35
2.3.4	***Algoritmo***	36
2.4	**Planejamento como Verificação Simbólica de Modelos**	37
2.4.1	***Representação Simbólica das Ações***	37
2.4.2	***Progressão Simbólica de Ações***	38
2.4.3	***Regressão Simbólica de Ações***	39

2.4.4 *Algoritmo* 39
3 TRABALHOS RELACIONADOS 41
3.1 **PropPlan - Propositional Planning** 41
3.2 **MIPS - Model-Checking Integrated Planning System** 41
3.3 **MBP - Model Based Planning** 42
3.4 **VACTL - Verificador de Modelos α-CTL** 42
3.5 **VACTL-SYM - Symbolic Regression for Non-Deterministic Actions** . . . 42
3.6 **PACTL-SYM - Planejador Baseado em Verificação Simbólica de Modelos** 43
3.7 **CGAMER - Efficient symbolic search for cost-optimal planning** 43
3.8 **Comparação dos Trabalhos Relacionados** 43
4 BUSCA SIMBÓLICA BIDIRECIONAL 45
4.1 **Algoritmo** 46
5 PROCEDIMENTOS METODOLÓGICOS 47
5.1 **Implementação da Busca Simbólica Bidirecional no Arcabouço de Planejamento com Verificação Simbólica de Modelos** 47
5.2 **Seleção dos Domínios para Avaliação dos Algoritmos** 47
5.3 **Teste dos Algoritmos nos Domínios da Competição Internacional de Planejamento** 47
5.4 **Análise Comparativa da Busca Simbólica Bidirecional em Relação ao Desempenho da Busca Simbólica Unidirecional** 48
6 ANÁLISE EXPERIMENTAL 49
6.1 **Domínios Benchmarks** 49
6.1.1 *Domínio de Logística* 49
6.1.2 *Domínio sliding-tiles* 50
6.1.3 *Domínio bottleneck* 51
6.2 **Resultados** 52
6.3 **Análise Comparativa de Desempenho** 54
7 CONSIDERAÇÕES FINAIS 57
REFERÊNCIAS 58

1 INTRODUÇÃO

Em nosso dia a dia, estamos sempre executando ações e raciocinando sobre os efeitos destas ações. No entanto, é verdade que agimos muito mais do que *planejamos*. A necessidade de planejamento pode ser sentida na realização de tarefas corriqueiras tais como organizar uma festa, organizar uma viagem ou até decidir que disciplinas serão cursadas em um dado semestre. No entanto, em algumas situações, o planejamento pode ser fundamental. É o caso, por exemplo, de uma operação de resgate após um desastre natural. Esta operação envolve um grande número de pessoas, recursos de comunicação e infraestrutura de transporte e as decisões precisam ser tomadas em um curto espaço de tempo. Uma ferramenta de planejamento pode suportar este processo tratando das restrições ou mesmo oferecendo planos alternativos ainda não considerados.

Assim, *Planejamento* é um processo deliberativo de escolha e organização de ações que visa alcançar da melhor forma possível objetivos pré-estabelecidos. A habilidade de planejar tarefas é um aspecto fundamental do comportamento inteligente e sua automatização tem sido um dos principais objetos de pesquisa na área de *Inteligência Artificial* (RUSSELL; NORVIG, 2016).

Existem diversos aspectos que dificultam o processo de automatização do planejamento de tarefas (GHALLAB; NAU; TRAVERSO, 2004). Buscando simplificar esse processo, o *Planejamento Clássico* supõe que o ambiente evolui de forma determinística; ou seja: (i) que um agente executa ações que conduzem a evolução do ambiente, de modo que um determinado estado final desejado seja alcançado; (ii) que o estado corrente do ambiente muda apenas como consequência das ações executadas pelo agente e; (iii) que não há incerteza sobre os efeitos das ações do agente (PEREIRA; BARROS, 2007).

1.1 Planejamento Clássico

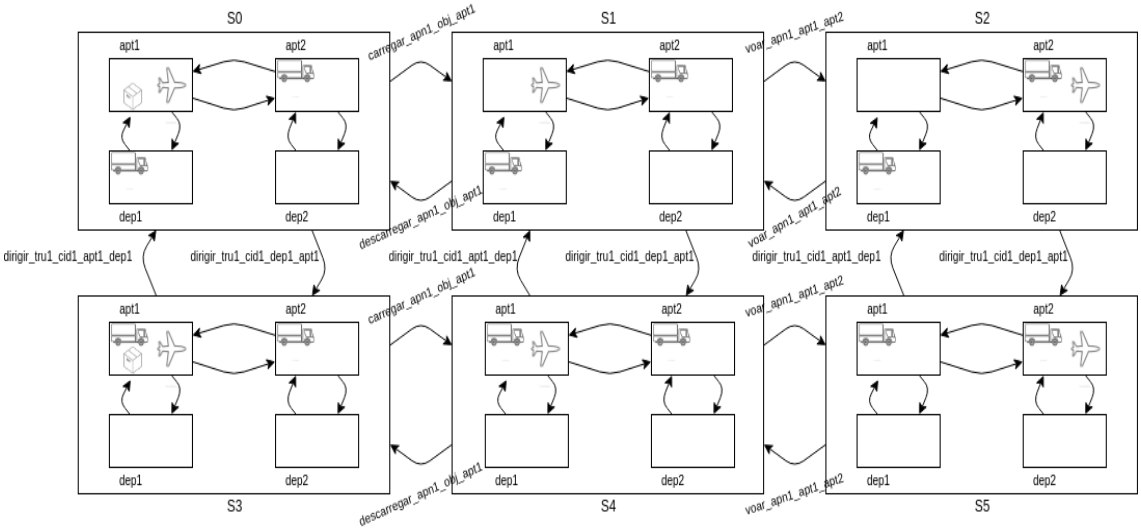
Planejamento é o processo de escolha de uma sequência de ações que quando são executadas em um dado estado inicial permite que um agente inteligente atinja suas metas. Em planejamento, a dinâmica do ambiente em que o agente se encontra é representada pelo *domínio de planejamento* que pode ser dado por um *grafo de transição de estados*, onde cada vértice é uma configuração possível do ambiente e cada aresta é rotulada pela ação que liga um estado a outro. Exemplos de domínios de planejamento: *mundo dos blocos* (NILSSON, 2014), domínio

de logística (BACCHUS, 2001), domínio do robô de marte (LONG; FOX, 2003), domínio das chaves (GÖBELBECKER et al., 2010), dentre outros.

O domínio de Logística, por exemplo, modela a tarefa de transportar um determinado número de pacotes de um local de origem para um local destino em uma região contendo *n* cidades, conectadas apenas por transporte aéreo. Cada cidade possui um conjunto de localização conectadas por rodovias usadas por caminhões. Para transportar um pacote dentro de uma mesma cidade, é necessário carregá-lo em um caminhão no local de origem, dirigir o caminhão até o local de destino e descarregar o pacote. Para transportar um pacote entre cidades, é necessário usar um avião (MENEZES, 2014).

A Figura 1 mostra um subgrafo de transições estados para o domínio de Logística com: 2 cidades (`cid1` com as localizações aeroporto `apt1` e depósito dep1) e; `cid2` com as localizações aeroporto `apt2` e depósito dep2) e; 2 caminhões (`tru1` e `tru2`), 1 avião (`apn1`) e 1 pacote (`obj`)). Cada nó representa um estado possível do ambiente e as arestas são rotuladas por ações, que quando executadas, modificam o estado do ambiente. No estado s_0 temos que o avião `apn1` e o pacote `obj` estão no aeroporto 1 da cidade `cit2`, o caminhão `tru1` está no depósito da `dep1` da cidade 1 e o caminhão `tru2` está no aeroporto `apt2` da cidade `cit2`. A execução da ação `carregar_apn1_obj_apt1` faz o pacote `obj` ser carregado para dentro do avião `apn1`.

Figura 1 – Subgrafo do grafo de transição de estados para o domínio de Logística.



Fonte – Elaborada pelo Autor.

No entanto, a representação de domínios de planejamento como *grafos de transição de estados* não é aplicada para sistemas de tamanho realistas, uma vez que é necessário expressar explicitamente uma grande quantidade de estados e transições. No lugar de uma representação

explícita, em planejamento adota-se uma *representação implícita* dada por meio de um *esquema de ações*.

A Figura 2 mostra um trecho da representação do domínio de logística em PDDL. Primeiramente, são definidos os predicados na lista `(:predicates)`. Após o nome de cada predicado, utiliza-se o símbolo `?` e o nome de uma variável que posteriormente será substituída por um objeto concreto em uma versão instanciada do domínio (*grounded version*). A versão com variáveis é denominada *lift version*. As ações são definidas em termos de seus parâmetros, suas pré-condições e seus efeitos. Os parâmetros definem que objetos estarão envolvidos na ação. As pré-condições indicam o que deve ser verdadeiro antes que a ação possa ser aplicada e nos efeitos é descrito como o estado atual do mundo se altera se a ação for aplicada.

Figura 2 – Trecho de um domínio de Logística descrito em PDDL.

```
(definicao (dominio logistica)
    (:predicados (pacote ?obj)
                 (caminhao ?tru)
                 (aviao ?apn)
                 (aeroporto ?apt)
                 (localizacao ?loc)
                 (cidade ?cid)
                 (dentro-cidade ?obj ?cid)
                 (em ?obj ?loc)
                 (dentro ?obj ?obj))
    (:action carregar-aviao
        :parameters (?obj ?apn ?loc)
        :precondition (and (pacote ?obj) (aviao ?apn)
                    (localizacao ?loc) (em ?apn ?loc) (em ?obj
?loc))
        :effect (and (not (em ?obj ?loc)) (dentro ?obj ?apn)))
    ...
```

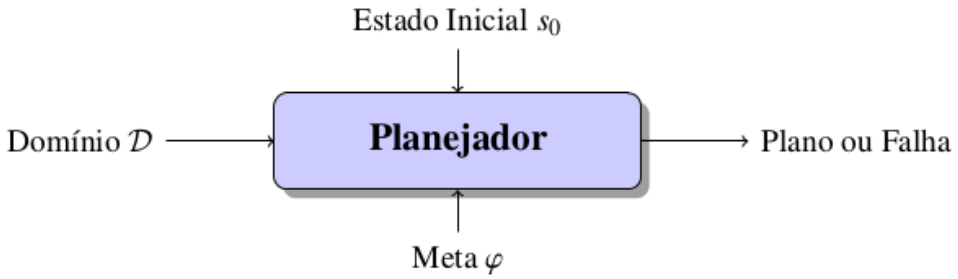
Fonte – Elaborada pelo Autor.

Para um mesmo domínio, é possível definir um conjunto de *problemas de planejamento*. Um *problema de planejamento* é composto por: (i) *domínio de planejamento*; (ii) *estado inicial*, a situação inicial do ambiente em que o agente encontra-se e; (iii) *meta de planejamento*, uma condição que deve ser satisfeita em algum estado final (GHALLAB; NAU; TRAVERSO, 2004).

A solução para um problema de planejamento é denominada *plano*, isto é, uma sequência de ações que quando executadas em um estado inicial s_0 leva o agente a algum estado satisfazendo a meta.

Um *planejador*, como ilustrado na Figura 3, é um algoritmo que recebe como entrada um problema de planejamento e devolve como saída um plano ou *falha*, se não for possível obter um plano.

Figura 3 – Planejador.



Fonte – Pereira e Barros (2007).

A *busca* por um plano pode ser feita: de forma *progressiva*, a partir do estado inicial, tentando alcançar algum estado que satisfaz a meta; de forma *regressiva*, a partir do conjunto de estados que satisfazem a meta, tentando alcançar o estado inicial; ou ainda, de forma *bidirecional*, executando simultaneamente as buscas progressiva e regressiva e tentando alcançar um ponto em comum entre estas buscas.

A complexidade do problema de decisão de verificar a existência do plano é PSPACE (GHALLAB; NAU; TRAVERSO, 2004).

1.2 Planejamento como Verificação de Modelos

Verificação de Modelos é uma abordagem formal utilizada para verificar se um sistema de transição de estados, representando a dinâmica de um sistema de hardware ou software, satisfaz certas propriedades especificadas como fórmulas lógicas. Com essa abordagem, no entanto, só é possível realizar a verificação de modelos com uma pequena quantidade de estados, uma vez que a representação de estados e transições é feita enumerando-se estado a estado, transição a transição.

Para permitir a verificação de sistemas de tamanho realista, utiliza-se a abordagem de *verificação simbólica de modelos*, em que o sistema de transição de estados é representado de forma *simbólica*. Nesta abordagem, os estados e transições não são tratados de forma individual (enumerativa), mas sim em conjuntos. Estruturas de dados, denominadas, *Diagramas de Decisão Binária* (BDDs) (BRYANT, 1986), são utilizadas para representar os conjuntos de estados e as

transições do modelo como fórmulas da lógica proposicional. Um estado é representado como um *e-lógico* de todas as proposições que são verdadeiras em um estado e conjuntos de estados são representados como um *ou-lógico* da representação individual de cada estado. As transições também são representadas como um ou-lógico das transições individuais. Desta forma, é possível representar sistemas de transição de estados com até 10^{20} estados (BURCH et al., 1992).

O arcabouço de verificação de modelos têm sido utilizado para obtenção de planos, uma área conhecida como *Planejamento baseada em Verificação de Modelos* (EDELKAMP; HELMERT, 2000; CIMATTI et al., 2003; PEREIRA; BARROS, 2007; SANTOS; BARROS, 2017; MENEZES; BARROS; PEREIRA, 2014). Tais abordagens utilizam a representação simbólica do sistema de transição de estados, representando o domínio de planejamento, e especificam as propriedades utilizando a lógica temporal CTL (Computation Tree Logic) (CLARKE; EMERSON, 1981).

Como limitações das abordagens de planejamento como verificação de modelos, destacamos:

- (i) o uso da representação explícita do domínio de planejamento por meio de um sistema de transição de estados.
- (ii) o uso da lógica temporal CTL, com a qual não é possível expressar as ações responsáveis pelas transições de estados;

O trabalho de Fourman (2000) difere dos demais trabalho da área de planejamento como verificação de modelos, pois é capaz de raciocinar sobre modelos implícitos, representados por meio de um esquema de ações. Neste trabalho as operações de progressão e regressão são independentes da relação de transição de estados, elas são definidas apenas sobre as pré-condições e efeitos das ações, representadas como BDDs. Este trabalho utiliza a busca regressiva para verificar a existência de um plano e, posteriormente, a busca progressiva para obter o plano solução. No entanto, menciona as possíveis vantagens de utilização da busca bidirecional na busca por um plano para um determinado problema. Esta abordagem limita-se também a solucionar problemas de planejamento clássico.

O trabalho de Torralba et al. (2017) utiliza as operações definidas em (FOURMAN, 2000) para encontrar um plano de forma eficiente em domínios com ações envolvendo custo. Utiliza também a busca bidirecional e uma adaptação da busca $A^\star$ para a busca simbólica. Com estas melhorias, o algoritmo proposto foi o segundo colocado na Competição Internacional de Planejamento (IPC) de 2014 (TORRALBA et al., 2017).

O trabalho de Menezes, Barros e Pereira (2014) estende as operações definidas por Fourman (2000) para domínios de planejamento com ações não-determinísticas. Esta abordagem também é baseada na lógica temporal α-CTL, com a qual é possível expressar as ações que causam as transições entre estados. A busca por uma solução é feita de forma regressiva, a partir de todos os estados que satisfazem a meta. Análises experimentais da busca simbólica regressiva indicam que há um grande consumo de memória para armazenar o BDD representando todos os estados que alcançam a meta. Para encontrar um plano em um domínio *benchmark* de Logística com 5 cidades, 15 pacotes, 2 caminhões e 2 aviões, a busca regressiva leva 11 horas, sendo grande parte deste gasto em operações de compactação do BDD (MENEZES, 2014).

1.3 Problemas de Planejamento sem Solução

Dizemos que um problema de *planejamento não possui solução* quando não é possível partir de um estado inicial para um estado que satisfaz a meta. Considere, por exemplo, o domínio de Logística da Figura 1 e o problema que possui s_0 como estado inicial e a meta é entregar o pacote no local $dep1$. Se, por exemplo, o caminhão estiver quebrado ou a estrada entre as localizações estiver interditada, temos um problema sem solução, uma vez que não é possível não há maneiras de transportar o pacote do local de origem $apt1$ para o local de destino $dep1$.

Problemas de planejamento sem solução tornaram-se um grande desafio para os planejadores estado-da-arte (MUISE C.; LIPOVETZKY, 2016), uma vez que a maioria dos algoritmos são baseados buscas heurísticas e estas nem sempre são capazes de distinguir a dificuldade de encontrar um plano da inexistência de plano. Assim, nos últimos anos, o estudo de algoritmos que verifiquem de forma eficiente se um problema de planejamento não possui solução têm ganhando mais atenção (GÖBELBECKER et al., 2010; SEIPP et al., 2016; TORRALBA, 2016; TORRALBA; HOFFMANN; KISSMANN, 2016; GNAD et al., 2016; POMMERENING; SEIPP, 2016; STEINMETZ; HOFFMANN, 2016; GNAD; STEINMETZ; HOFFMANN, 2016; BALYO; SUDA, 2016; HASLUM, 2016; KOROVIN; SUDA, 2016).

No ano de 2016, realizou-se pela primeira vez a *Competição Internacional de Planejamento para Problemas sem Solução* (UIPC). Esta competição teve como objetivo analisar algoritmos capazes de detectar se um problema de planejamento possui ou não solução. A UIPC-2016 contou com a participação de 14 algoritmos competidores. O vencedor da competição foi o algoritmo *Aidos* (SEIPP et al., 2016), um algoritmo baseado em busca heurística que detectou

262 dos 340 problemas corretamente como problemas sem solução. O segundo lugar foi o algoritmo Sympa (TORRALBA, 2016; TORRALBA et al., 2017) que, assim como este trabalho, é baseado em busca simbólica.

1.4 Motivação

Os indícios de melhoria da busca bidirecional já mencionados por Fourman (2000) e o excelente desempenho na UIPC obtido pelo algoritmo bidirecional de Torralba (2016), Torralba et al. (2017) motivaram:

- a proposta de incorporação deste tipo de busca no arcabouço de planejamento como verificação simbólica de modelos baseado na lógica α-CTL, o qual originalmente realiza uma busca regressiva no espaço de estados.
- a realização de testes de desempenho nos problemas de planejamento clássico e nos problemas de detecção da existência de plano da recente competição UIPC.

1.5 Objetivos

1.5.1 Objetivo Geral

O objetivo geral deste trabalho é a incorporação da busca simbólica bidirecional no arcabouço de planejamento com verificação simbólica de modelos proposto por Menezes (2014).

1.5.2 Objetivos Específicos

- Implementação da busca simbólica bidirecional;
- Teste do algoritmo de busca simbólica bidirecional em problemas do domínio de Logística da Competição Internacional de Planejamento, *track clássico*.
- Teste do algoritmo de busca simbólica bidirecional em problemas dos domínios da Competição Internacional de Planejamento para Problemas sem Solução, *track unsolvability*.
- Análise comparativa dos métodos de busca simbólica unidirecional (progressiva e regressiva) e bidirecional.

1.6 Organização

Este trabalho está organizado conforme segue: a Seção 2 apresenta os conceitos teóricos a respeito do funcionamento de um algoritmo de planejamento com verificação de modelos e verificação simbólica de modelos, de como representar de maneira eficiente os domínios de planejamento e de como funciona os métodos de busca simbólica regressiva e progressiva; a Seção 3 contém os trabalhos que possuem relação e que serviram de inspiração para o desenvolvimento desta pesquisa; a Seção 4 apresenta a principal contribuição deste trabalho, que é a definição e implementação da busca simbólica bidirecional no arcabouço de planejamento como verificação de modelos em α-CTL; a Seção 5 apresenta quais os passos executados para desenvolvimento deste trabalho; a Seção 6 contém a apresentação dos domínios utilizados para a análise experimental e quais os resultados obtidos nos testes e; por fim, a Seção 7 apresenta as considerações finais deste projeto de pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Representação da Dinâmica do Ambiente

Na Figura 4 apresenta quatro maneiras diferentes para descrever a dinâmica do ambiente em que o agente irá atuar, ou seja, o domínio de planejamento. A coluna da esquerda refere-se à *representação explícita* das transições entre estados (por meio do grafo de transições), enquanto a da direita é pertinente à *representação implícita* do domínio (por meio de linguagem de ações). A primeira linha refere-se à *representação enumerativa* do domínio (por meio da teoria de conjuntos), enquanto a segunda linha é pertinente à *representação simbólica* do domínio (representando conjunto de estados e ações por meio de fórmulas lógicas).

Figura 4 – Tipos de representações possíveis de estados e domínios de planejamento.

	Grafo de Transição de Estados	Linguagem de Ações
Estados Enumerados	Representação Explícita e Enumerativa	Representação Implícita e Enumerativa
Fórmulas Lógicas	Representação Explícita e Simbólica	Representação Implícita e Simbólica

Fonte – Elaborada pelo autor.

Representação Explícita e Enumerativa

Dado um conjunto de estados S de um ambiente e um conjunto de ações $\mathbb{A}$, representando as habilidades do agente, um domínio de planejamento pode ser representado explicitamente por meio de um sistema de transição de estados. Nessa representação, os estados são rotulados por um conjunto de proposições $\mathbb{P}$ (que representam as propriedades do ambiente) e as transições são rotuladas por elementos de $\mathbb{A}$. Tal sistema é denominado *sistema de transição de estados rotulado por ações*, sendo o par $(\mathbb{P}, \mathbb{A})$ sua assinatura (MENEZES, 2014).

Um domínio de planejamento representado por um *sistema de transição de estados rotulado por ações* é chamado de *enumerável* uma vez que existe uma bijeção entre um subconjunto dos números naturais e as possíveis valorações das proposições que rotulam os estados.

Representação Explícita e Simbólica

Nesta abordagem, conjuntos de estados são representados por meio de fórmulas lógicas proposicionais que são codificadas em estruturas de dados eficientes, denominadas *Diagramas de Decisão Binária* (BDDs). Essa abordagem é utilizada por algoritmos de planejamento baseados em verificação de modelos, como por exemplo o `MIPS` (EDELKAMP; HELMERT, 2001).

Nesta representação, é necessário dobrar o conjunto de proposições atômicas $\mathbb{P}$ para que seja possível representar os valores das proposições antes e depois de uma transição. A cópia do conjunto $\mathbb{P}$ é o conjunto $\mathbb{P}'$ (com variáveis linhas) tal que para cada $p \in \mathbb{P}$ há um $p' \in \mathbb{P}'$ correspondente. As proposições atômicas em $\mathbb{P}$ representam os valores das variáveis no estado que antecede a transição e as proposições atômicas em $\mathbb{P}'$ representam os valores das variáveis no estado que sucede a transição (Menezes (2014)).

Representação Implícita e Enumerativa

As duas abordagens citadas anteriormente representam, explicitamente, a função de transição de estados T. Outra forma de representação do domínio é a *representação implícita*, a qual é descrita apenas as mudanças que as ações causam nos estados em que são executadas. Esta é uma forma bastante compacta de representação de domínios de planejamento e usada na prática, uma vez que as representações explícitas sofrem com o problema da explosão de estados.

Na representação implícita, as ações são descritas em termos de suas pré-condições e efeitos. Uma linguagem amplamente utilizada para descrever ações é a linguagem `PDDL`. Em uma versão proposicional da linguagem `PDDDL`, o domínio de planejamento é definido por um conjunto de proposições atômicas $\mathbb{P}$ e um conjunto de ações $\mathbb{A}$ definida sobre estas proposições (MENEZES, 2014). A Figura 5 ilustra um trecho da descrição do domínio de logística proposicional, instanciado para os objetos `obj`, `apn1`, `tru1`, `tru2`, `apt1`, `apt2`, `dep1`, `dep2`, `cit1` e `cit2`,

Essa representação é chamada de enumerável porque os estados induzidos pelas ações são completamente rotulados pelas proposições, diferente da representação simbólica de conjunto de estados como fórmulas lógicas.

Representação Implícita e Simbólica

Na representação implícita e simbólica do domínio de planejamento, tando os estados como as ações são representados por fórmulas da lógica proposicional. Enquanto a representação explícita e simbólica descreve a função de transição T como fórmulas lógicas, na representação

Figura 5 – Trecho de um domínio de Logística instanciado com objetos (*grounded domain*).

```
(definition (domain logistica)
    (:predicates (pacote obj)
                 (caminhao tru1)
                 (caminhao tru2)
                 (aviao apn1)
                 (aeroporto apt1)
                 (aeroporto apt2)
                 ...
    (:action carregar-aviao
        :parameters (obj apn1 apt1)
        :precondicoes (and (pacote obj) (aviao apn1)
                    (localizacao ap1) (em apn1 apt1) (em obj apt1))
        :efeitos (and (not (em obj apt1)) (dentro obj apn1)))
    ...
```

Fonte – Elaborada pelo Autor.

implícita e simbólica, as pré-condições e efeitos das ações são representados por fórmulas lógicas (MENEZES, 2014).

A representação de uma ação $a \in \mathbb{A}$ por uma única fórmula lógica é dada pela conjunção das pré-condições e dos efeitos, com variáveis linha, que indicam os valores das proposições da ação no próximo estado.

Essa é a representação de domínio utilizada no algoritmo proposto por Menezes (2014) e que será utilizada neste trabalho.

2.2 Verificação de Modelos

O problema da verificação de modelos é determinado por verificar se um modelo $\mathbb{D} = \langle S, L, T \rangle$ satisfaz uma dada propriedade φ. O algoritmo de verificação recebe como entrada um modelo $\mathbb{D}$ e uma propriedade φ, e verifica a satisfatibilidade da propriedade φ nos estados do modelo $\mathbb{D}$. Se a propriedade é satisfeita no modelo $\mathbb{D}$, o algoritmo retorna *sucesso*. Caso contrário, um *contraexemplo* é retornado.

Figura 6 – Verificador de Modelos.

Fonte – Pereira e Barros (2007).

As descrições das propriedades a serem verificadas são feitas segundo uma lógica temporal, ou seja, lógicas modais cujas modalidades expressam aspectos temporais. A seguir é apresentado a estrutura utilizada para representar os modelos, denominada *Estrutura de Kripke*, e a lógica **CTL**, a qual é responsável por fundamentar formalmente essa verificação.

2.2.1 Estrutura de Kripke

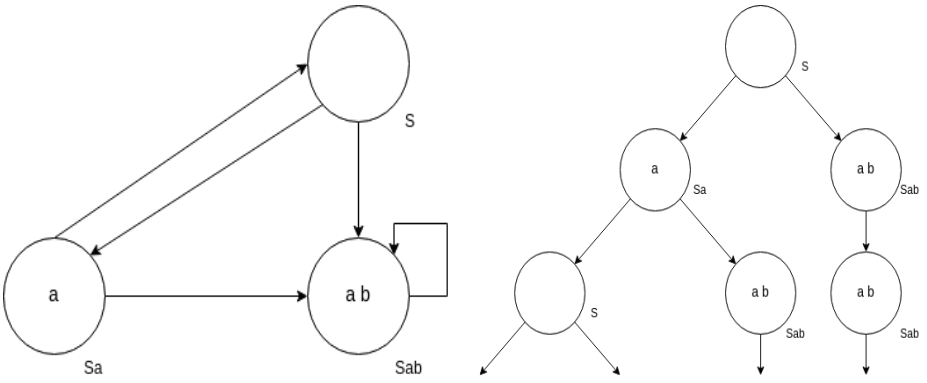
Uma estrutura de Kripke é formada por um conjunto finito de estados $S \neq \varnothing$, uma função de transição T entre os estados e uma função de rotulação de estados L (KRIPKE, 2007). Dado um conjunto de proposições $\mathbb{P} \neq \varnothing$, o modelo K do sistema é descrito por uma *estrutura de Kripke* sobre $\mathbb{P}$.

Formalmente, uma estrutura de Kripke sobre $\mathbb{P}$ é uma tupla $\mathbb{D} = \langle S, L, T \rangle$, onde:

- $S \neq \varnothing$ é um conjunto finito de estados;
- $L : S \to 2^P$ é uma função de interpretação de estados;
- $T : S \to 2^S$ é uma função de transição de estados;

Outra representação possível para uma estrutura de Kripke é a *árvore de computação*, uma árvore infinita, enraizada no estado inicial e representa todos os caminhos possíveis de computação do sistema modelado.

Figura 7 – Exemplo de estrutura de Kripke e sua árvore de computação correspondente.



(a) Estrutura de Kripke. (b) Árvore de Computação.

Fonte – Elaborada pelo autor.

2.2.2 Lógica CTL

Computation Tree Logic (CTL) é uma lógica temporal de tempo ramificado que permite especificar propriedades quantificadas sobre caminhos em uma árvore de computação (CLARKE; EMERSON, 1981). Esta lógica é dita de tempo ramificado pois considera que, para cada instante de tempo, pode haver vários futuros possíveis. Há quatro operadores temporais nesta lógica: $\circ$ (próximo), $\diamond$ (finalmente), $\Box$ (globalmente) e $\sqcup$ (até que). Todo operador temporal é precedido de um quantificador, existencial ($\exists$) ou universal ($\forall$).

Definição 1 *Sintaxe da lógica CTL:*

$$\varphi \doteq p \in \mathbb{P} | \neg\varphi_1 | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | \exists \circ \varphi_1 | \forall \circ \varphi_1 | \exists \Box \varphi_1 | \forall \Box \varphi_1 | \exists \diamond \varphi_1 | \forall \diamond \varphi_1 | \exists (\varphi_1 \sqcup \varphi_2) | \forall (\varphi_1 \sqcup \varphi_2)$$

Quando um estado particular s numa estrutura de Kripke D é designado como *estado inicial*, essa estrutura pode ser desdobrada numa árvore infinita, enraizada nesse estado, denominada árvore de computação e denotada por Υ^s_D. Também usamos Υ^s_D para denotar o conjunto de todos os caminhos que partem do estado s na estrutura de Kripke $\mathbb{D}$. Um *caminho* numa árvore de computação Υ^s_D é uma sequência infinita de estados $p = \langle s = p_0, p1, ... \rangle \in S^w$, onde S^w denota o conjunto de todas as sequências infinitas sobre S tal que $p_i \in T(p_{i-1})$, para $i > 0$ (PEREIRA; BARROS, 2007).

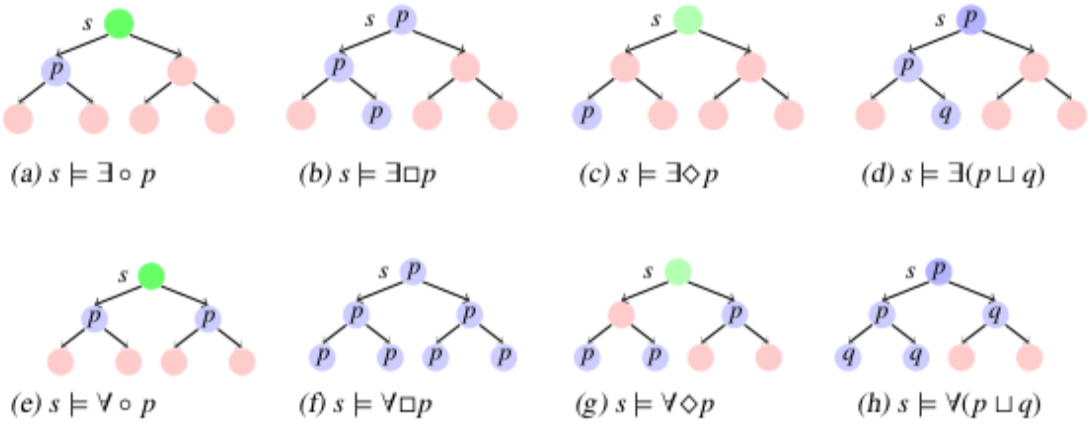
Intuitivamente, $\forall$ significa para todos os caminhos; $\exists$ significa que existe um caminho; $\circ$ refere-se ao próximo estado; $\Box$ refere-se a todos os estados globalmente; $\diamond$ refere-se a finalmente em um estado; e $\sqcup$ refere-se à obrigatoriedade da validade de uma fórmula até que outra seja válida (PEREIRA; BARROS, 2007).

Definição 2 *(Semântica da Lógica CTL) Sejam D uma estrutura de Kripke, s um estado dessa estrutura e φ uma fórmula CTL. A semântica das fórmulas CTL é definida como segue (PEREIRA; BARROS, 2007):*

- $s \vDash p$ se e somente se $p \in \mathbb{L}(s)$;
- $s \vDash \neg\varphi_1$ se e somente se $s \nvDash \varphi_1$;
- $s \vDash \varphi_1 \wedge \varphi_2$ se e somente se $s \vDash \varphi_1$ e $s \vDash \varphi_2$;
- $s \vDash \varphi_1 \vee \varphi_2$ se e somente se $s \vDash \varphi_1$ ou $s \vDash \varphi_2$;
- $s \vDash \exists \circ \varphi_1$ se e somente se, para algum caminho $p \in \Upsilon^S_{\mathbb{D}}$, $p_1 \vDash \varphi_1$;
- $s \vDash \forall \circ \varphi_1$ se e somente se, para todo caminho $p \in \Upsilon^S_{\mathbb{D}}$, $p_1 \vDash \varphi_1$;

- $s \vDash \exists \Box \varphi_1$ se e somente se, para algum caminho $p \in \Upsilon_{\mathbb{D}}^{S}$ e todo $i \geq 1$, $p_i \vDash \varphi_1$;
- $s \vDash \forall \Box \varphi_1$ se e somente se, para todo caminho $p \in \Upsilon_{\mathbb{D}}^{S}$ e todo $i \geq 1$, $p_i \vDash \varphi_1$;
- $s \vDash \exists \diamond \varphi_1$ se e somente se, para algum caminho $p \in \Upsilon_{\mathbb{D}}^{S}$, existe $i \geq 0$ tal que $p_i \vDash \varphi_1$;
- $s \vDash \forall \diamond \varphi_1$ se e somente se, para todo caminho $p \in \Upsilon_{\mathbb{D}}^{S}$, existe $i \geq 0$ tal que $p_i \vDash \varphi_1$;
- $s \vDash \exists(\varphi_1 \sqcup \varphi_2)$ se e somente se, para algum caminho $p \in \Upsilon_{\mathbb{D}}^{S}$, existe $j \geq 0$ tal que $p_j \vDash \varphi_2$ e, para todo $i < j$, $p_i \vDash \varphi_1$;
- $s \vDash \forall(\varphi_1 \sqcup \varphi_2)$ se e somente se, para todo caminho $p \in \Upsilon_{\mathbb{D}}^{S}$, existe $j \geq 0$ tal que $p_j \vDash \varphi_2$ e, para todo $i < j$, $p_i \vDash \varphi_1$.

Figura 8 – Semântica dos operadores temporais da lógica CTL.



Fonte – Pereira e Barros (2007).

Caracterização de ponto fixo da CTL

Um *ponto fixo* de uma função Γ é qualquer A tal que $\Gamma[A] = A$. O teorema a seguir garante que uma função monótona sempre tem pontos fixos mínimos e máximos (TARSKI, 1955).

Definição 3 *(Pontos fixos) Se* $\Gamma[Y]$ *é uma função monótona, então* $\Gamma[Y]$ *tem um ponto fixo mínimo denotado por* $\mu Y.\Gamma[Y]$ *e um ponto fixo máximo denotado por* $\nu Y.\Gamma[Y]$ *(TARSKI, 1955).*

Teorema 1 *Se S é finito, os operadores temporais globais de CTL podem ser caracterizados em termos de pontos fixos mínimos e máximos, conforme a seguir: (HUTH; RYAN, 2004)*

- $\exists \Box \varphi = \nu Y.(\varphi \wedge \exists \circ Y)$
- $\forall \Box \varphi = \nu Y.(\varphi \wedge \forall \circ Y)$
- $\exists \diamond \varphi = \mu Y.(\varphi \vee \exists \circ Y)$

- $\forall \diamond \varphi = \mu Y.(\varphi \vee \forall \circ Y)$
- $\exists(\varphi_1 \sqcup \varphi_2) = \mu Y.(\varphi_2 \vee (\varphi_1 \wedge \exists \circ Y))$
- $\forall(\varphi_1 \sqcup \varphi_2) = \mu Y.(\varphi_2 \vee (\varphi_1 \wedge \forall \circ Y))$

2.2.3 Imagem

A *imagem* de um conjunto de estados X computa o conjunto de estados que sucedem X. A seguir será mostrada a definição necessária para o cálculo da imagem:

Definição 4 *(Imagem de um conjunto de estados) Seja $D = \langle S, L, T \rangle$ um sistema de transição de estados com assinatura $(\mathbb{P})$ e seja $X \subseteq S$ um conjunto de estados. A imagem de X, denotada por $img(X)$, computa o conjunto de estados $Y \subseteq S$ tal que, para cada estado $s \in X$, há uma transição do estado s para um estado $s' \in Y$.*

$$img(X) = \{s \in S \mid \exists s' \in X \text{ e } T(s', s) \neq \varnothing\} \tag{2.1}$$

2.2.4 Pré-Imagem

A *pré-imagem* de um conjunto de estados X computa o conjunto de estados que antecedem X. A seguir será mostrada a definição necessária para o cálculo da pré-imagem:

Definição 5 *(Pré-imagem de um conjunto de estados) Seja $D = \langle S, L, T \rangle$ um sistema de transição de estados com assinatura $(\mathbb{P})$ e seja $X \subseteq S$ um conjunto de estados. A pré-imagem de X, denotada por $pre(X)$, computa o conjunto de estados $Y \subseteq S$ tal que, para cada estado $s \in Y$, há uma transição do estado s para um estado $s' \in X$.*

$$pre(X) = \{s' \in S \mid \exists s \in X \text{ e } T(s', s) \neq \varnothing\} \tag{2.2}$$

2.2.5 Algoritmo

Em sua versão mais geral, o verificador de modelos (HUTH; RYAN, 2004) é capaz de verificar qualquer fórmula temporal. No entanto, aqui iremos nos restringir ao algoritmo que verifica se uma fórmula temporal $\exists \diamond \varphi$ (existe um caminho que finalmente φ) é satisfeita.

A seguir, daremos o exemplo de um algoritmo para verificar se uma fórmula temporal é satisfeita no modelo formal.

O algoritmo VERIFICADOR DE MODELOS (Algoritmo 1) realiza busca regressiva a partir do conjunto de estados que satisfaz a fórmula φ, até que um ponto fixo seja

atingido. Ele recebe como entrada um modelo $\mathcal{M} =< S, L, T >$, um estado inicial s_0 e uma fórmula φ que representa a meta de planejamento. Inicialmente, X é um conjunto vazio e Y, o conjunto de todos os estados que satisfazem a fórmula φ por meio da execução da função $STF(\varphi)$. Em cada iteração do laço (linhas 4-7), o algoritmo salva em X o Y atual e, calcula o conjunto Y da próxima iteração, fazendo a união do Y atual com a pré-imagem de Y. Quando não for possível mais expandir o conjunto Y (um ponto fixo foi alcançado), o laço para e nas linhas 8-11 é verificado se o estado inicial s_0 pertence a Y. Se sim, o modelo satisfaz a fórmula a partir de s_0, caso contrário, o modelo não satisfaz a fórmula a partir de s_0.

Algoritmo 1: VERIFICADOR DE MODELOS

Entrada: $\mathcal{M} =< S, L, T >, s_0, \varphi$

```
1  início
2      X := {}
3      Y := STF(φ)
4      repita
5          X := Y
6          Y := Y ∪ pre(Y)
7      até X = Y;
8      se s0 ∈ Y então
9          retorna "φ é satisfeita a partir de s0."
10     senão
11         retorna "φ não é satisfeita a partir de s0."
```

Fonte – Menezes (2014).

2.2.6 Verificação Simbólica de Modelos

O algoritmo mostrado na seção anterior é ineficiente devido ao tamanho do modelo, não é possível armazenar todos os estados na memória, pois a representação do domínio é exponencial no número de proposições. Assim, a verificação simbólica consiste em uma abordagem mais eficiente para verificação de modelos, em que o modelo do sistema é representado simbolicamente por meio de Diagramas de Decisão Binária (BDDs), obtidos a partir da codificação proposicional de estados e transições.

2.2.7 Codificação proposicional para estados e transições

Seja $K = \langle S, L, T \rangle$ uma estrutura de Kripke sobre $\mathbb{P}$, a codificação proposicional para um estado $s \in S$, denotada por $\xi(s)$, é a fórmula

$$\xi(s) = \bigwedge_{p \in L(x)} p \wedge \bigwedge_{q \notin L(x)} \neg q; \tag{2.3}$$

e a codificação proposicional de um conjunto de estados $X \subseteq S$ é a fórmula

$$\xi(X) = \bigvee_{s \in X} \xi(s). \tag{2.4}$$

Para codificar uma transição, é necessário gerar uma cópia $\mathbb{P}'$ das proposições $\mathbb{P}$ e transformar cada cópia $p \in \mathbb{P}'$ em p'. As proposições do estado depois da transição são representadas pelas proposições pertencentes a $\mathbb{P}'$. A codificação proposicional para uma transição $t = (s, s') \in T$, denotada por $\xi(t)$, é dada pela fórmula

$$\xi(t) = \xi(s) \wedge \xi(s'); \tag{2.5}$$

e a codificação proposicional para uma relação de transição T é dada pela fórmula

$$\xi(T) = \bigvee_{t \in T} \xi(t). \tag{2.6}$$

2.2.8 Diagrama de Decisão Binária

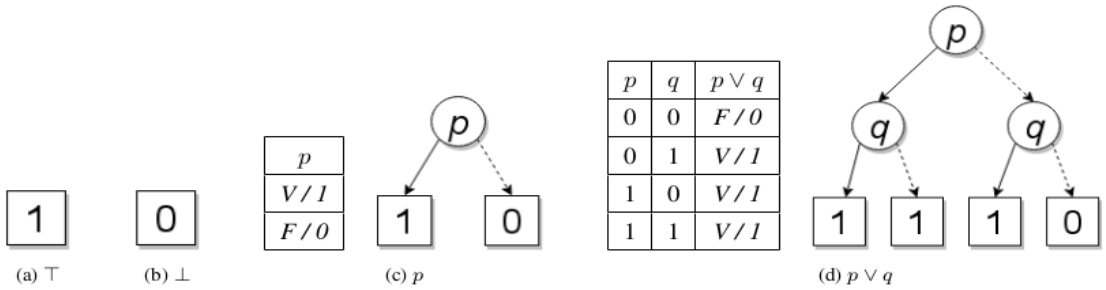
O *Diagrama de Decisão Binária* (Bryant (1986)) é uma estrutura de dados para representação de uma fórmula proposicional. Essa estrutura permite realizar operações entre fórmulas de maneira eficiente e consumindo menos recurso.

Graficamente, uma fórmula proposicional pode ser escrita por uma *Árvore de Decisão Binária* (uma forma mais simples do diagrama de decisão binária). Ela permite representar todas as valorações possíveis para os átomos proposicionais de uma fórmula, do mesmo modo que uma tabela verdade. Em uma árvore de decisão binária, os nós não-terminais são rotulados com átomos proposicionais e os nós terminais são rotulados com 1 ou 0, indicando verdadeiro e falso, respectivamente. Cada nó não-terminal tem duas arestas, uma linha sólida e uma tracejada,

representando, respectivamente, o átomo proposicional tendo a valoração verdade e o átomo proposicional tendo a valoração falsa.

Na Figura 6, são ilustrados alguns exemplos de árvore de decisão binária representando as fórmulas proposicionais $\top$, $\bot$, p e $p \vee q$. Note que cada caminho da raiz até um nó terminal representa uma possível valoração para os átomos proposicionais da fórmula, assim como uma possível valoração para a fórmula, de modo equivalente à tabela verdade.

Figura 9 – Árvores de decisão binária.

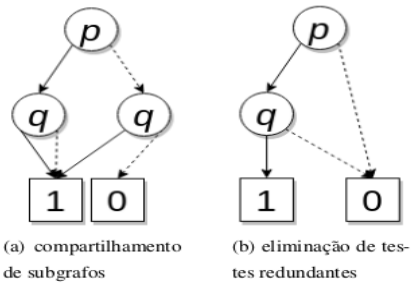


p	q	$p \vee q$
0	0	$F/0$
0	1	$V/1$
1	0	$V/1$
1	1	$V/1$

Fonte – Elaborada pelo autor.

Árvores de decisão binária, geralmente, possuem algumas redundâncias que podem ser exploradas. Parte dela pode ser eliminada por meio do compartilhamento de subgrafos isomorfos. Como 0 e 1 são os únicos nós terminais, pode-se otimizar movendo as arestas dos nós terminais para um único vértice com 0 e um único vértice com 1. Por exemplo, a árvore da Figura 10(a) pode ser otimizada dessa maneira e o resultado é mostrado na Figura 10(b). Quando esse compartilhamento de subgrafos isomorfos é realizado, a árvore é transformada em um grafo acíclico orientado, denominado *diagrama de decisão binária* (BRYANT, 1986).

Figura 10 – Compartilhamento de subgrafos isomorfos e remoção de testes redundantes.



Fonte – Elaborada pelo autor.

2.2.9 Cálculo da Imagem e Pré-Imagem Simbólica

Fórmulas Booleanas Quantificadas

Para a realizar o cálculo da imagem e pré-imagem através da manipulação das fórmulas lógicas, é necessário utilizar uma extensão da lógica proposicional que permita a quantificação sobre os valores das proposições, i.e., *Fórmulas Booleanas Quantificadas* (QBF) (BUNING; KARPINSKI; FLOGEL, 1995). Dada uma fórmula proposicional ϕ e uma proposição p que ocorre em ϕ, a *quantificação existencial* é definida como:

$$\exists p.\phi \equiv \phi[\top/p] \vee \phi[\bot/p]. \tag{2.7}$$

Para um conjunto de proposições $B = \{b_1, b_2, ..., b_n\}$, as quantificações existencial e universal são definidas, respectivamente, como:

$$\exists B.[\phi] \equiv \exists b_1.(...\exists b_n.\phi) \tag{2.8}$$

$$\forall B.[\phi] \equiv \forall b_1.(...\exists b_n.\phi) \tag{2.9}$$

Imagem Simbólica

A seguir é mostrado a definição necessária para realizar a operação de imagem simbólica de um conjunto de estados.

Definição 6 *(Imagem simbólica de um conjunto de estados) Sejam $D = \langle S, L, T \rangle$ um sistema de transição de estados com assinatura $(\mathbb{P})$; $\xi(X)$ a representação proposicional de $X \subseteq S$ e $\xi(T)$ a representação proposicional da relação de transição T. A imagem simbólica de X é dada por:*

$$simg(X) = \exists \mathbb{P}.(\xi(T) \wedge \xi(X)). \tag{2.10}$$

Pré-Imagem Simbólica

A seguir é mostrado a definição necessária para realizar a operação de pré-imagem simbólica de um conjunto de estados.

Definição 7 *(Pré-imagem simbólica de um conjunto de estados) Sejam $D = \langle S, L, T \rangle$ um sistema de transição de estados com assinatura $(\mathbb{P})$; $\xi(X')$ a representação proposicional de $X \subseteq S$*

utilizando "variáveis linha" e $\xi(T)$ a representação proposicional da relação de transição T. A pré-imagem simbólica de X é dada por:

$$spre(X) = \exists \mathbb{P}'.(\xi(T) \wedge \xi(X')). \tag{2.11}$$

2.2.10 Algoritmo Simbólico

O algoritmo VERIFICADOR SIMBÓLICO DE MODELOS (Algoritmo 2) realiza uma busca regressiva a partir do conjunto de estados que satisfazem a fórmula φ até que um ponto-fixo seja alcançado. Ele recebe como entrada a representação simbólica do modelo $\mathcal{M}$, dada pelo BDD $\mathcal{M}^{DD}$, o BDD representando o estado inicial, s_0^{DD}, e um BDD representando a meta , φ^{DD}. Inicialmente, X^{DD} é o BDD $\perp$ e Y^{DD}, o BDD que representa todos os estados que satisfazem a fórmula φ^{DD}. Em cada iteração do laço (linhas 4-7), o algoritmo salva em X^{DD} o Y^{DD} atual e, calcula o BDD Y^{DD} da próxima iteração, fazendo o "ou-lógico"do Y^{DD} atual com a pré-imagem simbólica de Y^{DD}. Quando não for possível mais expandir o BDD Y^{DD}, o laço para e nas linhas 8-11 é verificado se o estado inicial s_0^{DD} pertence a Y^{DD}. Se sim, o modelo satisfaz a fórmula φ^{DD}, caso contrário, o modelo não satisfaz a fórmula φ^{DD}.

Algoritmo 2: VERIFICADOR SIMBÓLICO DE MODELOS

Entrada: s_0^{DD}, φ^{DD}

```
1  início
2      X^DD := ⊥
3      Y^DD := STF(φ^DD)
4      repita
5          X^DD := Y^DD
6          Y^DD := Y^DD ∨ spre(Y^DD)
7      até X^DD = Y^DD;
8      se s_0^DD ∧ Y^DD ≠ ⊥^DD então
9          retorna "φ é satisfeita a partir de s_0."
10     senão
11         retorna "φ não é satisfeita a partir de s_0."
```

Fonte – Menezes (2014).

2.3 Planejamento como Verificação de Modelos

Verificadores de modelos podem ser utilizados para encontrar um plano de ações que atinja uma meta a partir do estado inicial. Esta área é conhecida como *planejamento como verificação de modelos*.

A maioria das abordagens de planejamento como verificação de modelos são baseadas na lógica CTL e na representação explícita do domínio de planejamento.

Neste trabalho, abordaremos planejamento como verificação de modelos capaz de raciocinar em modelos implícitos dado por um conjunto de proposições $\mathbb{P}$ e um conjunto de ações $\mathbb{A}$.

Assim, um domínio de planejamento $\mathbb{D}$ - com assinatura $\mathbb{P}$ e $\mathbb{A}$ - pode ser representado pelo conjunto de ações $\mathbb{A}$ em que cada $a \in \mathbb{A}$ é especificada pelo par $\langle$**precond(a)**,**efeitos(a)**$\rangle$, sendo **precond(a)**, um conjunto de proposições de $\mathbb{P}$ que devem ser verdadeiras no estado em que a ação **a** é executada; e **efeitos(a)** um par $\langle$**efeitos**$^+(a)$,**efeitos**$^-(a)\rangle$ de efeitos positivos e negativos da ação **a**, em que **efeitos**$^+(a)$ é o conjunto de proposições de $\mathbb{P}$ que se tornam verdadeiras após a execução da ação **a** e **efeitos**$^-(a)$ é o conjunto de proposições de $\mathbb{P}$ que se tornam falsas após **a** ser executada.

2.3.1 Lógica α-CTL

Em α-CTL, os símbolos usados para representar os operadores temporais, para diferenciar dos operados da lógica CTL, são assim "pontuados": $\odot$ (próximo), $\diamond$ (finalmente), $\boxdot$ (globalmente) e $\sqcup$ (até que). A sintaxe da lógica α-CTL é definida como:

$$\varphi \doteq p \in \mathbb{P} | \neg\varphi_1 | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | \exists \odot \varphi_1 | \forall \odot \varphi_1 | \exists \boxdot \varphi_1 | \forall \boxdot \varphi_1 | \exists \diamond \varphi_1 | \forall \diamond \varphi_1 | \exists(\varphi_1 \sqcup \varphi_2) | \forall(\varphi_1 \sqcup \varphi_2)$$

Seja $\mathbb{P} \neq \varnothing$ um conjunto de átomos proposicionais e $\mathbb{A}$ um conjunto finito de ações. A semântica de α-CTL é definida sobre um sistema de transição rotulado por ações $K = \langle S, L, T\rangle$ com assinatura $(\mathbb{P}, \mathbb{A})$, onde $S \neq \varnothing$ é um conjunto finito de estados, $L : S \to 2^{\mathbb{P}}$ é uma função de rotução de estados e $T : Sx\mathbb{A} \to 2^{\mathbb{P}}$ uma função de transição de estados em que cada transição é rotulada por uma ação.

O conjunto S contém todos os possíveis estados do ambiente de planejamento. A função L associa um conjunto de proposições $L(s) \in 2^{\mathbb{P}}$ a cada estado $s \in S$. A função T denota o conjunto de estados sucessores de s pela ação a, i.e., todos os estados que podem ser diretamente

alcançados pela execução da ação a no estado s. Se a ação a não é aplicável em s, tem-se que $T(s,a) = \varnothing$.

Intuitivamente, um estado s em K satisfaz uma fórmula $\forall \odot \varphi$ (ou $\exists \odot \varphi$) se existe uma ação $a \in \mathbb{A}$ que, quando executada em s, necessariamente (ou possivelmente) alcança um sucessor imediato de s que satisfaz a fórmula lógica φ. De forma similar, os outros operadores temporais são interpretados como os seus correspondentes na lógica CTL, porém considerando uma ação a.

2.3.2 Progressão de Ações

Para computar a *progressão* de um estado x por uma ação a, operação denotada por $progr^a(x)$, é necessário primeiramente verificar se a ação a é aplicável no estado x, i.e., se $precond(a) \subseteq x$. Nesse caso, o estado sucessor alcançado pela execução da ação a em x é obtido pela remoção dos efeitos negativos de a no estado x e pela subsequente adição dos efeitos positivos de a em x (MENEZES, 2014). Formalmente, a progressão de um estado x por uma ação a é dada por:

$$progr^a(x) = \begin{cases} (x \backslash efeitos^-(a)) \cup efeitos^+(a) & se \quad precond(a) \subseteq x \\ \varnothing & c.c. \end{cases} \tag{2.12}$$

A progressão de um conjunto de estados X pode ser definida para um conjunto de ações $\mathbb{A}$. Assim, a progressão para toda ação $a \in \mathbb{A}$ é dada por:

$$progr(X) = \bigcup_{a \in \mathbb{A}} progr^a(X). \tag{2.13}$$

2.3.3 Regressão de Ações

A *regressão* de um estado x, por uma dada ação a, produz um conjunto de estados predecessores, operação denotada por $regr^a(x)$. Um conceito muito importante na regressão é a relevância de uma ação para um estado. Para que uma ação a seja *relevante* em um estado x, ela deve contribuir com as propriedades satisfeitas em x, i.e., no mínimo um dos efeitos positivos de a deve pertencer ao estado x e nenhum dos efeitos positivos de a pertence ao estado x.

Formalmente, dado um estado x, uma ação é relevante para x se $efeitos^+(a) \cap x \neq \varnothing$ e $efeitos^-(a) \cap x = \varnothing$. O conjunto de estados predecessores de x pela ação a é obtido pela

remoção dos efeitos positivos de a em x e pela subsequente adição das precondições de a, i.e., $(x \backslash efeitos^{+}(a)) \cup precond(a)$ (GHALLAB; NAU; TRAVERSO, 2004). Caso a condição de relevância não seja satisfeita, não existe estado predecessor. Assim, a regressão de um estado x por uma ação a é dada por:

$$regr^{a}(x) = \begin{cases} (x \backslash efeitos^{+}(a)) \cup precond(a) & se \quad efeitos^{+}(a) \cap x \neq \varnothing \text{ e } efeitos^{-}(a) \cap x = \varnothing \\ \varnothing & c.c. \end{cases} \tag{2.14}$$

A regressão de um conjunto de estados X também pode ser definida para um conjunto de ações $\mathbb{A}$. Assim, a regressão para toda ação $a \in \mathbb{A}$ é dada por:

$$regr(X) = \bigcup_{a \in \mathbb{A}} regr^{a}(X). \tag{2.15}$$

2.3.4 Algoritmo

O algoritmo VERIFICADOR PLANEX (Algoritmo 3) verifica a existência de um plano solução em um domínio de planejamento dado por um conjunto de ações $\mathbb{A}$. Ele recebe como entrada um conjunto de ações $\mathbb{A}$, um estado inicial s_0 e uma fórmula φ representando a meta de planejamento. Inicialmente, X é um conjunto vazio e Y, o conjunto de todos os estados que satisfazem a fórmula φ. Em cada iteração (linhas 4-9), o algoritmo salva em X o Y atual e, calcula o conjunto Y da próxima iteração, fazendo a união do Y atual com a regressão de Y. Note que o cálculo da pré-imagem (representação explícita com transições) é substituído pelo cálculo da regressão (representação implícita com ações). Em cada iteração é verificado se o estado inicial s_0 pertence a Y. Se sim, o problema é solucionável. Se não for mais possível expandir o conjunto Y e o estado inicial s_0 não pertencer ao conjunto Y, então o problema não possui solução.

Algoritmo 3: VERIFICADOR PLANEX

Entrada: $\mathbb{A}, s_0, \varphi$

```
1  início
2      X := {}
3      Y := STF(φ)
4      repita
5          X := Y
6          Y := Y ∪ regr(Y)
7          se s0 ∈ Y então
8              retorna "Problema solucionável."
9      até X = Y;
10     retorna "Problema não solucionável."
```

Fonte – Menezes (2014).

2.4 Planejamento como Verificação Simbólica de Modelos

Em um domínio representado explicitamente, em que são descritas todas as transições entre estados, a busca por uma solução pode ser feita através de operações entre conjuntos, aplicando a verificação de modelos, como também pode ser realizada por meio de operações lógicas, onde é utilizada a verificação simbólica de modelos.

Porém, em domínios representados implicitamente, a busca por uma solução através de progressão e regressão de estados são realizadas de maneira simbólica, codificando estados e ações em fórmulas da lógica proposicional, que podem ser manipuladas eficientemente com diagramas de decisão binária.

O cálculo proposicional para computar estados predecessores em modelos implícitos, baseado na verificação simbólica de modelos, é denominada *regressão simbólica de ações* e *progressão simbólica de ações*. Inicialmente, é realizada a representação simbólica das ações, para então ser feito o cálculo da progressão e regressão de forma simbólica.

2.4.1 Representação Simbólica das Ações

A codificação para uma ação $a = \langle precond(a); efeitos(a) \rangle$ é denotada pelo par $a = \langle \xi(precond(a)); \xi(efeitos(a)) \rangle$. Cada efeito $e \in efeitos(a)$ é dado pela conjunção dos

efeitos positivos e negativos de a, as precondições de uma ação a é dada pela conjunção das proposições de $precond(a)$, e o conjunto de todos os efeitos de uma ação a é definido pela disjunção de cada efeito de a. A representação dos efeitos de uma ação $a \in \mathbb{A}$, denotada por $\xi(e)$, é dada por:

$$\xi(e) = \bigwedge_{d \in efeitos^{+}(a)} d \wedge \bigwedge_{r \in efeitos^{-}(a)} \neg r; \tag{2.16}$$

a codificação para as precondições de uma ação a, $\xi(precond(a))$, é dada por:

$$\xi(precond(a)) = \bigwedge_{p \in precond(a)} p; \tag{2.17}$$

e, for fim, o conjunto de todos os efeitos de uma ação a, denominado $efeitos(a)$, é dado por:

$$\xi(efeitos(a)) = \bigvee_{e \in efeitos(a)} \xi(e). \tag{2.18}$$

2.4.2 Progressão Simbólica de Ações

A *progressão simbólica* computa o conjunto de estados sucessores que levam, possivelmente, a estados pertencentes a X, ou seja, alguns efeitos de uma ação $a \in \mathbb{A}$ resultam em estados pertencentes a X, enquanto outros efeitos da mesma ação levam a estados que não pertencem a X.

Dada uma ação $a = \langle \xi(precond(a)); \xi(efeitos(a)) \rangle$ e a representação proposicional $\xi(X)$ de um conjunto de estados, a progressão simbólica de X segundo a ação a, denominada $sprogr^{a}(X)$ é dada por:

$$sprogr^{a}(X) = \exists modifica(a).(\xi(X) \wedge \xi(precond(a))) \wedge \xi(efeitos(a)). \tag{2.19}$$

A progressão simbólica de um conjunto de estados X, representado proposicionalmente por $\xi(X)$, também pode ser definida para um conjunto de ações $\mathbb{A}$. Respectivamente, a progressão simbólica para toda ação $a \in \mathbb{A}$ é dada por:

$$sprogr(X) = \bigvee_{a \in \mathbb{A}} sprogr^{a}(\xi(X)); \tag{2.20}$$

2.4.3 Regressão Simbólica de Ações

A *regressão simbólica* computa o conjunto de estados predecessores que levam, possivelmente, a estados pertencentes a X, ou seja, alguns efeitos de uma ação $a \in \mathbb{A}$ resultam em estados pertencentes a X, enquanto outros efeitos da mesma ação levam a estados que não pertencem a X.

Dada uma ação $a = \langle \xi(precond(a)); \xi(efeitos(a)) \rangle$ e a representação proposicional $\xi(X)$ de um conjunto de estados, a regressão simbólica de X segundo a ação a, denominada $sregr^a(X)$ (MENEZES, 2014), é dada por:

$$sregr^a(X) = \xi(precond(a)) \wedge \exists modifica(efeitos(a)).(\xi(efeitos(a)) \wedge \xi(X)). \quad (2.21)$$

Analisando a equação acima, a conjunção $\xi(efeitos(a)) \wedge \xi(X)$ elege o subconjunto de X alcançados pelos efeitos a. Se algum efeito de a é relevante para algum estado $x \in X$, então $\xi(efeitos(a)) \wedge \xi(X) \neq \perp$. Em seguida, a quantificação existencial usando o conjunto $modifica(efeitos(a))$ elimina uma a uma as proposições que ocorrem nos efeitos de a da fórmula restante. Finalmente, é feita a conjunção da fórmula restante com a fórmula representando a precondição da ação a, obtendo a fórmula que corresponde aos estados predecessores.

A regressão simbólica de um conjunto de estados X, representado proposicionalmente por $\xi(X)$, também pode ser definida para um conjunto de ações $\mathbb{A}$. Respectivamente, a regressão simbólica para toda ação $a \in \mathbb{A}$ é dada por:

$$sregr(X) = \bigvee_{a \in \mathbb{A}} sregr^a(\xi(X)); \quad (2.22)$$

2.4.4 Algoritmo

O algoritmo VERIFICADOR SIMBÓLICO PLANEX (Algoritmo 4) realiza uma busca regressiva a partir do BDD que representa os estados que satisfazem a fórmula φ^{DD} até que não seja mais possível realizar essa busca. Ele recebe como entrada as ações $\mathbb{A}^{DD}$, um estado inicial s_0^{DD} e uma fórmula φ^{DD} que representa a meta de planejamento, todos representados por BDD's. Inicialmente, X^{DD} é um BDD $\perp$ e Y^{DD}, o BDD que representa todos os estados que satisfazem a fórmula φ. Em cada iteração (linhas 4-7), o algoritmo guarda em X^{DD} o Y^{DD} atual e, calcula o BDD Y^{DD} da próxima iteração, fazendo o "ou-lógico"do Y^{DD} atual com a regressão

simbólica de Y^{DD}. Em cada iteração, é verificado se o estado inicial s_0^{DD} pertence a Y^{DD}. Se sim, o problema é solucionável. Se não for mais possível expandir o BDD Y^{DD} e o estado inicial s_0^{DD} não pertencer ao conjunto Y^{DD}, então o problema não possui solução.

Algoritmo 4: VERIFICADOR SIMBÓLICO PLANEX

Entrada: $\mathbb{A}^{DD}, s_0^{DD}, \varphi^{DD}$

1 **início**
2 $X^{DD} := \bot$
3 $Y^{DD} := STF(\varphi^{DD})$
4 **repita**
5 $X^{DD} := Y^{DD}$
6 $Y^{DD} := Y^{DD} \vee sregr(Y^{DD})$
7 **se** $Y^{DD} \wedge s_0^{DD} \neq \bot^{DD}$ **então**
8 **retorna** *"Problema solucionável."*
9 **até** $X^{DD} = Y^{DD}$;
10 **retorna** *"Problema não solucionável."*

Fonte – Menezes (2014).

3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados os principais trabalhos que contribuíram para o desenvolvimento desta pesquisa. A seguir estão organizados esses trabalhos em ordem cronológica e a subseção final contém uma tabela de comparação entre os trabalhos relacionados e esta pesquisa.

3.1 PropPlan - Propositional Planning

Fourman (2000) apresenta um algoritmo de planejamento denominado `PropPlan`, inspirado no `GraphPlan` (BLUM; FURST, 1997). O `PropPlan`, ao invés de explorar o espaço de estados concretamente, gerando e visitando estados individuais, ele explora o espaço de estados de maneira abstrata, através da computação de conjunto de estados acessíveis. O algoritmo utiliza uma estrutura de dados em camadas, assim, o conjunto Y_i são os estados que podem ser alcançados em i passos.

O algoritmo `PropPlan` utiliza os diagramas de decisão binária como representação de seus conjuntos de estados. Assim, Fourman (2000) consegue tornar a representação dos conjuntos de estados tratáveis para uma variedade de exemplos padrão.

3.2 MIPS - Model-Checking Integrated Planning System

Edelkamp e Helmert (2000) apresentam o primeiro sistema conhecido de planejamento baseado em métodos de verificação de modelos. O planejador `MIPS`, utiliza a estrutura de dados *diagrama de decisão binária* (BDDs) para representar, de forma compacta e eficiente, o conjunto de estados e transições de um domínio de planejamento.

O sistema executa uma busca em largura no espaço de estados, a primeira solução encontrada consistirá de um número mínimo de estados. Porém, como o algoritmo não era eficiente, Edelkamp e Helmert (2000) realizaram a implementação da busca bidirecional do espaço de estados ao invés da busca em profundidade, e utilizaram três métricas de decisão do tipo de busca a ser realizado: (1) o tamanho do BDD, expandiam o menor BDD; (2) o número de estados percorridos e, por último; (3) o tempo gasto na última busca realizada. Nos experimentos realizados com o domínio de logística, a métrica (3) foi analisada como a mais eficaz.

3.3 MBP - Model Based Planning

Cimatti et al. (2003) abordam o planejamento em domínios não-determinísticos, discutindo questões conceituais e práticas. Cimatti et al. (2003) caracterizaram, formalmente, os problemas de planejamento em três tipos diferentes: (i) *planejamento fraco*, no qual as soluções têm uma chance de sucesso; (ii) *planejamento forte*, no qual as soluções garantem atingir o objetivo e, por fim; (iii) *planejamento forte cíclico*, no qual o agente alcança o objetivo com estratégias de tentativa e erro iterativas.

Cimatti et al. (2003) apresentam o planejador *Model Based Planner* (`MBP`), o qual utiliza técnicas de planejamento de verificação simbólica de modelos e diagramas de decisão binária, que permitem uma representação e manipulação eficaz de fórmulas proposicionais. Como fundamento lógico, o `MBP` utiliza a lógica temporal CTL para seus cálculos de verificação de modelos.

3.4 VACTL - Verificador de Modelos α-CTL

Pereira e Barros (2007) propõe uma nova classe de metas de planejamento, denominada metas de alcançabilidade estendidas e mostra que, para essa classe de metas, a semântica da lógica CTL não é adequada para formalizar os algoritmos de validação de planos. As metas de alcançabilidade estendidas são caracterizadas por estabelecerem uma condição a ser preservada, ou evitada, em todos os estados visitados durante a execução do plano, diferente das metas de alcançabilidade simples que determinam que uma condição deve ser satisfeita, necessariamente, no estado final do plano.

Como forma de contornar a limitação da lógica CTL diante das metas de alcançabilidade estendidas, Pereira e Barros (2007) propôs uma nova versão de CTL, denominada α-CTL, que foi utilizada para o desenvolvimento de um arcabouço de planejamento para metas de alcançabilidade estendidas. Com a lógica α-CTL foi possível descrever o conjunto $\mathbb{A}$, que representa as ações aplicáveis no domínio de planejamento.

3.5 VACTL-SYM - Symbolic Regression for Non-Deterministic Actions

Menezes, Barros e Pereira (2014) propõe um arcabouço para sugerir mudanças em problemas de planejamento em que não é possível encontrar um plano, baseado na abordagem

formal de verificação de modelos e utilização da lógica α-CTL (PEREIRA; BARROS, 2007). Menezes, Barros e Pereira (2014) propõe também algoritmos simbólicos de busca progressiva e regressiva para verificação da existência de um plano em um problema de planejamento.

Em Menezes, Barros e Pereira (2014), é utilizado os BDDs para a representação dos estados e transições dos seguintes domínios de planejamento: domínio de logística, domínio do robô de marte e domínio das chaves.

3.6 PACTL-SYM - Planejador Baseado em Verificação Simbólica de Modelos

Santos e Barros (2017) apresentam uma versão simbólica para o planejador `PACTL`, no qual é utilizado técnicas de verificação simbólica de modelos e diagramas de decisão binária. A fundamentação lógica para o planejador `PACTL` é baseada na lógica α-CTL (PEREIRA; BARROS, 2007), assim, é possível que o planejador raciocine formalmente sobre as ações.

O domínio de problemas utilizado por Santos e Barros (2017) é denominado *Fully-Observable Non-Deterministic Problem* (FOND), o qual considera ações com efeitos não-determinísticos e a geração de sequência de ações que, eventualmente, terminam em becos-sem-saídas.

3.7 CGAMER - Efficient symbolic search for cost-optimal planning

Torralba et al. (2017) mostram que a representação simbólica dos estados e transições tornam os algoritmos de planejamento mais eficientes em termos de tempo e memória. No entanto, o desenvolvimento de heurísticas para a busca de estados sucessores deixam as técnicas de busca simbólica em segundo lugar.

Torralba et al. (2017) propõe duas otimizações para planejamento simbólico. Por um lado, são analisados e comparados diferentes métodos de cálculo de imagem para executar de forma eficiente a busca progressiva na busca simbólica. Por outro lado, utilizam restrições invariantes no estado para podar estados na busca simbólica.

3.8 Comparação dos Trabalhos Relacionados

A Tabela 1 mostra a comparação entre os trabalhos relacionados e esta pesquisa. As principais características consideradas são: o ano de desenvolvimento do trabalho, o tipo de representação para o domínio, qual a busca utilizada no planejador, a fundamentação lógica do

algoritmo, se ele é determinístico ou não determinístico e, por fim, se é simbólico ou não.

Tabela 1 – Comparação entre os trabalhos relacionados e esta pesquisa.

Trabalho	Ano	Representação	Busca	Lógica	Simbólico
PropPlan (FOURMAN, 2000)	2000	Implícita	Bidirecional	-	Sim
MIPS (EDELKAMP; HELMERT, 2000)	2000	Explícita	Bidirecional	CTL	Sim
MBP (CIMATTI et al., 2003)	2003	Explícita	Regressiva	CTL	Sim
VACTL (PEREIRA; BARROS, 2007)	2007	Explícita	Regressiva	α-CTL	Não
PACTL-SYM (SANTOS; BARROS, 2017)	2014	Implícita	Regressiva	α-CTL	Sim
VACTL-SYM (MENEZES; BARROS; PEREIRA, 2014)	2017	Implícita	Regressiva	α-CTL	Sim
CGAMER (TORRALBA et al., 2017)	2017	Implícita	Bidirecional	CTL	Sim
Planejador deste trabalho	2018	Implícita	Bidirecional	α-CTL	Sim

Fonte – Elaborada pelo autor.

4 BUSCA SIMBÓLICA BIDIRECIONAL

Este projeto de pesquisa propõe a incorporação da busca simbólica bidirecional no arcabouço de planejamento como verificação simbólica de modelos proposto por Menezes (2014), o qual utiliza busca unidirecional (progressiva ou regressiva). O método de busca simbólica bidirecional implementado foi inspirado no método proposto por (TORRALBA et al., 2017). No entanto, diferente deste método, é implementado em um arcabouço de verificação simbólica de modelos para a lógica α-CTL. Assim como nos métodos de busca progressiva e regressiva proposto por (MENEZES, 2014), BDDs são utilizados para representar e operar sobre conjuntos de estados do domínio de planejamento.

A busca simbólica bidirecional pode executar tanto a busca simbólica progressiva quanto a busca simbólica regressiva. A busca simbólica progressiva é iniciada do BDD que representa o estado inicial, s_0^{DD}, e avança até que encontre um estado que satisfaça a meta φ^{DD} ou, até que haja uma interseção com o BDD da busca simbólica regressiva. Por outro lado, a busca simbólica regressiva começa dos estados que satisfazem a meta, representando pelo BDD φ^{DD}, e é executada até que alcance o BDD que representa o estado inicial s_0^{DD} ou até que haja uma interseção com o BDD da busca simbólica progressiva.

A busca bidirecional implementada neste trabalho inicia avaliando o número de proposições do BDD que representa o estado inicial, s_0^{DD}, e o BDD que representa a meta φ^{DD}, assim, escolhendo para expandir o BDD que possui a menor cardinalidade. Se s_0^{DD} possui um número menor de proposições que φ^{DD}, então a busca progressiva é executada. Caso contrário, executa-se a busca regressiva. Assim, as duas buscas são realizadas de maneira intercalada, de modo que, a cada etapa, o algoritmo decide se será executada a *busca simbólica progressiva* ou a *busca simbólica regressiva*.

Para ter o controle de quais estados foram expandidos até um dado momento do algoritmo, são mantidos dois BDDs com os estados expandidos pela busca simbólica progressiva e busca simbólica regressiva, denominados X^{DD} e Y^{DD}, respectivamente.

Quando uma etapa é executada, os estados recém-gerados são comparados com o conjunto de estados expandidos pela busca simbólica oposta a que foi executada. No caso de uma correspondência, um plano solução foi encontrado, caso contrário o algoritmo segue a execução até que um dos dois métodos de busca simbólica tenha expandido todo o conjunto de estados (ponto-fixo encontrado).

4.1 Algoritmo

O algoritmo VERIFICADOR SIMBÓLICO PLANEX BIDIRECIONAL (Algoritmo 5) realiza uma busca simbólica progressiva a partir do BDD que representa o estado inicial s_0 e uma busca simbólica regressiva a partir do BDD que representa os estados que satisfazem a fórmula φ^{DD}. Ele recebe como entrada as ações $\mathbb{A}^{DD}$, um estado inicial s_0^{DD} e uma fórmula φ^{DD} que representa a meta de planejamento, todos representados por BDDs. Inicialmente, X^{DD} é o BDD que representa o estado inicial, denotado por s_0^{DD}, e Y^{DD}, o BDD que representa todos os estados que satisfazem a fórmula φ. Em cada iteração (linhas 4-10), o algoritmo verifica qual o BDD que possui a menor cardinalidade. Se for o X^{DD}, então é executada uma busca simbólica progressiva nesse BDD, caso contrário, é realizada uma busca simbólica regressiva em Y^{DD}. Em cada iteração, é verificado se o BDD X^{DD} e o BDD Y^{DD} possuem interseção entre si. Caso a interseção seja diferente de $\perp^{DD}$ (BDD vazio), então o problema é solucionável. Caso contrário, as buscas continuam sendo executadas de forma intercalada até que seja encontrado um ponto fixo, assim, o problema não possui solução.

Algoritmo 5: VERIFICADOR SIMBÓLICO PLANEX BIDIRECIONAL

Entrada: $\mathbb{A}^{DD}, s_0^{DD}, \varphi^{DD}$

```
1  início
2      X^DD := s_0^DD
3      Y^DD := STF(φ^DD)
4      repita
5          se nProposicoes(X^DD) < nProposicoes(Y^DD) então
6              X^DD = X^DD ∨ sprog(X^DD)
7          senão
8              Y^DD := Y^DD ∨ sregr(Y^DD)
9          se X^DD ∧ Y^DD ≠ ⊥^DD então
10             retorna "Problema solucionável."
11     até X^DD = Y^DD;
12     retorna "Problema não solucionável."
```

Fonte – Elaborado pelo autor.

5 PROCEDIMENTOS METODOLÓGICOS

A seguir, destacamos os procedimentos metodológicos adotados para conclusão deste projeto de pesquisa.

5.1 Implementação da Busca Simbólica Bidirecional no Arcabouço de Planejamento com Verificação Simbólica de Modelos

O Arcabouço de Planejamento com Verificação Simbólica de Modelos proposto por Menezes (2014) utiliza a busca simbólica progressiva e a busca simbólica regressiva na tarefa de verificar a existência de um plano para um problema de planejamento. É Implementado o método de busca simbólica bidirecional neste arcabouço, utilizando:

- BDDs para representar os conjuntos de proposições durante a busca;
- como métrica para a escolha do método de qual busca simbólica será executada em cada passo (se progressiva ou regressiva), o princípio de escolha do BDD com menor cardinalidade respectivo a busca.

As ideias para implementação da expansão da busca bidirecional são oriundas do trabalho de Torralba et al. (2017).

5.2 Seleção dos Domínios para Avaliação dos Algoritmos

Os domínios que são utilizados para a avaliação do algoritmo proposto neste trabalho e os algoritmos propostos por Menezes (2014) são da Competição Internacional de Planejamento (IPC) e da Competição Internacional de Planejamento para Problemas sem Solução (UIPC). Para os testes com problemas de planejamento que possuem solução, utiliza-se o domínio de Logística, proposto em 1998. Já para os testes com problemas de planejamento que não possuem solução, são utilizados os domínios *sliding-tiles* e *bottleneck*, propostos em 2016.

5.3 Teste dos Algoritmos nos Domínios da Competição Internacional de Planejamento

Cada problema de planejamento escolhido é submetido aos métodos de busca simbólica unidirecional (busca simbólica progressiva e busca simbólica regressiva) e busca simbólica bidirecional. Para cada problema é verificado o tempo de execução de cada uma das buscas e a quantidade de camadas expandidas na tarefa de verificar a existência de um plano.

5.4 Análise Comparativa da Busca Simbólica Bidirecional em Relação ao Desempenho da Busca Simbólica Unidirecional

Após a execução dos testes e obtenção dos resultados, a análise é feita separadamente para cada domínio de planejamento e, assim, é possível constatar se a proposta de implantação da busca simbólica bidirecional no arcabouço de verificação simbólica de modelos melhorou em comparação aos algoritmos de busca simbólica progressiva e busca simbólica regressiva.

6 ANÁLISE EXPERIMENTAL

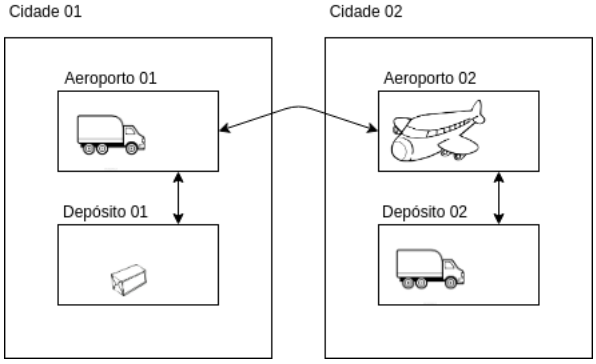
6.1 Domínios Benchmarks

Neste trabalho foram utilizados o domínio de Logística, proveniente do *track* clássico da Competição Internacional de Planejamento (IPC) e os domínios *sliding-tile* e *bottleneck*, provenientes da Competição Internacional de Planejamento para Problemas sem Solução (UIPC).

6.1.1 Domínio de Logística

A tarefa do domínio de logística é a de transportar um pacote de um local de origem para um local de destino. O ambiente desse problema é definido como *n* cidades, onde cada cidade possui um depósito e um aeroporto, sendo que existe uma estrada ligando cada depósito ao aeroporto da mesma cidade e cada aeroporto possui uma linha de vôo para qualquer outro aeroporto de outra cidade. Existem dois meios de transporte para o pacote, o caminhão para transportar entre o aeroporto e o depósito de uma mesma cidade, e o avião para fazer o transporte entre os aeroportos das cidades. As ações que podem ser executadas nesse domíno são: *carregar-caminhao*, onde o caminhão e o pacote devem estar no mesmo local, assim, é possível carregar o pacote para dentro do caminhão; *carregar-aviao*, em que é possível carregar o pacote para dentro do avião, sendo que o pacote e o avião devem estar no mesmo local como pré-condição; *descarregar-caminhao*, ação que retira o pacote de dentro do caminhão; *descarregar-aviao*, onde é retirado o pacote de dentro do avião; *dirigir-caminhao*, ação que move o caminhão de um determinado local para outro na mesma cidade e, por fim; *voar-aviao*, ação aplicável para que o avião voe de um determinado aeroporto de origem para um aeroporto de destino.

Figura 11 – Exemplo de um problema no domínio de Logística.



Fonte – Elaborada pelo Autor.

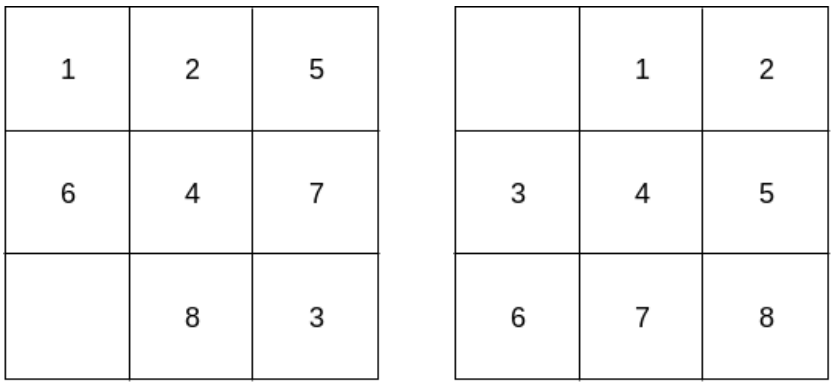
A Figura 11 exemplifica o domínio de logística, no qual existem duas cidades, em que cada cidade possui um depósito, um aeroporto e um caminhão para fazer o transporte entre o depósito e o aeroporto desta cidade, e ainda um avião para transportar cargas entre os aeroportos. Por fim, ainda é possível ver que existe um pacote, objeto no qual é o objetivo de transporte do domínio.

6.1.2 Domínio sliding-tiles

O domínio *sliding-tiles* modela a tarefa do clássico problema do quebra-cabeças de blocos deslizantes. Neste domínio, um estado representa um tabuleiro $N \times N$ com peças numeradas e um quadrado vazio. Uma peça adjacente ao quadrado vazio pode deslizar para esse quadrado.

Esse domínio possui 4 ações: *mover-para-cima*, onde o espaço vazio é trocado com o número que está logo acima, se existir; *mover-para-baixo*, ação que troca o espaço vazio com o número logo abaixo, se possível; *mover-para-esquerda*, onde o espaço vazio é trocado com o número que está imediatamente a esquerda, se existir, e, por fim; *mover-para-direita*, que troca o espaço vazio com o número imediatamente a direta, se possível.

Figura 12 – Exemplo de um problema no domínio *sliding-tiles*.



(a) Estado Inicial (b) Estado Meta

Fonte – Elaborada pelo autor.

A Figura 12 apresenta um exemplo de problema neste domínio, onde a Figura 12(a) representa o estado inicial e a Figura 12(b) representa a meta.

Para que um problema no domínios *sliding-tiles* não possua solução é necessário que, no estado inicial, a soma do número de inversões dos números como estão dispostos seja par (KORF; TAYLOR, 1996). Por exemplo, a disposição dos números da Figura 12(a) fica da

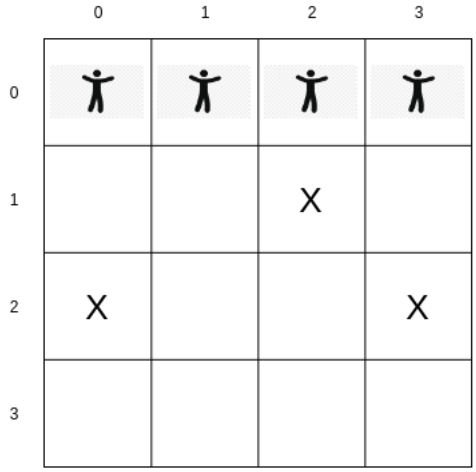
seguinte forma: 1, 2, 5, 6, 4, 7, 8 e 3, o número de inversões de cada dígito é feito contando a quantidade de números menores que ele estão a sua direita. A soma do número de inversões para esse problema é 7, logo esse é um exemplo de um problema de planejamento que não possui solução, pois não é possível obter um plano aplicando as ações estabelecidas.

6.1.3 Domínio bottleneck

A tarefa do domínio *bottleneck* é a de movimentar *n* pessoas, posicionadas inicialmente nas *n* posições da primeira linha de um tabuleiro, para as *n* posições da última linha do tabuleiro. As limitação do domínio são: (i) os quadrados em que as pessoas estão ou que já passaram tornam-se inativos, ou seja, não podem mais ser ocupados por pessoas futuramente; (ii) alguns espaços já iniciam inativos e; (iii) duas pessoas não podem ocupar o mesmo quadrado.

A única ação existente nesse domínio é a de uma pessoa *p* se *mover* de um quadrado *a* para outro *b*, para isso é necessário que a pessoa esteja no quadrado *a* e que o quadrado *b* esteja ativo. Após a execução da ação de mover, a pessoa estará no local *b* e esse não estará mais ativo.

Figura 13 – Exemplo de um problema no domínio *bottleneck*.



Fonte – Elaborada pelo Autor.

A Figura 13 apresenta um exemplo de um problema no domíno *bottleneck*, onde existem 4 pessoas posicionadas nos 4 quadrados da primeira linha e alguns locais do tabuleiro já iniciam inválidos. O objetivo desse problema é, aplicando a ação de *mover*, fazer com que as 4 pessoas cheguem nos 4 quadrados da última linha sem passar por locais inválidos. Esse é um exemplo de um problema de planejamento que não possui solução, pois não é possível obter um plano aplicando a ação estabelecida.

6.2 Resultados

Os algoritmos foram implementados em `Java`, utilizando-se a biblioteca `JavaBdd` que contém as operações para manipulação dos BDDs. Definimos como parâmetro para os BDDs um número máximo de nós de $9x10^6$ e o tamanho da cache de $9x10^5$. Os testes foram executados em uma máquina com processador Intel Core $i7-7500U@2.70GHz$ x 4 e $8GB$ de memória RAM dedicados para a máquina virtual Java (JVM). No domínio de logística foram testados 6 problemas e nos domínios *sliding-tiles* e *bottleneck* foram testados 25 problemas em cada.

A Tabela 2 apresenta o resultado da *busca simbólica regressiva* no domínio da Logística, na qual é possível ver que foi possível verificar a existência de um plano somente para os problemas Logistics-04 e Logistics-06. Nos demais problemas ocorreu um problema de memória insuficiente.

Tabela 2 – Resultado da busca simbólica regressiva no domínio de Logística.

Problema	Número de Proposições	Número de Ações	Tamanho do Plano	Profundidade Alcançada	Tempo
Logistics-04	48	78	20	20	3,085
Logistics-06	48	78	25	25	2,391
Logistics-08	99	171	31	16	-
Logistics-10	168	308	48	7	-
Logistics-12	168	308	43	9	-
Logistics-14	275	648	58	5	-

Fonte – Elaborada pelo autor.

A Tabela 3 apresenta o resultado da *busca simbólica progressiva* no domínio da Logística, na qual é possível ver que foi possível verificar a existência de um plano para os problemas Logistics-04 e Logistics-06. Nos demais problemas ocorreu um problema de memória insuficiente.

Tabela 3 – Resultado da busca simbólica progressiva no domínio de Logística.

Problema	Número de Proposições	Número de Ações	Tamanho do Plano	Profundidade Alcançada	Tempo
Logistics-04	48	78	20	20	1,727
Logistics-06	48	78	25	25	1,584
Logistics-08	99	171	31	28	-
Logistics-10	168	308	48	14	-
Logistics-12	168	308	43	14	-
Logistics-14	275	648	58	9	-

Fonte – Elaborada pelo autor.

A Tabela 4 apresenta o resultado da *busca simbólica bidirecional* no domínio da Logística, na qual é possível ver que foi possível verificar a existência de um plano para os problemas Logistics-04, Logistics-06 e Logistics-08. A busca simbólica bidirecional conseguiu verificar a existência de um plano para o problema Logistics-08, o que não foi possível na busca simbólica regressiva e na busca simbólica progressiva. Nos demais problemas ocorreu um problema de memória insuficiente.

Tabela 4 – Resultado da busca simbólica bidirecional no domínio de Logística.

Problema	Número de Proposições	Número de Ações	Tamanho do Plano	Profundidade Alcançada	Tempo
Logistics-04	48	78	20	20	1,395
Logistics-06	48	78	25	25	1,472
Logistics-08	99	171	31	31	3445,6
Logistics-10	168	308	48	19	-
Logistics-12	168	308	43	20	-
Logistics-14	275	648	58	15	-

Fonte – Elaborada pelo autor.

A Tabela 5 apresenta os resultados obtidos pelas *buscas simbólicas* no domínio *sliding-tiles*, na qual é possível ver que cada busca simbólica, separadamente, obteve desempenho semelhante para os 13 problemas iniciais, os quais são referentes a resolver um tabuleiro de tamanho $3x3$. A busca simbólica que obteve o melhor desempenho foi a busca simbólica progressiva, ficando um pouco a frente da busca simbólica bidirecional. Os problemas 14 à 25 são referentes a resolver um tabuleiro de dimensões $4x3$, na execução das buscas para esses problemas ocorreu um problema de memória insuficiente.

Tabela 5 – Resultado das buscas simbólicas no domínio *sliding-tiles*.

Problema	Busca Simbólica Regressiva	Busca Simbólica Progressiva	Busca Simbólica Bidirecional
01	530,586	10,320	15,658
02	530,311	10,808	16,494
03	551,834	10,362	15,661
04	545,644	9,796	15,387
05	549,692	10,414	15,536
06	534,661	11,558	16,834
07	526,787	11,723	17,123
08	543,472	10,761	15,181
09	535,351	12,153	16,282
10	551,391	11,401	16,126
11	534,781	11,172	16,985
12	535,244	11,650	16,526
13	550,327	11,631	16,339

Fonte – Elaborada pelo autor.

A Tabela 6 apresenta os resultados obtidos pelas *buscas simbólicas* no domínio *bottleneck*, na qual é possível ver que a busca simbólica progressiva e a busca simbólica bidirecional, resolvendo 8 problemas, foram melhores que a busca simbólica regressiva, que resolveu 4 problemas, para esse domínio. Nos problemas 9 a 25 ocorreu um problema de memória insuficiente.

Tabela 6 – Resultado das buscas simbólicas no domínio *bottleneck*.

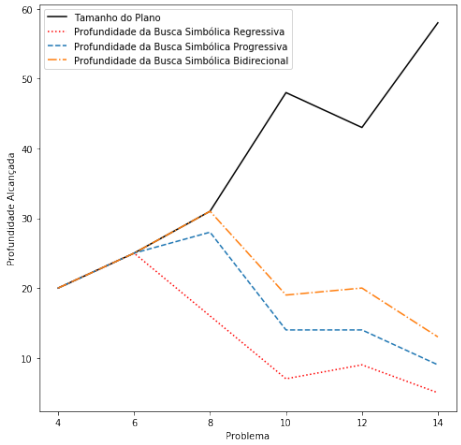
Problema	Busca Simbólica Regressiva	Busca Simbólica Progressiva	Busca Simbólica Bidirecional
01	1,130	0,146	0,154
02	3,057	0,510	0,641
03	11,979	1,120	1,556
04	467,316	2,218	2,302
05	-	4,962	6,162
06	-	21,184	31,733
07	-	85,644	118,766
08	-	4423,497	19705,301

Fonte – Elaborada pelo autor.

6.3 Análise Comparativa de Desempenho

A Figura 14 mostra a profundidade alcançada por cada busca simbólica e o tamanho do plano para cada problema. Comparando a profundidade alcançada na busca, pode-se afirmar que a busca simbólica regressiva foi a que obeteve o pior rendimento nesse domínio, e a busca simbólica que obteve o resultado mais satisfatório foi a busca simbólica bidirecional, que resolveu um problema a mais que as demais e alcançou uma profundidade maior que as outras buscas nos problemas que ocorreu problema de memória insuficiente.

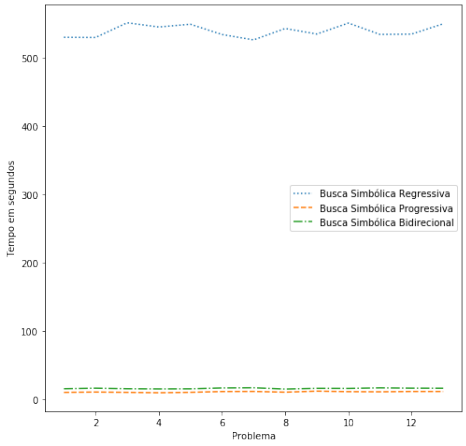
Figura 14 – Gráfico de comparação da profundidade de camadas alcançada pelas buscas no domínio de Logística.



Fonte – Elaborada pelo Autor.

A Figura 15 apresenta o desempenho das buscas simbólicas no domínio *sliding-tiles*, considerando como fator o tempo. Nos problemas que foi possível verificar que o mesmo não tinha solução, a busca simbólica regressiva foi a busca que obteve o pior resultado, e a busca simbólica progressiva e a busca simbólica bidirecional obtiveram os melhores resultados, e muito próximos. Se for considerado uma margem de erro de poucos segundos, essas buscas podem ter seu desempenho considerado como empatados para esse domínio.

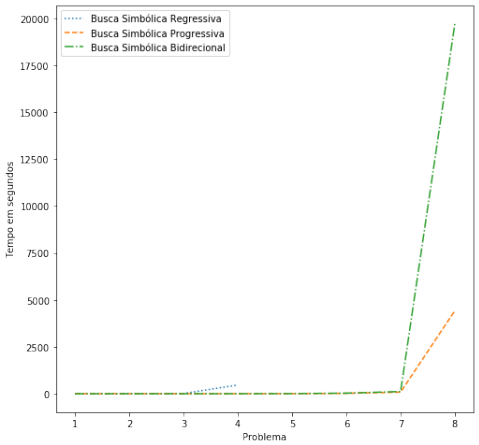
Figura 15 – Gráfico de comparação do tempo das buscas no domínio *sliding-tiles*.



Fonte – Elaborada pelo Autor.

A Figura 16 apresenta o desempenho das buscas simbólicas no domínio *bottleneck*, considerando também o tempo como fator. Nos problemas que foi possível verificar a existência ou não de um plano, a busca simbólica obteve o pior desempenho. A busca simbólica progressiva foi a que obteve o melhor resultado, resolvendo mais problemas que a busca simbólica regressiva e a mesma quantidade da busca simbólica bidirecional, mas com um melhor desempenho de tempo de execução. A busca simbólica bidirecional resolveu o mesmo número de problemas que a busca simbólica progressiva, porém essa busca gastou mais tempo na verificação da existência de plano para os problemas.

Figura 16 – Gráfico de comparação do tempo das buscas no domínio *bottleneck*.



Fonte – Elaborada pelo Autor.

7 CONSIDERAÇÕES FINAIS

Neste trabalho foi realizada a incorporação da busca simbólica bidirecional no arcabouço de planejamento como verificação simbólica de modelos baseado na lógica α-CTL. Os testes foram realizados em dois *tracks* da Competição Internacional de Planejamento (IPC), o *track* clássico e o *track unsolvability* (problemas sem solução).

Os domínios considerados nos testes foram: Logística, para o *track* clássico, e *sliding-tiles* e *bottleneck*, para o *track unsolvability*. A comparação dos algoritmos de busca simbólica regressiva, busca simbólica progressiva e busca simbólica bidirecional foi realizada considerando como fatores de medida o tempo e a profundidade alcançada, esse último somente para o *track* clássico.

A busca simbólica bidirecional obteve melhor desempenho no *track* clássico, com melhores tempos e uma maior profundidade alcançada no domínio de Logística. No entanto, considerando o *track unsolvability*, a busca simbólica que obteve o melhor desempenho foi a progressiva, nos domínios *sliding-tiles* e *bottleneck*.

Como trabalhos futuros pode-se destacar:

- Implementação de métricas para a escolha de qual busca simbólica executar em um dado momento da busca simbólica bidirecional;
- Implementação de heurísticas na busca simbólica bidirecional para, possivelmente, melhorar o desempenho na verificação da existência de um plano;
- Testes com mais domínios *benchmarks* do *track* clássico;
- Testes com mais domínios *benchmarks* do *track unsolvability*.

REFERÊNCIAS

BACCHUS, F. Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. **Ai magazine**, v. 22, n. 3, p. 47, 2001.

BALYO, T.; SUDA, M. Reachlunch entering the unsolvability ipc 2016. **Unsolvability IPC: planner abstracts**, p. 3–5, 2016.

BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. **Artificial intelligence**, Elsevier, v. 90, n. 1-2, p. 281–300, 1997.

BRYANT, R. E. Graph-based algorithms for boolean function manipulation. **Computers, IEEE Transactions on**, IEEE, v. 100, n. 8, p. 677–691, 1986.

BUNING, H. K.; KARPINSKI, M.; FLOGEL, A. Resolution for quantified boolean formulas. **Information and computation**, Elsevier, v. 117, n. 1, p. 12–18, 1995.

BURCH, J. R.; CLARKE, E. M.; MCMILLAN, K. L.; DILL, D. L.; HWANG, L.-J. Symbolic model checking: 1020 states and beyond. **Information and computation**, Elsevier, v. 98, n. 2, p. 142–170, 1992.

CIMATTI, A.; PISTORE, M.; ROVERI, M.; TRAVERSO, P. Weak, strong, and strong cyclic planning via symbolic model checking. **Artificial Intelligence**, Elsevier, v. 147, n. 1-2, p. 35–84, 2003.

CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In: SPRINGER. **Workshop on Logic of Programs**. [S.l.], 1981. p. 52–71.

EDELKAMP, S.; HELMERT, M. On the implementation of mips. In: **AIPS-Workshop on Model-Theoretic Approaches to Planning**. [S.l.: s.n.], 2000. p. 18–25.

EDELKAMP, S.; HELMERT, M. Mips: The model-checking integrated planning system. **AI magazine**, v. 22, n. 3, p. 67, 2001.

FOURMAN, M. P. Propositional planning. In: **Proceedings of AIPS-00 Workshop on Model-Theoretic Approaches to Planning**. [S.l.: s.n.], 2000. p. 10–17.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning:** theory and practice. [S.l.]: Elsevier, 2004.

GNAD, D.; STEINMETZ, M.; HOFFMANN, J. Django: Unchaining the power of red-black planning. **Unsolvability IPC: planner abstracts**, p. 19–22, 2016.

GNAD, D.; TORRALBA, Á.; HOFFMANN, J.; WEHRLE, M. Decoupled search for proving unsolvability. **Unsolvability IPC: planner abstracts**, p. 16–18, 2016.

GÖBELBECKER, M.; KELLER, T.; EYERICH, P.; BRENNER, M.; NEBEL, B. Coming up with good excuses: What to do when no plan can be found. **Cognitive Robotics**, v. 10081, 2010.

HASLUM, P. Adapting h++ for proving plan non-existence. **Unsolvability IPC: planner abstracts**, p. 1–1, 2016.

HUTH, M.; RYAN, M. **Logic in computer science:** Modelling and reasoning about systems. [S.l.]: Cambridge university press, 2004.

KORF, R. E.; TAYLOR, L. A. Finding optimal solutions to the twenty-four puzzle. In: **Proceedings of the national conference on artificial intelligence**. [S.l.: s.n.], 1996. p. 1202–1207.

KOROVIN, K.; SUDA, M. iproverplan: a system description. **Unsolvability IPC: planner abstracts**, p. 6–7, 2016.

KRIPKE, S. **Semantical considerations of the modal logic**. In: . [S.l.: s.n.], 2007.

LONG, D.; FOX, M. The 3rd international planning competition: Results and analysis. **Journal of Artificial Intelligence Research**, v. 20, p. 1–59, 2003.

MENEZES, M. V. **Mudanças em Problemas de Planejamento sem Solução**. Tese (Doutorado) — Tese de Doutorado, Universidade de Sao Paulo, Instituto de Matemática e Estatıstica, 2014.

MENEZES, M. V. de; BARROS, L. N. de; PEREIRA, S. do L. **Symbolic Regression for Non-Deterministic Actions**. In: . [S.l.: s.n.], 2014.

MUISE C.; LIPOVETZKY, N. **Unsolvability international planning competition**. 2016. https://unsolve-ipc.eng.unimelb.edu.au/. Acesso: 28/11/2018.

NILSSON, N. J. **Principles of artificial intelligence**. [S.l.]: Morgan Kaufmann, 2014.

PEREIRA, S. do L.; BARROS, L. N. de. **Planejamento sob incerteza para metas de alcançabilidade estendidas**. Tese (Doutorado) — Tese de Doutorado, Universidade de Sao Paulo, Instituto de Matemática e Estatıstica, 2007.

POMMERENING, F.; SEIPP, J. Fast downward dead-end pattern database. **Unsolvability IPC: planner abstracts**, p. 2–2, 2016.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SANTOS, V. B. dos; BARROS, L. N. de. **Pactl-sym**: um planejador baseado em verificação simbólica de modelos. In: . [S.l.: s.n.], 2017.

SEIPP, J.; POMMERENING, F.; SIEVERS, S.; WEHRLE, M.; FAWCETT, C.; ALKHAZRAJI, Y. Fast downward aidos. **Unsolvability IPC: planner abstracts**, p. 28–38, 2016.

STEINMETZ, M.; HOFFMANN, J. Clone: A critical-path driven clause learner. **UIPC 2016 planner abstracts**, p. 24–27, 2016.

TARSKI, A. A lattice-theoretical fixpoint theorem and its applications. **Pacific journal of Mathematics**, Mathematical Sciences Publishers, v. 5, n. 2, p. 285–309, 1955.

TORRALBA, A. Sympa: Symbolic perimeter abstractions for proving unsolvability. **Unsolvability IPC: planner abstracts**, p. 8–11, 2016.

TORRALBA, Á.; ALCÁZAR, V.; KISSMANN, P.; EDELKAMP, S. Efficient symbolic search for cost-optimal planning. **Artificial Intelligence**, Elsevier, v. 242, p. 52–79, 2017.

TORRALBA, Á.; HOFFMANN, J.; KISSMANN, P. Ms-unsat and simulationdominance: Merge-and-shrink and dominance pruning for proving unsolvability. **Unsolvability IPC: planner abstracts**, p. 12–15, 2016.