UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MACILIO DA SILVA FERREIRA

ANÁLISE DO PLANEJADOR SIMBÓLICO ALPHA-CTL NA DETECÇÃO DE PROBLEMAS DE PLANEJAMENTO SEM SOLUÇÃO

QUIXADÁ

2017

MACILIO DA SILVA FERREIRA

ANÁLISE DO PLANEJADOR SIMBÓLICO ALPHA-CTL NA DETECÇÃO DE PROBLEMAS DE PLANEJAMENTO SEM SOLUÇÃO

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: Drª. Maria Viviane de Menezes

Coorientador: Dr°. Paulo de Tarso Guerra Oliveira

QUIXADÁ

2017

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)

F442a Ferreira, Macilio da Silva.
Análise do planejador simbólico Alpha-CTL na detecção de problemas de planejamento sem solução / Macilio da Silva Ferreira. – 2017.
61 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2017.
Orientação: Profa. Dra. Maria Viviane de Menezes.
Coorientação: Prof. Dr. Paulo de Tarso Guerra Oliveira.

1. Planejamento automatizado. 2. Inteligência artificial. 3. Verificação de modelos. 4. Lógica temporal. I. Título.

CDD 004

MACILIO DA SILVA FERREIRA

ANÁLISE DO PLANEJADOR SIMBÓLICO ALPHA-CTL NA DETECÇÃO DE PROBLEMAS DE PLANEJAMENTO SEM SOLUÇÃO

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

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

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

Prof. Críston Pereira de Souza
Universidade Federal do Ceará - UFC

Dedicado em memória de Raimundo Lopes de Souza e Maria José Lopes da Silva.

AGRADECIMENTOS

Agradeço muito a Deus por todas as bençãos que me foram dadas, principalmente por me dar forças para superar os momentos difíceis.

À professora Maria Viviane de Menezes, pela excelente orientação. Sempre se dedicando e contribuindo com ideias para melhoria deste projeto de pesquisa. Ao professor Paulo de Tarso Guerra Oliveira, por está sempre disposto a ajudar. A todos os professores da Universidade Federal do Ceará - Campus de Quixadá que contribuíram para a minha formação. Em especial, aos professores participantes da banca examinadora Maria Viviane de Menezes, Paulo de Tarso Guerra Oliveira e Criston Pereira de Souza pelo tempo, pelas valiosas colaborações e sugestões.

À toda minha família, que apesar da distância, esteve presente nos momentos difíceis dessa jornada, sempre me apoiando e compreendendo minhas ausências para a conclusão deste trabalho. Em especial, aos meus pais, Manoel Salvador do Nascimento Ferreira e Marcia Maria da Silva Ferreira, a minha irmã Magna da Silva Ferreira. Ao meu irmão Mardson da Silva Ferreira, pelo apoio emocional estando sempre disposto a me ajudar. Amo vocês e agradeço de coração por todo apoio, que foi fundamental para a minha formação. A todos os amigos que fiz durante a graduação, agradeço a cada um de vocês que contribuíram diretamente ou indiretamente para a conclusão deste projeto de pesquisa. Em especial, a Joyce Nayne, pelo incentivo e apoio diário.

“Eu acredito que ás vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar. “

(Alan Turing)

RESUMO

Planejamento é a subárea da Inteligência Artificial que preocupa-se em propor uma sequência de ações de modo a alcançar um objetivo. A solução de um problema de planejamento é uma sequência de ações, denominada plano, que possibilita o agente alcançar um estado que satisfaz o objetivo (estado meta) a partir de um estado inicial. Assim, quando não é possível encontrar um plano que alcance a meta, dizemos que o problema de planejamento não possui solução. A Competição Internacional de Planejamento para Problemas sem Solução (UIPC) foi realizada no âmbito da IPC, pela primeira vez em 2016. O objetivo da UIPC-2016 é analisar algoritmos capazes de detectar se um problema de planejamento possui ou não solução. O presente trabalho propõe uma avaliação do algoritmo de planejamento como verificação simbólica de modelos baseado na lógica α-CTL (MENEZES, 2014) para detecção de problemas de planejamento sem solução utilizando os domínios fornecidos pela competição. Deste modo, realizamos uma comparação entre os resultados obtidos pelo algoritmo de planejamento como verificação simbólica de modelos baseado na lógica α-CTL e os resultados obtidos pelos algoritmos participantes da UIPC-2016.

Palavras-chave: Planejamento Automatizado. Inteligência Artificial. Verificação de Modelos. Lógica Temporal. Problemas sem Solução.

ABSTRACT

Planning is the subarea of Artificial Intelligence that is concerned with proposing a sequence of actions in order to achieve a goal. The solution to a planning problem is a sequence of actions, called a plan, that enables the agent to reach a state that satisfies the goal (goal state) from an initial state. Thus, when it is not possible to find a plan that reaches the goal, we say that the planning problem has no solution. The first unsolvability international planning competition (UIPC) was held in 2016 by International Planning Competition (IPC), with the goal of determining whether a given planning task is unsolvable. The present work proposes an evaluation of the planning algorithm as a symbolic verification of models based on the α-CTL logic (MENEZES, 2014) to detect problems of planning without solution using the domains provided by the competition. Thus, we perform a comparison between the results obtained by the planning algorithm as a symbolic verification of models based on the α-CTL logic and the results obtained by the participant algorithms of the UIPC-2016.

Keywords: Automated Planning. Artificial intelligence. Model Check. Temporal Logic. Unsolvable Problems.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Na vida real, todos dias, nos deparamos com a necessidade de planejar nossas ações para que nossas metas sejam alcançadas (PEREIRA, 2007). Por exemplo, um estudante de graduação, que possui como meta concluir seu curso, precisa planejar a cada semestre quais disciplinas irá cursar para que ao final consiga integralizar os créditos exigidos por seu curso. Uma viagem de férias também precisa ser planejada, uma vez que é necessário comprar passagens, reservar hospedagem, selecionar lugares a visitar, etc.

Neste contexto, *Planejamento* é o processo de escolha e organização de ações que procura satisfazer objetivos previamente estabelecidos. *Planejamento Automatizado*, por sua vez, é a subárea da Inteligência Artificial que estuda esse processo de raciocínio usando o computador, há quase 60 anos (NEWELL; SHAW; SIMON, 1959).

1.1 Planejamento Automatizado

Um planejador (Figura 1) é um algoritmo que recebe como entrada: (i) a dinâmica do ambiente em que o agente irá atuar, denominada *domínio de planejamento*; (ii) a situação inicial que o agente encontra-se (denominado *estado inicial*) e; (iii) uma meta a ser alcançada. Retorna como saída um *plano* de ações para que o agente saia do estado inicial e atinja suas metas ou; *falha*, se não for possível obter tal plano.

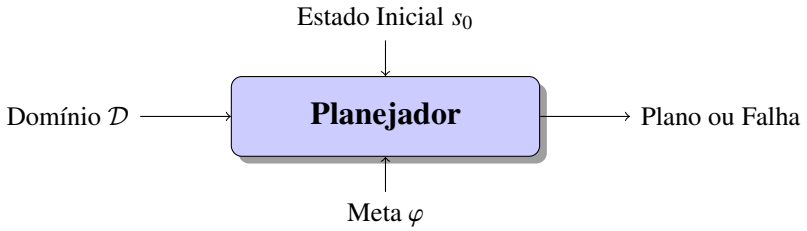


Figura 1 – *Planejador* (PEREIRA, 2007).

Dado um conjunto de proposições $\mathbb{P}$ que descrevem as configurações do mundo, um *domínio de planejamento* pode ser dado explicitamente por meio de um grafo orientado e rotulado. A Figura 2 apresenta o grafo de transições para um domínio de planejamento denominado *mundo dos blocos* (NILSSON, 1980). Os nós do grafo representam os estados do ambiente, os quais são rotulados pelo conjunto de proposições que são verdadeiras no estado[1].

[1] Adota-se a suposição do mundo fechado (*closed world assumption*) na qual representa-se apenas o que é verdadeiro em cada estado. As proposições cujos valores são falsos não são representadas.

Por exemplo, o estado s_{11} da Figura 2 é descrito pelas proposições: $sobre(A,B)$, indicando que o bloco A está sobre o bloco B; $sobre(B,C)$, indicando que o bloco B está sobre o bloco C; livre(A), indicando que não há blocos sobre o bloco A e $sobre(C,Mesa)$, indicando que o bloco C está sobre a mesa. As arestas são rotuladas com as ações que o agente é capaz de executar.

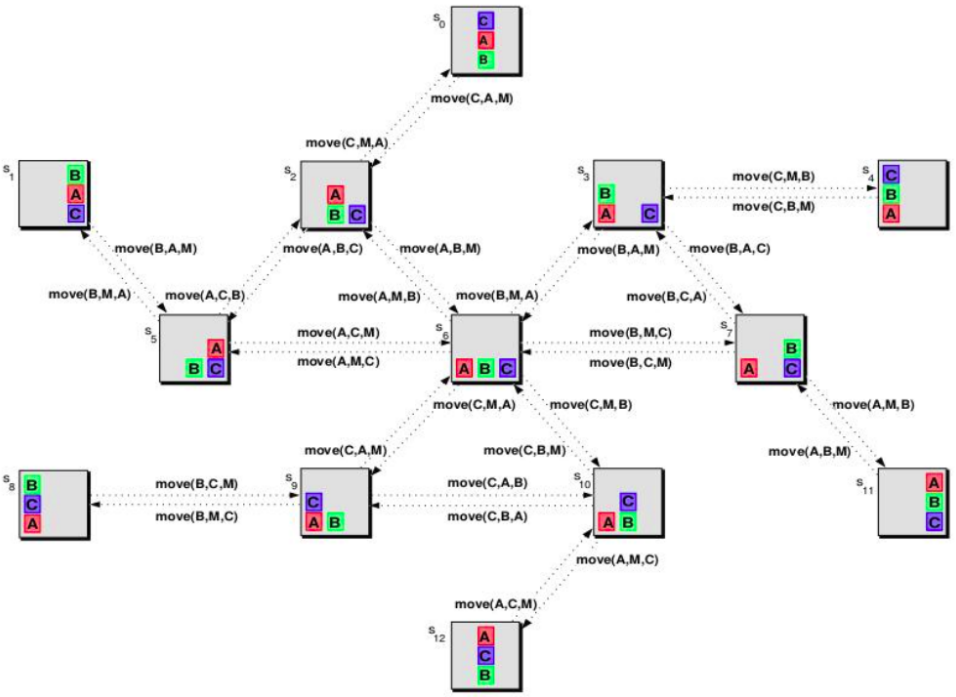


Figura 2 – Grafo de transições para o domínio do mundo dos blocos (PEREIRA, 2007).

Para cada domínio, é possível definir um conjunto de *problemas de planejamento*. Em cada problema, define-se o *estado inicial* em que o agente encontra-se e o objetivo que deve ser alcançado, denominado *meta de planejamento*. A solução para um problema de planejamento é denominada *plano*, uma sequência de ações que levam o agente do estado inicial a um estado satisfazendo a meta.

Exemplo 1. Suponha a representação do domínio do mundo dos blocos ilustrada na Figura 2. Imagine que inicialmente o agente esteja no estado s_4 (onde o bloco C está sobre o bloco B, o bloco B está sobre o bloco A e o bloco A está sobre a mesa M). O objetivo do agente é alcançar o estado s_{11}, onde *o bloco A está sobre o bloco B, o bloco B está sobre o bloco C e o bloco C está sobre a mesa M*. Assim, uma solução possível para o problema é a seguinte sequência de ações: *move(C,B,M), move(B,A,C) e move(A,M,B)*.

Na maioria das vezes, modelos realísticos de ambientes possuem um número muito grande de estados e buscar a solução estado a estado torna-se uma tarefa computacionalmente

inviável (problema da explosão de estados). Por este motivo, em geral, é utilizada uma representação implícita para descrever o domínio de planejamento. Nesta representação, o domínio é dado por uma linguagem de ações, tais como: STRIPS (Stanford Research Institute Problem Solver) (FIKES; NILSSON, 1971), PDDL (Plannig Description Domain Language) (MCDERMOTT et al., 1998) ou RDDL (Relational Dynamic Influence Diagram Language) (SANNER, 2010)). As ações são então utilizadas posteriormente para induzir um subgrafo em uma busca a partir de um estado inicial (busca progressiva) ou a partir de um conjunto de estados meta (busca regressiva). Essas ações são definidas em termos de suas pré-condições, que descrevem as condições que precisam ser verdadeiras no estado s onde a ação a é executada, e efeitos, que descrevem como o estado s é modificado pela execução da ação a.

A Figura 3 apresenta a descrição em *PDDL* da ação *move* do domínio do mundo dos blocos, onde X, Y e Z são blocos. Para a realização da ação $move(X,Y,Z)$ as pré-condições são: bloco(X), sobre(X,Y), livre(X) e livre(Z), ou seja, que X seja um bloco, que ele esteja sobre o bloco Y, que não tenha nenhum bloco sobre X e que não tenha nenhum bloco sobre Z. A ocorrência da ação causa os seguintes efeitos: sobre(X,Z), livre(Y), $\neg sobre(X,Y)$ e $\neg livre(Z)$, ou seja, o bloco X estará sobre o bloco Z, logo o bloco Z não estará mais livre, o bloco Y está livre, visto que o bloco X não está mais sobre ele.

```
move(X,Y,Z)
precondição:  bloco(X), sobre(X,Y), livre(X), livre(Z)
efeitos:  sobre(X,Z), livre(Y), ¬sobre(X,Y), ¬livre(Z)
```

Figura 3 – Descrição em *PDDL* da ação $move(X,Y,Z)$.

A complexidade de decidir se um problema de planejamento possui ou não solução (PLANEX *problem*) é PSPACE (EROL; NAU; SUBRAHMANIAN, 1995), uma classe de complexidade que se refere a problemas que podem ser resolvidos por uma máquina de Turing determinística com uma quantidade de espaço polinomial (RUSSELL STUART, 2009).

1.2 Problemas de Planejamento sem Solução

Como mostrado na Figura 1, nem sempre é possível obter um plano para um problema de planejamento. Observe a seguir, um problema de planejamento em um domínio denominado *domínio das chaves* (GÖBELBECKER et al., 2010).

Exemplo 2. (Domínios das Chaves) No *domínio das chaves* um robô deve navegar entre as salas de um ambiente e, se necessário, deve abrir portas com chaves específicas. Neste problema, temos duas salas ($sala_0$ e $sala_1$); uma chave ($chave_0$) e uma porta ($porta_0$). Um conjunto de proposições $\mathbb{P}$ descreve a localização do robô, a localização das chaves, o estado das portas (*aberta* ou *fechada*), qual chave abre uma determinada porta e as conexões entre as salas. O conjunto de ações $\mathbb{A}$ representa as habilidades do robô no ambiente: abrir e fechar portas e mover de uma sala para outra, carregando ou não uma chave. Por exemplo, a ação *mover-0-1*, move o robô da sala 0 para a sala 1, a ação *moverchave-0-1* movimenta o robô, juntamente com a chave, da sala 0 para a sala 1, as ações *abrir1* e *fechar1* fazem, respectivamente, o robô abrir e fechar a porta a partir da sala 1 (MENEZES, 2014).

Suponha um problema de planejamento neste domínio em que o estado inicial é o estado s_0, mostrado na Figura 4. Neste estado o robô está na sala_0, a porta está fechada e a chave encontra-se na sala_1. O objetivo do robô é ir para a sala_1. Neste caso, não é possível obter uma solução para o problema de planejamento, uma vez que a chave que abre a porta está localizada na sala_1 e o robô não possui meios de obter tal chave.

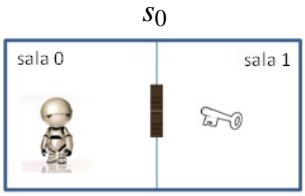


Figura 4 – Estado inicial de um *Problema de Planejamento sem Solução* .

No entanto, o principal foco dos trabalhos na área de Planejamento em Inteligência Artificial têm sido o desenvolvimento de algoritmos eficientes para encontrar planos de ações (KAUTZ; SELMAN et al., 1992; CIMATTI et al., 2003; HOFFMANN, 2001; HELMERT, 2006) para os diferentes tipos de problemas de planejamento: determinísticos, não-determinísticos, probabilísticos, com metas super especificadas, com ações com duração específica, etc. *Problemas sem solução* tornaram-se um grande desafio para os planejadores estado-da-arte (BALYO; SUDA, 2016), uma vez que a maioria dos algoritmos são baseados em busca heurística e esta nem sempre é capaz de distinguir a dificuldade de encontrar uma solução para um dado problema da inexistência dessa solução. Assim, somente nos últimos anos, alguma atenção foi dada no estudo de algoritmos capazes de detectar de forma eficiente quando um problema de planejamento não possui solução.

1.3 Motivação

A Competição Internacional de Planejamento (IPC) é um evento bianual, realizado em paralelo com a Conferência Internacional de Planejamento e Escalonamento Automatizado (ICAPS). Essa competição tem como objetivo definir o estado-da-arte em planejamento, avaliando e motivando a pesquisa existente, bem como identificando novas linhas de pesquisa para o futuro (PEREIRA, 2007).

No ano de 2016, realizou-se pela primeira vez no âmbito da IPC a *Competição Internacional de Planejamento para Problemas sem Solução* (UIPC-2016). Esta competição restringiu-se a analisar algoritmos capazes de detectar se um problema de planejamento possui ou não solução. Com relação aos objetivos da competição, o site da UIPC, define que:

> *"The aim of the unsolvability IPC is to test the ability of classical automated planners to detect when a planning task has no solution. The benchmarks used in the track will contain a mix of solvable and unsolvable instances, and points will only be awarded for correctly identifying the unsolvable instances (MUISE; LIPOVETZKY, 2016)".*

A UIPC-2016 contou com a participação de 14 (quatorze) algoritmos competidores, foram eles: *Aidos* (Aidos1, Aidos2 e Aidos3) (SEIPP et al., 2016), *SymPA* (TORRALBA, 2016), *SimDominance* (TORRALBA; HOFFMANN; KISSMANN, 2016), *M+S* (TORRALBA; HOFFMANN; KISSMANN, 2016), *DECS* (GNAD et al., 2016), *DE-PDB* (POMMERENING; SEIPP, 2016), *CLone* (STEINMETZ; HOFFMANN, 2016), *Django* (GNAD; STEINMETZ; HOFFMANN, 2016), *blind* (GNAD; STEINMETZ; HOFFMANN, 2016), *ReachLunch* (BALYO; SUDA, 2016), h^{++} (HASLUM, 2016) e *iProverPlan* (KOROVIN; SUDA, 2016). Para a competição foram disponibilizados 15 (quinze) domínios *benchmarks*: *bottleneck*, *pegsol*, *over-tpp*, *over-nomystery*, *over-rovers*, *sliding-tile*s, *doc-transfer*, *tetris*, *diagnosis*, *bag-transport*, *chess-pebbling*, *bag-gripper*, c, *pegsol-row5* e *cave-diving*.

A Figura 5 mostra a tabela de resultados da UIPC-2016. Nas linhas estão dispostos os algoritmos participantes e nas colunas os nomes dos domínios *benchmarks*. Na *escala de cinza* da tabela, quanto mais escuro for a célula, maior a quantidade de problemas que o algoritmo conseguiu detectar corretamente como sem solução para um dado domínio. Observe ainda que para certos domínios tais como *bottleneck* e *pegsol* a maioria dos planejadores obteve bom desempenho em detectar se um problema possui solução. No entanto, para outros domínios mais difíceis tais como *pegsol-row5* e *cave-diving* nenhum algoritmo foi capaz de detectar todos os problemas sem solução destes domínios. O vencedor da competição foi o algoritmo *Aidos*, um

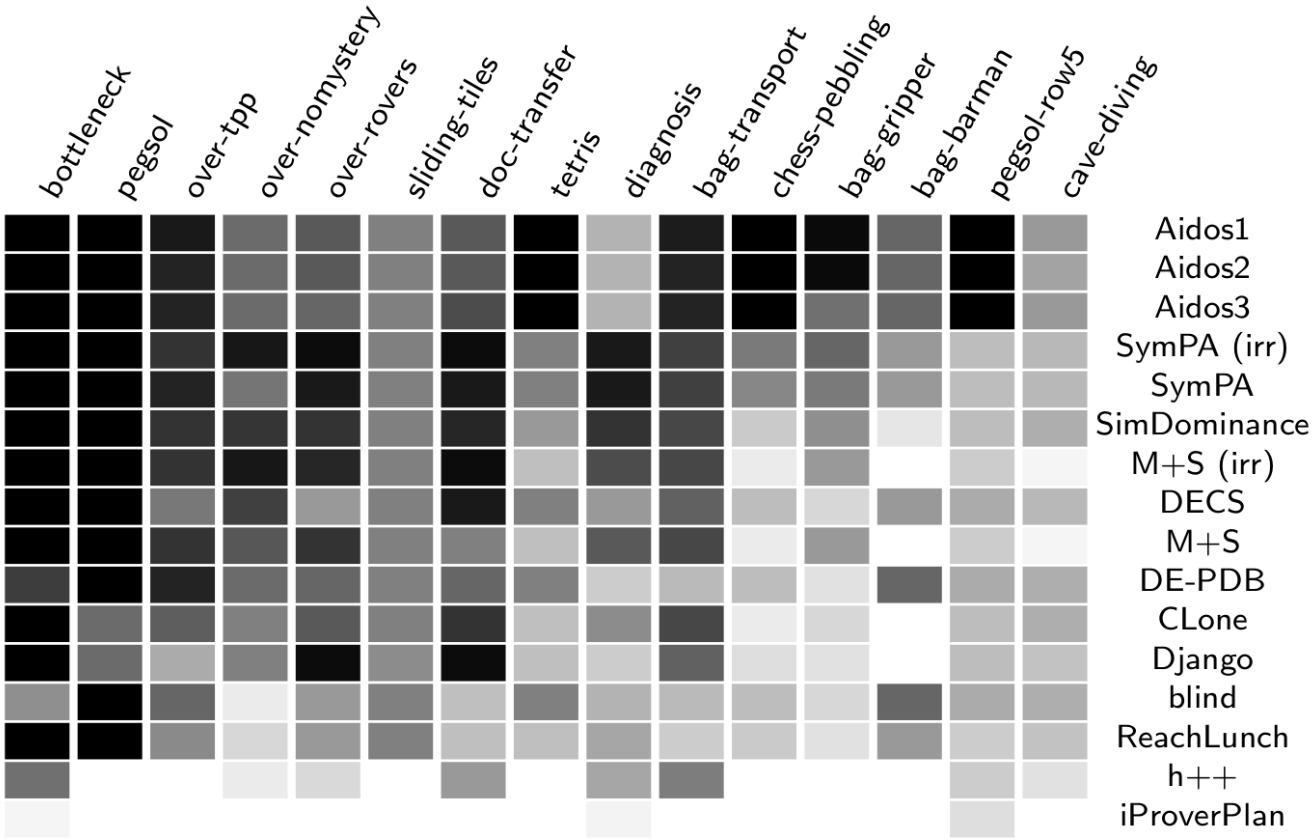


Figura 5 – Desempenho dos algoritmos participantes da UIPC-2016 na tarefa de identificar corretamente se um problema de planejamento não possui solução (MUISE; LIPOVETZKY, 2016).

algoritmo baseado em busca heurística que detectou 262 dos 340 problemas sem solução nas limitações de tempo (30 minutos) e memória (8GB) da competição.

O presente trabalho consiste no estudo e avaliação de desempenho de um planejador baseado em verificação simbólica de modelos, proposto por (MENEZES, 2014), utilizando os domínios *benchmarks* da UIPC-2016. Este algoritmo foi inicialmente desenvolvido para detecção de problemas de planejamento sem solução e posterior sugestão de modificações a fim de tornar o problema solucionável. Tal algoritmo foi testado utilizando três domínios: dois domínios provenientes da IPC, o *domínio de logística* (BACCHUS, 2001) e o *domínio do robô de marte* (DIMOPOULOS et al., 2006) e; o *domínio das chaves* (GÖBELBECKER et al., 2010) descrito anteriormente. Todos os problemas destes domínios são instâncias originalmente solucionáveis. Estas instâncias foram artificialmente modificadas com a introdução de erros para que os problemas passassem a não ter solução. Assim, nenhuma avaliação do algoritmo foi feita com problemas *benchmarks* elaborados com o principal foco de avaliar a capacidade de detecção de problemas sem solução, tais como os domínios e problemas elaborados para a realização da

UIPC-2016.

1.4 Objetivos

O objetivo principal deste trabalho é a avaliação do algoritmo de planejamento como verificação simbólica de modelos baseado na lógica α-CTL (MENEZES, 2014) para detecção de problemas de planejamento sem solução. A lógica α-CTL é uma lógica temporal cuja semântica é baseada em ações (PEREIRA, 2007). A representação do espaço de estados e a busca por uma solução é realizada de forma simbólica, utilizando estruturas conhecidas como *Diagramas de Decisão Binária* (BRYANT, 1986).

Como objetivos específicos deste trabalho, temos:

- Estudo do algoritmo de planejamento como verificação simbólica de modelos α-CTL, o qual envolve o estudo dos seguintes conceitos: lógica temporal α-CTL, diagramas de decisão binária, representação simbólica de estados, transições e ações, cálculos simbólicos envolvendo estados, transições e ações.
- Estudo dos algoritmos participantes da UIPC-2016.
- Estudo dos domínios e problemas de planejamento propostos na UIPC-2016.
- Análise do desempenho do algoritmo de verificação simbólica de modelos α-CTL na detecção de problemas sem solução.
- Análise do desempenho do algoritmo de verificação simbólica de modelos α-CTL seguindo as regras da UIPC-2016, incluindo os limites de tempo e memória e possíveis retornos de falso-positivos.

1.5 Contribuições

As principais contribuições do presente trabalho são:

- Elaboração de comparativo das estratégias utilizadas pelos algoritmos participantes da UIPC-2016 para a detecção de problemas de planejamento sem solução.
- Avaliação de desempenho do planejador baseado em verificação simbólica de modelos com os domínios *benchmarks* da UIPC-2016.

1.6 Organização

Este trabalho está organizado conforme segue: o Capítulo 2 apresenta um resumo sobre os conceitos teóricos envolvidos na implementação do algoritmo de planejamento como verificação simbólica de modelos; o Capítulo 3 contém um estudo sobre os algoritmos participantes da UIPC-2016; o Capítulo 4 contém os resultados obtidos na análise experimental. Por fim, no Capítulo 5 mostramos as conclusões e discorremos sobre trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos teóricos envolvidos na construção do algoritmo de planejamento baseado em verificação simbólica de modelos α-CTL que foi avaliado neste trabalho.

2.1 Verificação de Modelos CTL

A *verificação de modelos* (CLARKE; GRUMBERG; PELED, 1999) é um método formal utilizado para determinar se um modelo de um sistema $\mathcal{M}$ satisfaz ou não uma dada propriedade φ. Na Figura 6 podemos observar o esquema de um verificador de modelos que recebe como entrada o modelo $\mathcal{M}$ e a propriedade φ e devolve *sucesso* se o modelo $\mathcal{M}$ satisfaz a propriedade φ ($\mathcal{M} \vDash \varphi$), ou um *contra exemplo* se o modelo $\mathcal{M}$ não satisfaz a propriedade φ ($\mathcal{M} \nvDash \varphi$).

Figura 6 – *Verificador de Modelos* (PEREIRA, 2007).

A seguir, apresentamos os conceitos de estrutura de Kripke, utilizada para representar o modelo do sistema, e de lógica temporal CTL, utilizada para especificar as propriedades que serão verificadas.

2.1.1 Estrutura de Kripke

Definição 1 (Estrutura de Kripke)**.** Seja $\mathbb{P} \neq \varnothing$ um conjunto de átomos proposicionais. Uma estrutura de Kripke (KRIPKE, 1963) $\mathcal{M}$ sobre $\mathbb{P}$ é uma tupla $< \mathcal{S}, \mathcal{L}, \mathcal{T} >$, em que (PEREIRA, 2007):

- $\mathcal{S}$ é um conjunto de estados;
- $\mathcal{T} : S \times S$ é uma relação de transição de estados;
- $\mathcal{L} : S \mapsto 2^{\mathbb{P}}$ é uma função de rotulação de estados. Cada estado $s \in S$ é rotulado pelo conjunto das proposições que são verdadeiras no estado (suposição do mundo fechado - *closed world assumption* (RUSSELL STUART, 2009)).

Exemplo 3 (Estrutura de Kripke)**.** A Figura 7 ilustra uma estrutura de Kripke $\mathcal{M} =< \mathcal{S}, \mathcal{L}, \mathcal{T} >$ sobre o conjunto de proposições $\mathbb{P} = \{p, q\}$. Nesta estrutura, temos que:

- $\mathcal{S} = \{s_0, s_1, s_2\}$;
- $\mathcal{T} = \{(s_0, s_1), (s_0, s_2), (s_1, s_2), (s_2, s_2)\}$ e;
- $\mathcal{L}(s_0) = \{p, q\}$, $\mathcal{L}(s_1) = \{p\}$ e $\mathcal{L}(s_2) = \{q\}$.

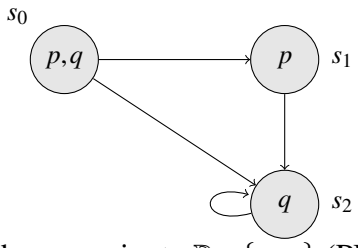


Figura 7 – Estrutura de Kripke sobre o conjunto $\mathbb{P} = \{p, q\}$ (PEREIRA, 2007).

É possível desdobrar a estrutura de Kripke para obter uma árvore de computação infinita. Esta árvore é enraizada no estado inicial da estrutura e cada ramo da árvore representa um possível caminho de computação. Ao percorrer um caminho nessa estrutura e chegar em um estado sem saída, este estado é replicado na árvore de computação, tornando-a infinita.

2.1.2 A lógica temporal CTL

A lógica CTL (Computation Tree Logic) (CLARKE; EMERSON, 1982) é uma lógica temporal que permite especificar propriedades quantificadas sobre caminhos em uma árvore de computação. Esta lógica considera que, para cada instante de tempo, pode haver vários futuros possíveis.

Os operadores temporais da lógica CTL são: $\circ$ (*próximo*), $\Diamond$ (*finalmente*), $\Box$ (*globalmente*) e $\sqcup$ (*até que*). Cada operador temporal deve ser precedido de um quantificador de caminho: $\exists$: existe um caminho ou $\forall$ para todos os caminhos.

Definição 2. (Sintaxe da Lógica CTL) Seja $\mathbb{P}$ um conjunto de átomos proposicionais, a sintaxe da lógica CTL é definida como a seguir (PEREIRA, 2007):
$\varphi \doteq p \in \mathbb{P} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \circ \varphi_1 \mid \forall \circ \varphi_1 \mid \exists \Box \varphi_1 \mid \forall \Box \varphi_1 \mid \exists \Diamond \varphi_1 \mid \forall \Diamond \varphi_1 \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall[\varphi_1 \sqcup \varphi_2]$

Na Figura 8 apresentamos a semântica dos operadores temporais da lógica CTL. Sendo s o estado inicial, tem-se que: em (a) para algum próximo estado p é verdadeira; em

(b) para algum caminho a partir de s, p é verdadeira para todos os estados; em (c) para algum caminho, p é finalmente verdadeira; em (d) para algum caminho, p é verdadeira até que q seja verdadeira; em (e) para todos os próximos estados p é verdadeira; em (f) para todos os caminhos, partindo de s, p é verdadeira; em (g) p é finalmente verdadeira para todos os caminhos; e em (h), para todos os caminhos p é verdadeira até que q seja verdadeira (MENEZES, 2014).

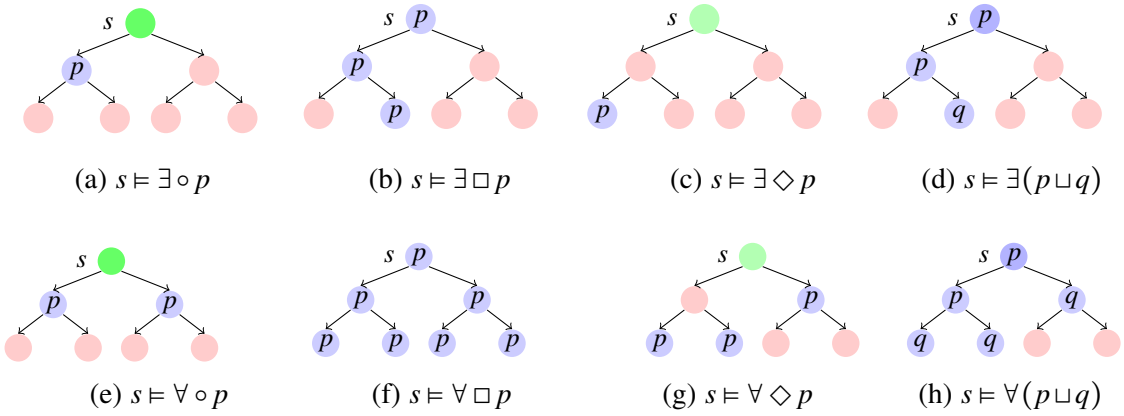


Figura 8 – Semântica dos operadores temporais da lógica CTL (PEREIRA, 2007).

Definição 3. (Semântica da Lógica CTL) Seja $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{T})$ uma estrutura de Kripke, $s \in \mathcal{S}$ e φ uma fórmula na lógica CTL. A relação $(\mathcal{M}, s) \vDash \varphi$ é definida por indução estrutural em φ, como a seguir [adaptado de (HUTH; RYAN, 2008)]:

- $(\mathcal{M}, s) \vDash \top$ e $(\mathcal{M}, s) \nvDash \bot$
- $(\mathcal{M}, s) \vDash p$ se e somente se $p \in \mathcal{L}(s)$
- $(\mathcal{M}, s) \vDash \neg\varphi$ se e somente se $(\mathcal{M}, s) \nvDash \varphi$
- $(\mathcal{M}, s) \vDash \varphi_1 \wedge \varphi_2$ se e somente se $(\mathcal{M}, s) \vDash \varphi_1$ e $(\mathcal{M}, s) \vDash \varphi_2$
- $(\mathcal{M}, s) \vDash \varphi_1 \vee \varphi_2$ se e somente se $(\mathcal{M}, s) \vDash \varphi_1$ ou $(\mathcal{M}, s) \vDash \varphi_2$
- $(\mathcal{M}, s) \vDash \varphi_1 \rightarrow \varphi_2$ se e somente se $(\mathcal{M}, s) \nvDash \varphi_1$ ou $(\mathcal{M}, s) \vDash \varphi_2$
- $(\mathcal{M}, s) \vDash \forall \circ \varphi$ se e somente se para todo s_1 tal que $(s, s_1) \in T$, temos $(\mathcal{M}, s) \vDash \varphi$.
- $(\mathcal{M}, s) \vDash \exists \circ \varphi$ se e somente se para algum s_1 tal que $(s, s_1) \in T$, temos $(\mathcal{M}, s_1) \vDash \varphi$.
- $(\mathcal{M}, s) \vDash \forall \Box \varphi$ é verdade se e somente se para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para todos os s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \vDash \varphi$.
- $(\mathcal{M}, s) \vDash \exists \Box \varphi$ é verdade se e somente se existe um caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para todos os s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \vDash \varphi$.
- $(\mathcal{M}, s) \vDash \forall \Diamond \varphi$ é verdade se e somente se para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde

$s_1 = s$, existe algum s_i ao longo deste caminho, tal que $(\mathcal{M}, s_i) \vDash \varphi$.

- $(\mathcal{M}, s) \vDash \exists \Diamond \varphi$ é verdade se e somente se existe um caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para algum s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \vDash \varphi$.
- $(\mathcal{M}, s) \vDash \forall [\varphi_1 \sqcup \varphi_2]$ é verdade se e somente se para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, este caminho satisfaz $\varphi_1 \sqcup \varphi_2$, isto é, existe algum s_i ao longo deste caminho tal que $(\mathcal{M}, s_i) \vDash \varphi_2$ e, para cada $j < i$, temos $(\mathcal{M}, s_j) \vDash \varphi_1$.
- $(\mathcal{M}, s) \vDash \exists [\varphi_1 \sqcup \varphi_2]$ é verdade se e somente se existe algum caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e este caminho satisfaz $\varphi_1 \sqcup \varphi_2$.

2.2 Verificação Simbólica de Modelos

A *verificação simbólica de modelos* é uma abordagem amplamente utilizada para controlar o problema da explosão do espaço de estados, uma vez que o algoritmo de verificação possui um número de estados exponencial no número de proposições utilizadas para descrever o sistema (HUTH; RYAN, 2008).

Assim, os estados e transições da estrutura de Kripke não são representados de forma individual, mas sim como conjuntos de estados, utilizando estruturas de dados eficientes denominadas *Diagramas de Decisão Binária* (BDDs) (BRYANT, 1986). Tais estruturas possuem a capacidade de representar conjunto de estados com aproximadamente 10^{20} estados (BURCH et al., 1992). Desta forma conjuntos de estados e transições são representados como fórmulas da lógica proposicional para que, em seguida, sejam representados por meio de BDDs.

2.2.1 Representação de estados e transições

A seguir, veremos como codificar estados, conjuntos de estados e transições de uma estrutura de Kripke como fórmulas da lógica proposicional.

Definição 4. (Representação proposicional para um estado) Seja $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ uma estrutura de Kripke sobre um conjunto de proposições $\mathbb{P}$. Um estado $s \in \mathcal{S}$ pode ser denotado segundo a fórmula proposicional $\xi(s)$, em que:

$$\xi(s) = \bigwedge_{p \in \mathcal{L}(s)} p \quad \wedge \quad \bigwedge_{p \in \mathbb{P} - \mathcal{L}(s)} \neg p, \tag{2.1}$$

Definição 5. (Representação proposicional para conjunto de estados) Seja $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ uma estrutura de Kripke sobre um conjunto de proposições $\mathbb{P}$. Um conjunto de estados $X \subseteq \mathcal{S}$ pode ser denotado segundo a fórmula proposicional $\xi(X)$, em que:

$$\xi(X) = \bigvee_{s \in X} \xi(s). \tag{2.2}$$

O ponto-chave que torna esta representação interessante é que o BDD que representa um conjunto de estados pode ser bem pequeno (HUTH; RYAN, 2008).

A estrutura de Kripke é uma representação da dinâmica do ambiente, desta forma as proposições podem mudar seu valor verdade conforme ocorrem transições entre os estados. Assim, em um estado s, por exemplo, podemos ter que a proposição p é verdadeira, isto é, $p \in \mathcal{L}(s)$ e em um estado posterior s' podemos ter que o valor de p já não seja mais verdade, isto é, $p \notin \mathcal{L}(s')$.

Desta forma, na codificação das transições é preciso considerar uma cópia do conjunto de proposições $\mathbb{P}$, o qual será denominado $\mathbb{P}'$. Assim, cada $p' \in \mathbb{P}'$ corresponderá a uma variável $p \in \mathbb{P}$, mas denotará propriedades que valem em um estado posterior a execução da transição.

Definição 6. (Representação proposicional para uma transição) Seja $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ uma estrutura de Kripke sobre um conjunto de proposições $\mathbb{P}$ e seja $\mathbb{P}$' uma cópia do conjunto $\mathbb{P}$. Um transição $(s, s') \in \mathcal{T}$ pode ser denotada por meio de uma fórmula proposicional $\xi((s, s'))$ em que:

$$\xi((s, s')) = \xi(s) \wedge \xi(s') \tag{2.3}$$

Definição 7. (Representação proposicional para a relação de transição) Seja $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ uma estrutura de Kripke sobre um conjunto de proposições $\mathbb{P}$ e seja $\mathbb{P}$' uma cópia do conjunto $\mathbb{P}$. A relação de transição $\mathcal{T}$ pode ser denotada por meio da fórmula proposicional $\xi(T)$, em que:

$$\xi(\mathcal{T}) = \bigvee_{(s,s') \in T} \xi((s, s')). \tag{2.4}$$

Exemplo 4. (Codificação proposicional da estrutura de Kripke) Considere a estrutura de Kripke, mostrada na Figura 7. O conjunto de estados $S = \{s_0, s_1, s_2\}$ pode ser representado pela fórmula:

$$\xi(S) = (p \wedge \neg q) \vee (p \wedge q) \vee (\neg p \wedge q).$$

e a relação de transição $\mathcal{T}$ pode ser representada como:

$$\xi(\mathcal{T}) = (p \wedge q \wedge p' \wedge \neg q') \vee (p \wedge q \wedge \neg p' \wedge q') \vee (p \wedge \neg q \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge \neg p' \wedge q').$$

2.2.2 Diagramas de decisão binária

Diagramas de Decisão Binária (BDDs) (BRYANT, 1986) são estruturas de dados utilizadas para representar uma fórmula proposicional. Um BDD é um grafo acíclico, similar a uma árvore de decisão binária, em que os nós não terminais são rotulados com proposições e os nós terminais com 1 ou 0, indicando se o átomo proposicional é verdadeiro e falso, respectivamente.

Para permitir a representação compacta de uma fórmula são realizadas remoções de nós duplicados e remoção de testes redundantes (HUTH; RYAN, 2008), além de também fixar uma ordenação das proposições. Por este motivo tais diagramas são denominados *Diagramas de Decisão Binária Ordenados e Reduzidos*.

Exemplo 5. (Valoração de uma fórmula) Suponha a fórmula proposicional $p \vee q$. Todas as valorações (*modelos*) possíveis desta fórmula podem ser dispostas em uma tabela verdade como a Tabela 1.

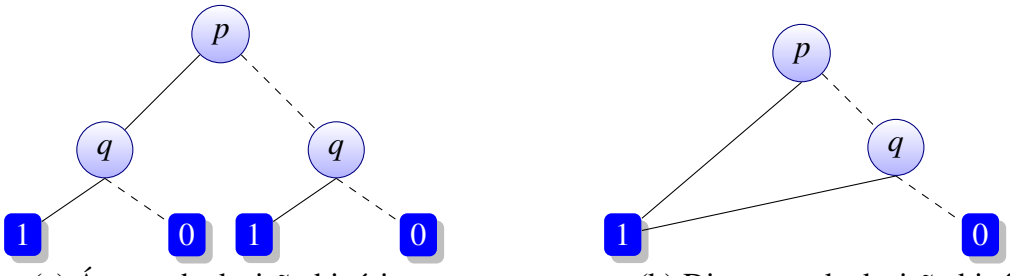
p	q	$p \vee q$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$

Tabela 1 – Tabela-verdade para a fórmula $p \vee q$

A Figura 9(a) ilustra uma árvore de decisão binária para a fórmula $p \vee q$. Observe que cada nó não terminal é uma variável proposicional e possui duas arestas saindo dele: a aresta sólida representa o valor verdadeiro e a aresta pontilhada representa o valor falso. A Figura 9(b) ilustra o BDD correspondente à árevore de decisão binária. Note que no caso de a valoração de p ser verdadeira (aresta sólida), a aresta vai diretamente para a folha 1, indicando que a fórmula $p \vee q$ é verdadeira. Se p for falso (aresta pontilhada), é necessário analisar o valor da proposição q. Se q for verdade, a fórmula $p \vee q$ é verdade (folha 1). Senão, a fórmula $p \vee q$ é falsa (folha 0).

2.3 Planejamento como Verificação de Modelos

O arcabouço de verificação de modelos pode ser utilizado para determinar se é possível obter um plano de ações para um problema de planejamento, uma área conhecida como



(a) Árvore de decisão binária (b) Diagrama de decisão binária

Figura 9 – (a) Árvore de decisão binária para a fórmula $p \vee q$. (b) BDD para a fórmula $p \vee q$.

planejamento baseado em verificação de modelos (EDELKAMP; HELMERT, 2001; CIMATTI et al., 2003; GIUNCHIGLIA; TRAVERSO, 2000; PEREIRA; BARROS, 2008)

A Figura 10 ilustra o esquema do *planejamento baseado em verificação de modelos*. O algoritmo recebe como entrada um domínio de planejamento $\mathcal{D}$, que representa a dinâmica do ambiente, um estado inicial do ambiente s_0 e *a meta de planejamento* φ. O *planejador* devolve "Sim, se $(\mathcal{M}, s_0) \models \varphi$, caso contrário, "Não"é retornado.

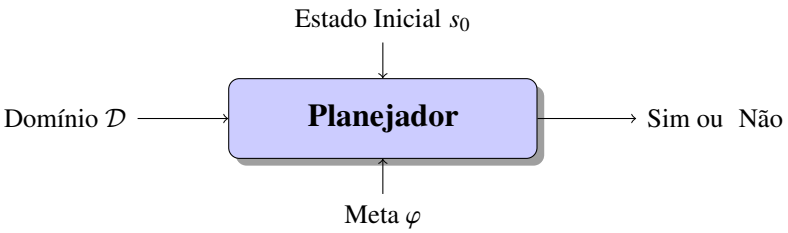


Figura 10 – *Planejamento baseado em Verificação de Modelos* (PEREIRA, 2007).

A diferença significativa entre verificadores e planejadores é que enquanto um verificador fornece como saída todos os estados do modelo que satisfazem uma dada propriedade, um planejador preocupa-se apenas em garantir que a propriedade seja satisfeita a partir de um dado estado inicial.

O algoritmo de planejamento como verificação de modelos realiza uma busca regressiva a partir do conjunto $\mathbb{X}$ de todos os estados que satisfazem a meta de planejamento φ. Em cada passo, computa o conjunto de estados a partir dos quais é possível alcançar um estado em $\mathbb{X}$ pela execução de uma ação e adiciona tais estados no conjunto $\mathbb{X}$. O algoritmo para quando nenhum novo estado é adicionado no conjunto $\mathbb{X}$.

2.3.1 Sistema de Transição de Estados Rotulado por Ações

Seja $\mathbb{P} \neq \varnothing$ um conjunto de proposições e $\mathbb{A} \neq \varnothing$ um conjunto de ações, um *sistema de transição de estados rotulados por ações* $\mathcal{D} =< \mathcal{S}, \mathcal{L}, \mathcal{T} >$ é uma estrutura, similar à estrutura

de Kripke, cujas transições são rotuladas por ações $a \in \mathbb{A}$.

Exemplo 6 (Sistema de Transição de Estados Rotulado por Ações)**.** A Figura 7 ilustra um Sistema de Transição de Estados Rotulado por Ações sobre o conjunto de proposições $\mathbb{P} = \{p, q\}$ e o conjunto de ações $\mathbb{A} = \{a, b, c\}$. Note que este sistema é similar à Estrura de Kripke da Figura 7, no entanto, possui as transições rotuladas com as ações a, b e c.

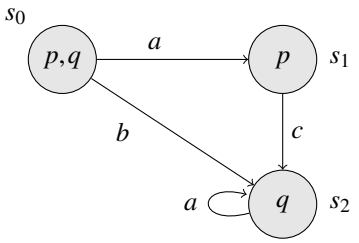


Figura 11 – Sistema de Transição de Estados Rotulado por Ações sobre o conjunto $\mathbb{P} = \{p, q\}$ e o conjunto de ações $\mathbb{A} = \{a, b, c\}$.

2.3.2 Lógica Temporal α-CTL

A lógica CTL têm sido amplamente utilizada em planejadores baseados em verificação de modelos. Porém, como pode ser observado na Figura 8, a estrutura de Kripke não considera no rótulo das transições as ações que causam transições entre estados. Assim, os planejadores baseados nessa lógica necessitam implementar uma solução extra-lógica para ter controle sobre a escolha das ações.

Para resolver a limitação da lógica CTL, foi proposta a lógica α-CTL (PEREIRA, 2007; PEREIRA; BARROS, 2008), uma extensão da CTL que possui a semântica baseada em ações, sendo capaz de distinguir as ações que causam transições entre estados. Os operados da α-CTL diferenciam-se dos operadas da lógica CTL por serem "pontuados", ou seja, $\odot$ (próximo estado), $\boxdot$ (globalmente), $\diamondsuit$ (finalmente) e $\sqcup$ (até que). Assim como na CTL, todos os operadores temporais devem ser precedidos de um quantificador de caminhos ($\exists$ ou $\forall$).

Como podemos perceber, o significado de cada operador e a forma como as fórmulas devem ser construídas permanecem iguais, porém foi adicionado o fato de que para um estado s satisfazer uma fórmula φ uma ação deve acontecer. Por exemplo, um estado s satisfaz uma fórmula $\forall \odot \varphi$ se *existe* uma ação $\alpha \in \mathbb{A}$ que, quando executada em s, *obrigatoriamente* alcança um estado sucessor de s que satisfaz a fórmula φ. A sintaxe da α-CTL é semelhante a da CTL,

Definição 8. (Sintaxe da Lógica α-CTL) Seja onde $\mathbb{P}$ um conjunto de proposições atômicas, a sintaxe da lógica α-CTL é definida como a seguir (PEREIRA, 2007):
$\varphi ::= p | \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \odot \varphi_1 \mid \forall \odot \varphi_1 \mid \exists \boxdot \varphi_1 \mid \forall \boxdot \varphi_1 \mid \exists \diamondsuit \varphi_1 \mid \forall \diamondsuit \varphi_1 \mid \exists (\varphi_1 \sqcup \varphi_2) \mid \forall (\varphi_1 \sqcup \varphi_2)$

A Figura 12 ilustra a semântica dos operadores temporais da lógica α - CTL. Observe que diferentemente da Figura 8, as transições entre estados são rotuladas com as ações a, b, c e d. Assim, sendo s um estado inicial, tem-se que: em (a), existe um próximo estado tal que *para alguma ação* p é verdade; em (b), existe um caminho a partir de s onde *para alguma ação*, p é verdade para todos os estados; em (c), existe um caminho tal que, em algum estado futuro, para alguma ação, p é verdade; em (d), existe algum estado futuro, *para alguma ação*, em que p é verdade até que q seja verdade ; em (e) para *todos os próximos estados*, *para alguma ação*, p é verdadeira (neste caso, executando-se a ação a); em (f) para todos os caminhos a partir de s, *para alguma ação*, p é verdade; em (g) para todos os caminhos a partir de s, *para alguma ação*, p é verdade para todos os estados futuros; em (h) para todos os estados do caminho a partir de s, *para alguma ação*, p é verdade até que q seja verdade.

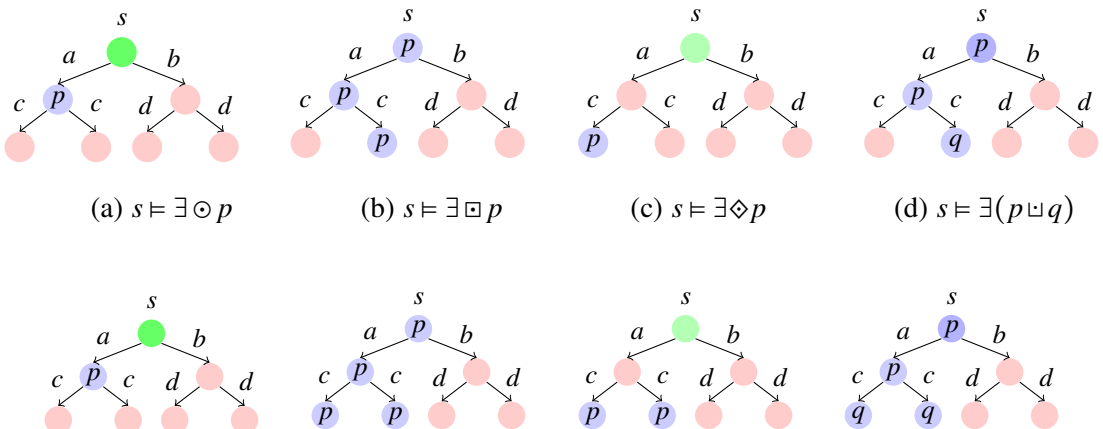


Figura 12 – Semântica dos operadores temporais da lógica α - CTL (MENEZES, 2014).

Definição 9. (Semântica da Lógica α-CTL) Sejam $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ um sistemas de transição de estados em que os estados são rotulados por um conjunto de átomos proposicionais $\mathbb{P}$ e as transições são rotuladas por um conjunto de ações $\mathbb{A}$. A semântica da lógica α-CTL é definida como a seguir [adaptado de (PEREIRA, 2007)]:

- $(\mathcal{M}, s) \vDash \top$ e $(\mathcal{M}, s) \nvDash \bot$

- $(\mathcal{M}, s) \models p$ se e somente se $p \in \mathcal{L}(s)$
- $(\mathcal{M}, s) \models \neg\varphi$ se e somente se $(\mathcal{M}, s) \not\models \varphi$
- $(\mathcal{M}, s) \models \varphi_1 \wedge \varphi_2$ se e somente se $(\mathcal{M}, s) \models \varphi_1$ e $(\mathcal{M}, s) \models \varphi_2$
- $(\mathcal{M}, s) \models \varphi_1 \vee \varphi_2$ se e somente se $(\mathcal{M}, s) \models \varphi_1$ ou $(\mathcal{M}, s) \models \varphi_2$
- $(\mathcal{M}, s) \models \varphi_1 \rightarrow \varphi_2$ se e somente se $(\mathcal{M}, s) \not\models \varphi_1$ ou $(\mathcal{M}, s) \models \varphi_2$
- $(\mathcal{M}, s) \models \forall \odot \varphi$ se e somente se para todo s_1 existe $a \in \mathbb{A}$ tal que $(s, a, s_1) \in T$ e $(\mathcal{M}, s_1) \models \varphi$.
- $(\mathcal{M}, s) \models \exists \odot \varphi$ se e somente se para algum s_1 existe $a \in \mathbb{A}$ tal que $(s, a, s_1) \in T$ e $(\mathcal{M}, s_1) \models \varphi$.
- $(\mathcal{M}, s) \models \forall \boxdot \varphi$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para todos os s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \models \varphi$.
- $(\mathcal{M}, s) \models \exists \boxdot \varphi$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que para algum caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para todos os s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \models \varphi$.
- $(\mathcal{M}, s) \models \forall \diamondsuit \varphi$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, existe algum s_i ao longo deste caminho, tal que $(\mathcal{M}, s_i) \models \varphi$.
- $(\mathcal{M}, s) \models \exists \diamondsuit \varphi$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que algum caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e para algum s_i ao longo deste caminho, temos $(\mathcal{M}, s_i) \models \varphi$.
- $(\mathcal{M}, s) \models \forall [\varphi_1 \sqcup \varphi_2]$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que para todos os caminhos $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, este caminho satisfaz $\varphi_1 \sqcup \varphi_2$, isto é, existe algum s_i ao longo deste caminho tal que $(\mathcal{M}, s_i) \models \varphi_2$ e, para cada $j < i$, temos $(\mathcal{M}, s_j) \models \varphi_1$.
- $(\mathcal{M}, s) \models \exists [\varphi_1 \sqcup \varphi_2]$ é verdade se e somente se existe uma ação $a \in \mathbb{A}$ tal que para algum caminho $(s_1, s_2, s_3, \ldots)$, onde $s_1 = s$, e este caminho satisfaz $\varphi_1 \sqcup \varphi_2$.

2.3.3 O Algoritmo de Planejamento como Verificação de Modelos

O Algoritmo 1 mostra o pseudocódigo do verificador de existência de plano baseado em verificação de modelos, o qual denominamos aqui de MC-PlanEX. Este algoritmo é uma adaptação do planejador como verificação de modelos para a tarefa de decidir se um problema de planejamento possui ou não solução. Este algoritmo recebe como entrada um problema de planejamento dado por: (i) um conjunto de ações $\mathbb{A}$ em termos de suas pré-condições e efeitos, que representa implicitamente o domínio de planejamento; (ii) um estado inicial s_0 e a meta de planejamento φ. Para domínios de planejamento clássico, como os que serão avaliados neste trabalho, a propriedade a ser verificada em lógica temporal é se "*existe algum caminho que por*

alguma ação a meta seja alcançada.", $\exists\Diamond\varphi$, em que φ é a meta de planejamento.

Inicialmente, duas variáveis X e Y são declaradas (linha 02). Tais variáveis serão utilizadas para armazenar estados do domínio de planejamento. Na linha 03, a variável X é inicializada com $\varnothing$ e a variável Y é inicializada com o conjunto de estados que satisfazem a meta φ, os chamados estados meta (linha 04). No laço das linhas 05-11 é realizada uma busca regressiva, em que os estados de Y são armazenados em X (linha 06) e o conjunto Y é acrescido de todos os estados antecessores a algum estado em Y (operação de *regressão*). Se em algum momento o estado inicial for alcançado (linha 08), o problema é solucionável e o algoritmo retorna 1 (linha 09). Caso contrário, um ponto-fixo será alcançado (caracterização de ponto-fixo da lógica α-CTL) e o algoritmo retorna 0, indicando que não é possível obter um plano para tal problema.

: Algoritmo de Planejamento como Verificação de Modelos

```
1:  MC-PlanEX (𝔸, s_0, φ)
2:  local var X, Y ;
3:  X ← ∅ ;
4:  Y ← SAT(φ);
5:  while X = Y do
6:      X ← Y;
7:      Y ← Y∨ regressao(𝔸, Y);
8:      if s_0 ∈ Y then
9:          return 1; // O problema possui solução.
10: return 0; // O problema não possui solução.
```

A função *regressao* da linha 7, recebe como parâmetro um conjunto de ações $\mathbb{A}$, um conjunto de estados $\mathbb{Y}$ e calcula o conjunto de estados predecessores baseado na semântica da lógica α-CTL.

2.4 Planejamento como Verificação Simbólica de Modelos

Na abordagem de planejamento como verificação simbólica de modelos (MENEZES, 2014), o conjunto de estados do domínio de planejamento é representado como fórmulas das lógica proposicional, as quais são codificadas como *Diagramas de Decisão Binária*. Além disso, cada ação do domínio de planejamento, em termos de suas pré-condições e efeitos, são representadas como fórmulas da lógica proposicional e também são codificadas como *Diagramas*

de Decisão Binária.

O conjunto de estados antecessores é computado por meio de operações de regressão simbólica (MENEZES; BARROS; PEREIRA, 2014) baseada nas ações. A regressão é realizada por operações nos BDDs (operação exists), que corresponde ao quantificador existencial da lógica QBF (*Quantified Boolean Formulas* (STAALMARCK, 2003)), já usado nas abordagens de verificação simbólica de modelos tradicionais (GIUNCHIGLIA; TRAVERSO, 2000; CIMATTI et al., 2003).

3 TRABALHOS RELACIONADOS

Trabalhos na área de Planejamento em Inteligência Artificial, em sua minoria, abordam o problema de determinar se um domínio de planejamento não possui solução. Neste capítulo iremos descrever os principais algoritmos participantes da UIPC 2016, capazes de decidir se um problema de planejamento possui ou não solução. Após realizar um estudo destes algoritmos foi possível dividi-los em três categorias: *algoritmos baseados em busca heurística* (Aidos (Seipp et al. (2016)), h++ (Haslum (2016)), *CLone* (Steinmetz e Hoffmann (2016)) e *DE-PDB* (Pommerening e Seipp (2016))), *algoritmos baseados em planejamento como satisfação booleana (SAT-PLAN Kautz, Selman et al. (1992))* (ReachLunch (Balyo e Suda (2016)), iProverPlan (Korovin e Suda (2016)), SimDominance (Torralba, Hoffmann e Kissmann (2016)), M+S (Torralba, Hoffmann e Kissmann (2016))) e *algoritmos baseados em representação simbólica do espaço de estados* (SymPA (Torralba (2016))).

O algoritmo Aidos foi o vencedor da UIPC 2016 e detectou corretamente como não solucionáveis 262 problemas de 340 problemas. O segundo colocado foi o algoritmo SymPa que detectou corretamente como não solucionáveis 232 problemas de 340 problemas.

A Tabela 2 resume as técnicas utilizadas por cada um dos algoritmos participantes da UIPC-2016 e o agrupamento destas técnicas em algoritmos heurísticos, algoritmos baseados em satisfação booleaana e algoritmos baseado em representação simbólica.

Tabela 2 – Categorização dos trabalhos relacionados

	Algoritmos	**Estratégia de Busca**	**Heurísticas Utilizadas**	**Outras Observações**
Algoritmos heurísticos	*Aidos*	Largura	h^{++}, h^{max} *e A**	Baseado no Fast Downward
	Clone	Profundidade	$h(c)$	Aprendizado de cláusulas SAT
	DE-PBD	Largura	detecção de dead-ends	Execução em duas etapas
	h^{++}	Nenhuma	h^{++}	Computa planos de custo mínimo para um problema relaxado
Algoritmos baseados em satisfação booleana	*ReachLunch*	Profundidade	Nenhuma	Execução em duas etapas
	IproverPlan	Nenhuma	Nenhuma	Utiliza um provador de teoremas
Algoritmos baseados em representação simbólica	*SymPa*	Simbólica, Bidirecional	abstração de perímetro	Utiliza BDDs

3.1 Algoritmos baseados em busca heurística

O trabalho proposto por Seipp et al. (2016) descreve o conjunto de algoritmos *Aidos* que possui três algoritmos, *Aidos 1, Aidos 2 e Aidos 3*. Todos os algoritmos foram implementados no sistema Fast Downward (Helmert (2006)). Na implementação foi utilizada, além de técnicas já implementadas no *Fast Downward* (HELMERT, 2006), outras técnicas desenvolvidas especialmente para auxiliar na tarefa de detectar se um problema de planejamento não possui solução.

O algoritmo *CLone* proposto por Steinmetz e Hoffmann (2016) realiza uma busca em profundidade no espaço de estados do problema. Para otimizar a busca, o algoritmo utiliza-se de uma heurística de caminho crítico para reduzir o número de estados visitados e identificar *dead-ends* (becos sem saída).

Pommerening e Seipp (2016) propõe o algoritmo *DE-PDB* (Fast Downward Dead-End Pattern Database). Para detectar se um problema de planejamento não possui solução, o algoritmo calcula padrões para uma coleção de padrões e, em seguida, usa esses tais padrões para podar estados durante a busca em profundidade.

O algoritmo proposto por Haslum (2016) é uma adaptação da heurística de computação de planos de custo mínimo h^{++} (HASLUM et al., 2012). O algoritmo computa um plano relaxado não ótimo em cada iteração. Deste modo, para detectar se um problema de planejamento não possui solução verifica se o problema relaxado não possui solução. Se o problema relaxado não possui solução, o problema original também não possui solução e vice-versa.

3.2 Algoritmos baseados em planejamento como satisfação booleana (SAT-PLAN)

Balyo e Suda (2016) propõe o algoritmo *Reachlunch* que executa uma busca de força bruta em profundidade no espaço de estados. Caso a busca não consiga detectar que o problema não possui solução, o problema é codificado como um problema de satisfação booleana. O problema é passado como entrada para um solucionador implementado com base no programa *Solis SAT Solver*.

O algoritmo proposto por Korovin e Suda (2016) codifica o problema de planejamento como um problema de satisfação booleana e usa uma extensão do IPROVER: um provador de teorema automatizado para lógica de primeira ordem, para detectar se um problema

de planejamento não possui solução.

3.3 Algoritmos baseados em representação simbólica do espaço de estados

O algoritmo *SymPa* proposto por Torralba (2016) realiza uma busca simbólica para a exploração do espaço de estados, utilizando Diagramas de Decisão Binária (BDDs) para representar e manipular conjuntos de estados. Nos casos em que é inviável prosseguir com a busca, o algoritmo é capaz de encontrar variáveis que podem ser removidas para que a busca prossiga.

4 ANÁLISE EXPERIMENTAL

Neste capítulo são apresentados os resultados obtidos a partir da execução do algoritmo de planejamento como verificação simbólica de modelos em quatro domínios da UIPC-2016. Os objetivos desta análise experimental são:

- (i) verificar o tempo de execução do planejador na tarefa de detectar se os problemas apresentados possuem ou não solução. Nestes experimentos, o tempo limite foi de 8 horas e a quantidade de memória foi limitada em 16GB.
- (ii) avaliar o desempenho do planejador em relação aos planejadores participantes da UIPC comparando a quantidade de problemas corretamente classificados como não solucionáveis. Nesta análise foram utilizados os limites de tempo e memória estabelecidos na UIPC que são, respectivamente, 30 minutos e 8GB de memória.

Para a realização dos experimentos foi utilizado um computador com processador Intel Core i7, 16.0 GB de memória RAM e sistema operacional Linux 14.04 LTS de 64 bits.

4.1 Descrição dos domínios

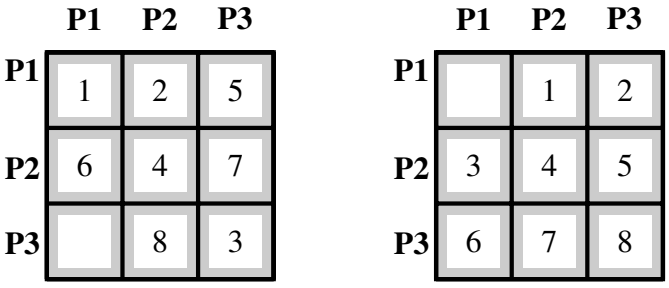
Nesta seção, apresentamos os domínios *sliding-tiles*, *pegsol*, *pegsol-row5* e *bottleneck* que foram utilizados como *benchmarks* na UIPC-2016. Os domínios foram escolhidos com base na interpretação realizada a partir da Figura 5, apresentada no capítulo 1. Assim, foram selecionados dois domínios considerados fáceis (*bottleneck* e *pegsol*), um domínio intermediário (*sliding-tiles*) e um domínio considerado difícil (*pegsol-row5*).

4.1.1 Domínio Sliding-Tiles

O domínio *Sliding tiles* é baseado no jogo *8-Puzzle* (Quebra-cabeça de 8 peças) que faz parte da família de quebra-cabeças deslizantes. O jogo é representado por um tabuleiro com nove quadros (tabuleiro 3 *x* 3), contendo oito peças numeradas e um quadrado vazio (posição em branco). É possível realizar movimentos com uma determinada peça se, e somente se, esta peça é adjacente ao quadrado vazio. Neste caso, são permitidos movimentos para cima, para a esquerda, para a direita e para baixo.

Dada uma possível configuração inicial, deseja-se conhecer qual a sequência de movimentos (ações) para chegar em uma configuração meta. A Figura 13 apresenta uma descrição de um possível estado inicial e de uma possível meta para o domínio do quebra-cabeça

de oito peças.



(a) Estado Inicial (b) Estado Meta

Figura 13 – Representação de um problema do domínio *Sliding-Tiles* em um tabuleiro 4x3

A Figura 23 do Apêndice **??** apresenta a definição do domínio *Sliding-Tiles* na linguagem *PDDL* (*Planning Domain Definition Language*). Os predicados do domínio são: *(tile ?x)*, especificando que ?x é um quadro; *(xposition ?x)*, especificando que ?x é uma posição horizontal do tabuleiro; *(yposition ?y)*, especificando que y é uma posição vertical do tabuleiro; *(at ?t ?x ?y)*, especificando que existe um quadro ?t na posição (?x, ?y) do tabuleiro; *(blank ?x ?y)*, especificando que o quadrado branco se encontra na posição (?x, ?y) do tabuleiro; *(inc ?p ?pp)*, especificando que a partir da posição ?p é possível mover-se para a posição ?pp; *(dec ?p ?pp)*, especificando que a partir da posição ?p é possível mover-se para a posição ?pp. As ações deste domínio são: *(move-up)*, que move um quadrado para uma posição para cima, *(move-down)*, que move um quadrado para uma posição para baixo, *(move-left)*, que move um quadrado uma posição para a esquerda e *(move-right)* que move um quadrado uma posição para a direita.

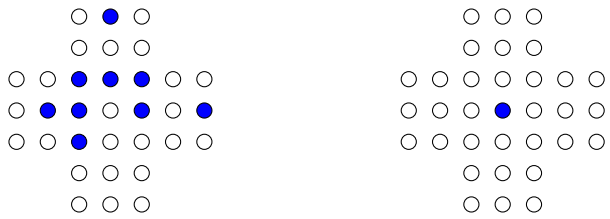
A Figura 24 do Apêndice **??** apresenta a descrição na linguagem *PDDL* de um problema no domínio *Sliding-Tiles*. Este é o problema ilustrado na Figura 13. Primeiramente são especificados o nome do problema e o domínio ao qual ele pertence. Logo após são definidos os objetos que instanciarão os predicados, sendo eles: *t1, t2, t3, t4, t5, t6, t7, t8, p1, p2 e p3*, representando os quadros (t) e as posições x e y (p). Em seguida, é especificado o estado inicial por meio de todos os predicados que são verdadeiros neste estado, conforme ilustrada na Figura 13(a). Para este problema a meta é definida conforme mostrado na Figura 13(b).

Para um mesmo domínio, há diversas instancias de problemas, sendo que estas podem variar no número de proposições, número de ações, na definição do estado inicial e da meta. Por exemplo, no problema *tiles*_1 (Figura 24) após utilizar os objetos do problema

para instanciar os predicados e operadores do domínio obtivemos, 113 proposições e 192 ações. No entanto, o domínio *Sliding-Tiles* possui uma mesma quantidade de proposições e ações para cada tabuleiro no qual os problemas são representados, ou seja, desde que o tamanho do tabuleiro não seja modificado, o número de predicados instanciados e de ações não variam entre os problemas desse domínio. Os problemas do domínio são representados em dois tabuleiros diferentes, alguns problemas seguem o padrão do quebra-cabeça de oito peças (tabuleiro $3x3$) e outros são representados por um tabuleiro $4x3$. Estes problemas possuem 242 predicados instanciados e 660 ações.

4.1.2 Domínio Pegsol

O domínio *Pegsol* foi baseado no jogo *Peg Solitaire*, o qual consiste em um quadro com formato de cruz que contém 33 buracos, onde as peças (*pegs*) são dispostas de modo a deixar pelo menos um buraco vazio, ou seja, uma posição em branco. Somente movimentos horizontais e verticais são permitidos e ao realizá-los as peças são retiradas gradativamente. A Figura 14 mostra uma configuração do jogo ilustrando como são distribuídas as peças no quadro. Um círculo preenchido indica que a localização está ocupada, ou seja, contém uma peça, e um círculo em branco indica que a posição está livre.



(a) Estado Inicial (b) Estado Meta

Figura 14 – Representação de um problema do domínio *Pegsol*.

A tarefa do domínio *Pegsol* é remover as peças até que uma configuração meta seja alcançada. Para remover uma peça é necessário realizar a ação *jump* (saltar), que é a única ação existente no domínio. Para a realização de um salto é necessário que a localização inicial esteja ocupada, que a localização de destino esteja livre, que exista uma peça entre elas e que estas estejam na mesma linha ou coluna.

A Figura 15 ilustra a ação *jump* (saltar) na vertical sobre uma peça. Deste modo, observe que a localização L_0 está ocupada pela peça que queremos "colocar"na localização L_2, a localização L_2 está livre e que existe uma peça na localização L_1. Assim, é possível realizar a

ação *jump* sobre a peça que está em L_1. O movimento similar pode ser realizado na horizontal.

Figura 15 – Representação da ação *jump* na vertical. A Figura *a* representa a configuração antes do salto e a Figura *b* representa a configuração depois do salto.

A Figura 19 do Apêndice **??** apresenta a definição do domínio *Pegsol* na linguagem *PDDL*. Os predicados deste domínio são: $(location\ ?obj)$, especificando que $?obj$ é uma localização; $(in\text{-}line\ ?x\ ?y\ ?z)$, especificando que os objetos $?x$, $?y$ e $?z$ estão na mesma linha ou coluna; $(occupied\ ?obj)$, especificando que a localização $?obj$ está ocupada e $(free\ ?obj)$, especificando que a localização $?obj$ está livre. O domínio possui somente a ação *jump*, ela recebe como parâmetro três localizações: *location ?from*, *location ?over* e *location ?to* , e realiza um "salto" da localização $?from$ (origem) sobre a localização $?over$ (peça) para a localização $?to$ (destino).

A Figura 20 do Apêndice **??** apresenta a descrição na linguagem *PDDL* do problema da Figura 14.

4.1.3 Domínio Pegsol-Row 5

O domínio *Pegsol-Row5* é uma variação do domínio *Pegsol*. Assim, tem-se um quadro onde por padrão, as últimas 5 linhas são livres e as demais ocupadas por peças. Os movimentos possíveis neste domínio são os mesmo do domínio *Pegsol*. A quantidade de linhas acima das linhas padrões é proporcional ao número de colunas, deste modo seja n o número de linhas de um problema neste domínio, o quadro possui tamanho $(n+5)\times n$. A Figura 16 ilustra a distribuição das peças no quadro para um problema do domínio *Pegsol-Row5*, onde tem-se n igual a 1 (problema 1).

O objetivo neste domínio é mover uma peça para a parte superior da linha que fica no meio do quadro usando as regras de movimentação do domínio *PegSol*.

A definição em *PDDL* do domínio *Pegsol-Row5* é mostrada na Figura 21 do Apêndice **??**. Pode-se observar que o mesmo é semelhante ao domínio *Pegsol*, assim possuem os mesmos predicados e as mesmas ações. No entanto, de acordo com as Figuras 14 e 16, e com base

(a) Estado Inicial (b) Estado Meta

Figura 16 – Representação de um problema do domínio *Pegsol-Row5* com n igual a 1.

em seus conceitos, é possível perceber que a disposição das peças no quadro difere entre os domínios.

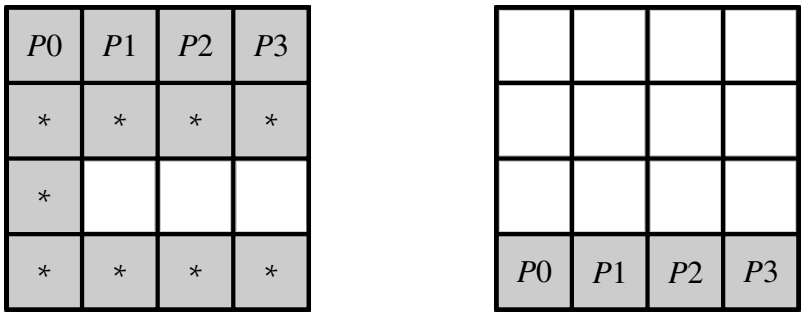
Um exemplo de problema de planejamento nesse domínio é apresentado na Figura 22. Note que, primeiramente é especificado o nome do problema e em seguida o nome do domínio que ele pertence. Logo após são definidos os objetos que serão utilizados para instanciar os predicados, são eles: *pos-0-0 pos-0-1 pos-0-2 pos-0-3 pos-0-4 pos-0-5*. Posteriormente é especificado o estado inicial, o qual indica que todos os objetos são localizações e que estão na mesma linha, também especifica que as localizações *pos-0-0, pos-0-2, pos-0-3, pos-0-4 e pos-0-5* estão livres enquanto que a localização *pos-0-0* está ocupada. Por fim, é especificado a meta do problema, que consiste em mover uma peça até a localização *pos-0-5*.

4.1.4 Domínio Bottleneck

O domínio *Bottleneck* possui n pessoas dispostas em um conjunto de localizações representado por uma matriz nxn. Considera as pessoas individualmente. Assim, as pessoas possuem objetivos diferentes e devem mover-se pela grade de modo a alcançar sua localização meta. Somente algumas posições estão ativas, ou seja, não é possível que as pessoas passem através de todas as posições da matriz, visto que nem todas as posições estão conectadas. Após uma posição ser visitada ela torna-se intransitável para qualquer pessoa.

A tarefa do domínio *Bottleneck* é mover-se pela grade sem revisitar uma posição até alcançar uma localização meta. A Figura 17 representa um problema deste domínio, onde $P0$, $P1$, $P2$ e $P3$ são as pessoas, as posições que contém um "$*$" estão ativas e as posições em branco indicam que essas localizações são inacessíveis.

A definição em *PDDL* do domínio *Bottleneck* é mostrada na Figura 25 do Apêndice **??**. Esse domínio possui os seguintes predicados: $(person ?p)$, especificando que $?p$ é uma pessoa; $(location ?loc)$, especificando que $?loc$ é uma localização; $(active ?loc)$, indicando



(a) Estado Inicial (b) Estado Meta

Figura 17 – Representação de um problema do domínio *Bottleneck* contendo 4 pessoas e 13 localizações.

que a localização ?*loc* está ativa, ou seja, as pessoas podem passar por ela; $(connected\ ?loc1\ ?loc2)$, especificando que as localizações ?*loc*1 e ?*loc*2 estão conectadas e $(person-at\ ?p\ ?loc)$, indicando que a pessoa ?*p* encontra-se na posição ?*loc*. Possui somente a ação *move*, a qual possui três parâmetros: uma pessoa (?*p*), uma localização de origem (?*l*1) e uma localização de destino (?*l*2). Deste modo, a ação move uma pessoa de uma localização (origem) para outra (destino).

A Figura 26 do Apêndice **??** mostra a definição em *PDDL* do problema representado pela Figura 17.

4.2 Desempenho do Planejador como Verificação Simbólica de Modelos

Esta seção apresenta os resultados dos experimentos realizados executando-se o planejador como verificação simbólica de modelos recebendo como entrada os problemas referentes aos quatro domínios descritos anteriormente. Para esses resultados foi considerado um tempo limite de 8 horas.

As Tabelas 3, 4, 5 e 6 apresentam os resultados obtidos para os problemas dos domínios, respectivamente, *sliding-tiles*, *pegsol*, *pegsol-row5* e *bottleneck*. Cada linha da tabela representa o resultado de um problema e as colunas estão organizadas de forma a mostrar: o nome do problema; o número de predicados instanciados do problema; o número de ações; o número de camadas que o algoritmo computou na busca por uma solução; o tempo gasto para o algoritmo decidir se o problema é solucionável e; a classificação dada como saída (*solucionável*, *sem solução* ou *desconhecido*).

Problema	**# Predicados**	**# Ações**	**# Camadas**	**Tempo**	**Classificação**
tiles_1	113	192	73	0m25.815s	*sem solução*
tiles_2	113	192	73	0m25.846s	*sem solução*
tiles_3	113	192	73	0m25.075s	*sem solução*
tiles_4	113	192	73	0m25.401s	*sem solução*
tiles_5	113	192	73	0m20.416s	*sem solução*
tiles_6	113	192	73	0m32.468s	*sem solução*
tiles_7	113	192	73	0m24.600s	*sem solução*
tiles_8	113	192	73	0m25.632s	*sem solução*
tiles_9	113	192	73	0m30.497s	*sem solução*
tiles_10	113	192	73	0m31.802s	*sem solução*
tiles_11	242	660	27	40m39.949s	*sem solução*
tiles_12	242	660	27	37m43.956s	*sem solução*
tiles_13	242	660	27	38m03.399s	*sem solução*
tiles_14	242	660	27	36m55.760s	*sem solução*
tiles_15	242	660	27	45m51.299s	*sem solução*
tiles_16	242	660	27	38m16.465s	*sem solução*
tiles_17	242	660	27	35m54.777s	*sem solução*
tiles_18	242	660	27	32m51.224s	*sem solução*
tiles_19	242	660	27	42m31.305s	*sem solução*
tiles_20	242	660	27	38m39.638s	*sem solução*
tiles_sat_1	113	192	19	0m3.284s	*solucionável*
tiles_sat_2	113	192	24	0m7.916s	*solucionável*
tiles_sat_3	113	192	32	0m12.512s	*solucionável*

Tabela 3 – Resultado da execução do algoritmo para os problemas do domínio *Sliding-Tiles*.

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
pegsol_5	175	76	–	–	*desconhecido*
pegsol_6	175	76	–	–	*desconhecido*
pegsol_7	175	76	–	–	*desconhecido*
pegsol_8	175	76	–	–	*desconhecido*
pegsol_9	175	76	–	–	*desconhecido*
pegsol_10	175	76	–	–	*desconhecido*
pegsol_11	175	76	–	–	*desconhecido*
pegsol_12	175	76	–	–	*desconhecido*
pegsol_13	175	76	–	–	*desconhecido*
pegsol_14	175	76	–	–	*desconhecido*
pegsol_15	175	76	–	–	*desconhecido*
pegsol_16	175	76	–	–	*desconhecido*
pegsol_17	175	76	–	–	*desconhecido*
pegsol_18	175	76	–	–	*desconhecido*
pegsol_19	175	76	–	–	*desconhecido*
pegsol_20	175	76	–	–	*desconhecido*
pegsol_21	175	76	–	–	*desconhecido*
pegsol_22	175	76	–	–	*desconhecido*
pegsol_23	175	76	–	–	*desconhecido*
pegsol_24	175	76	–	–	*desconhecido*
pegsol_sat_1	175	76	–	–	*desconhecido*
pegsol_sat_2	175	76	–	–	*desconhecido*
pegsol_sat_3	175	76	–	–	*desconhecido*
pegsol_sat_4	175	76	–	–	*desconhecido*
pegsol_sat_5	175	76	–	–	*desconhecido*

Tabela 4 – Resultado da execução do algoritmo para os problemas do domínio *Pegsol*.

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
peg_row5_1	26	8	3	0m0.189s	*sem solução*
peg_row5_2	62	20	4	0m5.915s	*sem solução*
peg_row5_3	124	52	4	135m6.506s	*sem solução*
peg_row5_4	200	92	–	–	*desconhecido*
peg_row5_5	290	140	–	–	*desconhecido*
peg_row5_6	394	196	–	–	*desconhecido*
peg_row5_7	512	260	–	–	*desconhecido*
peg_row5_8	644	332	–	–	*desconhecido*
peg_row5_9	790	412	–	–	*desconhecido*
peg_row5_10	950	500	–	–	*desconhecido*
peg_row5_11	1124	596	–	–	*desconhecido*
peg_row5_12	1312	700	–	–	*desconhecido*
peg_row5_13	–	–	–	–	*desconhecido*
peg_row5_14	–	–	–	–	*desconhecido*
peg_row5_15	–	–	–	–	*desconhecido*
peg_row5_sat_1	–	–	–	–	*desconhecido*
peg_row5_sat_2	–	–	–	–	*desconhecido*
peg_row5_sat_3	–	–	–	–	*desconhecido*
peg_row5_sat_4	–	–	–	–	*desconhecido*
peg_row5_sat_5	–	–	–	–	*desconhecido*

Tabela 5 – Resultado da execução do algoritmo para os problemas do domínio *Pegsol-Row5*.

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
bottleneck_1	238	624	5	0m2.228s	*sem solução*
bottleneck_2	270	728	5	0m2.106s	*sem solução*
bottleneck_3	304	840	5	0m2.588s	*sem solução*
bottleneck_4	572	2100	6	0m6.174s	*sem solução*
bottleneck_5	621	2310	6	0m6.586s	*sem solução*
bottleneck_6	672	2530	6	0m7.856s	*sem solução*
bottleneck_7	725	2760	6	0m8.874s	*sem solução*
bottleneck_8	1184	5580	7	0m46.928s	*sem solução*
bottleneck_9	1254	5952	7	0m51.985s	*sem solução*
bottleneck_10	1326	6336	7	0m57.617s	*sem solução*
bottleneck_11	1400	6732	7	1m3.648s	*sem solução*
bottleneck_12	1476	7140	7	1m9.156s	*sem solução*
bottleneck_13	2200	12642	8	7m32.585s	*sem solução*
bottleneck_14	2295	13244	8	8m1.953s	*sem solução*
bottleneck_15	2392	13860	8	8m43.050s	*sem solução*
bottleneck_16	2491	14490	8	9m32.188s	*sem solução*
bottleneck_17	2592	15134	8	10m48.741s	*sem solução*
bottleneck_18	2695	15792	8	12m41.164s	*sem solução*
bottleneck_19	3770	25536	9	107m3.436s	*sem solução*
bottleneck_20	3894	26448	9	132m6.565s	*sem solução*
bottleneck_21	4020	27376	9	86m48.575s	*sem solução*
bottleneck_22	4148	28320	9	90m58.204s	*sem solução*
bottleneck_23	4278	29280	9	141m33.105s	*sem solução*
bottleneck_24	4410	30256	9	113m21.419s	*sem solução*
bottleneck_25	4544	31248	9	106m26.364s	*sem solução*

Tabela 6 – Resultado da execução do algoritmo para os problemas do domínio *Bottleneck*.

4.3 Desempenho do Planejador como Verificação Simbólica de Modelos com as restrições da UIPC-2016

Esta seção mostra os resultados dos experimentos realizados executando-se o planejador como verificação simbólica de modelos passando como entrada os problemas referentes aos quatro domínios descritos na seção 4.1. Para esses resultados foi considerado um tempo limite de 30 minutos.

As tabelas 7, 9 e 8 mostram os resultados obtidos para os problemas dos domínios *sliding-tiles*, *bottleneck* e *pegsol-row5*. Assim como nas tabelas da seção anterior, cada linha da tabela representa o resultado de um problema e as colunas estão organizadas de modo a mostrar: o nome do problema; o número de predicados instanciados do problema; o número de ações; o número de camadas que o algoritmo computou na busca por uma solução; o tempo gasto para o algoritmo decidir se o problema é solucionável e a classificação dada como saída (*solucionável*,

sem solução ou *desconhecido*).

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
tiles_1	113	192	73	0m25.815s	*sem solução*
tiles_2	113	192	73	0m25.846s	*sem solução*
tiles_3	113	192	73	0m25.075s	*sem solução*
tiles_4	113	192	73	0m25.401s	*sem solução*
tiles_5	113	192	73	0m20.416s	*sem solução*
tiles_6	113	192	73	0m32.468s	*sem solução*
tiles_7	113	192	73	0m24.600s	*sem solução*
tiles_8	113	192	73	0m25.632s	*sem solução*
tiles_9	113	192	73	0m30.497s	*sem solução*
tiles_10	113	192	73	0m31.802s	*sem solução*

Tabela 7 – Resultado da execução do algoritmo para os problemas do domínio *Sliding-Tiles* com restrição de tempo.

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
peg_row5_1	26	8	3	0m0.189s	*sem solução*
peg_row5_2	62	20	4	0m5.915s	*sem solução*

Tabela 8 – Resultado da execução do algoritmo para os problemas do domínio *Pegsol-Row5* com restrição de tempo.

Problema	# Predicados	# Ações	# Camadas	Tempo	Classificação
bottleneck_1	238	624	5	0m2.228s	*sem solução*
bottleneck_2	270	728	5	0m2.106s	*sem solução*
bottleneck_3	304	840	5	0m2.588s	*sem solução*
bottleneck_4	572	2100	6	0m6.174s	*sem solução*
bottleneck_5	621	2310	6	0m6.586s	*sem solução*
bottleneck_6	672	2530	6	0m7.856s	*sem solução*
bottleneck_7	725	2760	6	0m8.874s	*sem solução*
bottleneck_8	1184	5580	7	0m46.928s	*sem solução*
bottleneck_9	1254	5952	7	0m51.985s	*sem solução*
bottleneck_10	1326	6336	7	0m57.617s	*sem solução*
bottleneck_11	1400	6732	7	1m3.648s	*sem solução*
bottleneck_12	1476	7140	7	1m9.156s	*sem solução*
bottleneck_13	2200	12642	8	7m32.585s	*sem solução*
bottleneck_14	2295	13244	8	8m1.953s	*sem solução*
bottleneck_15	2392	13860	8	8m43.050s	*sem solução*
bottleneck_16	2491	14490	8	9m32.188s	*sem solução*
bottleneck_17	2592	15134	8	10m48.741s	*sem solução*
bottleneck_18	2695	15792	8	12m41.164s	*sem solução*

Tabela 9 – Resultado da execução do algoritmo para os problemas do domínio *Bottleneck* com restrição de tempo.

4.4 Análise Comparativa

Nesta seção realizamos uma análise comparativa entre o planejador baseado em verificação simbólica de modelos α-CTL e os algoritmos participantes da UIPC. Na competição, o foco principal é a cobertura de problemas identificados corretamente como não solucionáveis. Não há pontos atribuídos para os problemas solucionáveis, uma vez que elas somente foram introduzidas para evitar que simplesmente um algoritmo pudesse retornar "*unsolvable*" para todos os problemas.

O algoritmo de planejamento como verificação simbólica de modelos baseado na lógica α-CTL, com limite de tempo de 8 horas, conseguiu decidir corretamente como não solucionáveis 48 dos 84 problemas não solucionáveis dados como entrada. Sendo que conseguiu identificar corretamente todos os problemas do domínio *Sliding-Tiles* e do domínio *Bottleneck*, mas não obteve o mesmo desempenho no domínio *Pegsol-Row5*, onde conseguiu identificar como não solucionáveis apenas 3 dos 15 problemas dados como entrada. O algoritmo não obteve um bom desempenho no domínio *Pegsol*, visto que não conseguiu resolver nenhum problema desse domínio.

O algoritmo com o limite de tempo utilizado na competição (30 minutos) conseguiu decidir corretamente como não solucionáveis 30 dos 84 problemas não solucionáveis dados como entrada. Deste modo, mesmo com o limite de tempo o algoritmo manteve o bom desempenho para os domínios *Sliding-Tiles* e *Bottleneck*, assim como o baixo desempenho para os domínios *Pegsol-Row5* e *Pegsol*. Observou-se que nenhum problema dos quatro domínios passou a ser não solucionável ou deixou de ser não solucionável com a inserção da restrição de tempo nos experimentos.

O gráfico18 apresenta para cada planejador o número de problemas corretamente classificados como não-solucionáveis para os domínios *Sliding-Tiles*, *Pegsol*, *Pegsol-Row5* e *Bottleneck* utilizados nesta análise. Observa-se que o planejador baseado em α-CTL para o domínio *Sliding-Tiles* conseguiu identificar corretamente a mesma quantidade de problemas que a maioria dos planejadores. Para o domínio *Pegsol-Row5* ele se aproximou da quantidade de problemas decididos pelo *SymPA* (segundo colocado na UIPC-2016) e ficou relativamente distante do Aidos (vencedor da UIPC-2016). O desempenho do planejador α-CTL sobre o domínio *Bottleneck* é semelhante ao desempenho da maioria dos planejadores. No entanto, para o domínio *Pegsol* há uma grande diferença de desempenho em relação ao planejador Aidos, mas

é igual se comparado com os planejadores IProver e h^{++}, os quais também não conseguiram decidir nenhum problema desse domínio. Acreditamos que existem pelo menos duas causas para o mau desempenho do planejador sobre o domínio *Pegsol*: grande quantidade de *invariantes* e; pequena quantidade de proposições mutualmente exclusivas (*constraints*).

Os problemas do domínio *Pegsol* possuem um grande número de proposições que não são modificadas pelas ações, as chamadas *invariantes*. Por exemplo, o problema $pegsol_5$ das 175, 109 são invariantes. Todas estas proposições são verdadeiras no estado inicial e mantém-se verdadeira em todos os possíveis estados do espaço de busca. Assim, a busca regressiva a partir da meta realiza esforço desnecessário ao tentar raciocinar sobre os possíveis valores que tais proposições podem assumir. Uma outra causa é a pequena quantidade de proposições mutualmente exclusivas. Tais proposições podem ajudar a reduzir drasticamente o número de estados do espaço de busca. Das 175 proposições do problema $pegsol_5$, 33 pares de proposições são mutualmente excludentes, pois para cada uma dos 33 espaços do tabuleiro, há uma proposição para indicar que a posição está livre e uma proposição para indicar que a posição está ocupada. Assim, podemos utilizar apenas a proposição que indica que a posição está livre e a sua negação para indicar que a posição está ocupada. Assim, é possível reduzir o tamanho do espaço de estados de 2^{175} para $2^{175-33} = 2^{142}$.

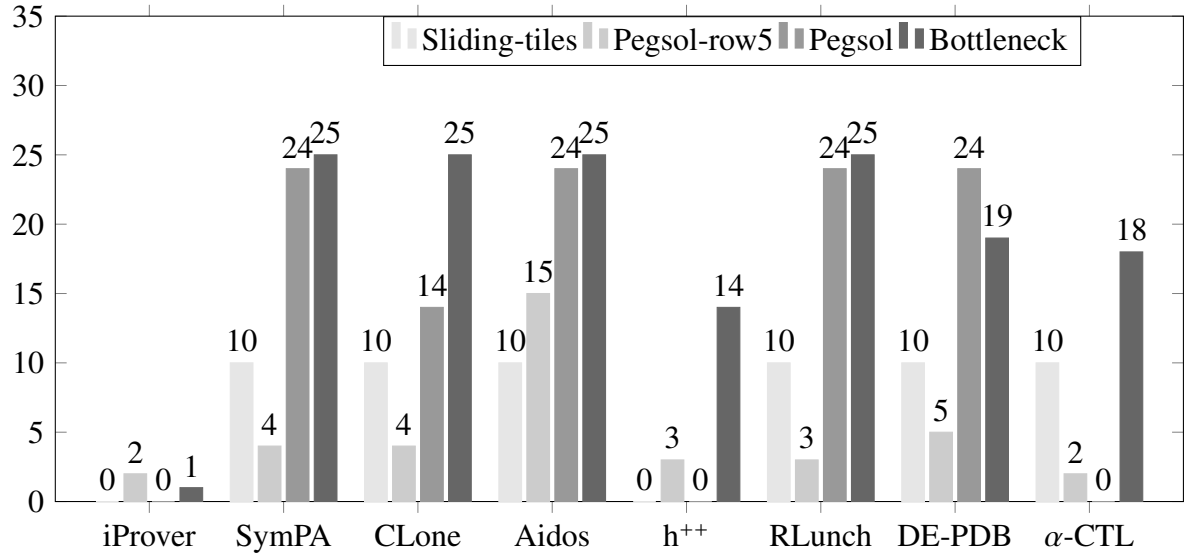


Figura 18 – Comparativo do número de problemas detectados corretamente como sem solução pelos participantes da UIPC e pelo planejador α-CTL para os domínios Sliding-tiles, Pegsol-row5, Pegsol e Bottleneck.

Foram introduzidas restrições de estados para reduzir a quantidade de valorações possíveis para as proposições dos problemas, essas restrições diminuem o espaço de estados e

faz com que o algoritmo tenha um bom desempenho para domínios com restrições que afetam consideravelmente o espaço de estado, por exemplo restrições de localização.

Os domínios *Sliding-Tiles* e *Bottleneck* possuem restrições de localização, no primeiro só pode existir um quadro em branco e uma peça só pode estar em uma posição do tabuleiro. No segundo domínio podemos ter apenas uma pessoa por posição da matriz.

Portanto, podemos concluir que o algoritmo obteve um bom desempenho sobre dois (*Sliding-Tiles* e *Bottleneck*) dos quatro domínios utilizados nesta análise, um desempenho intermediário sobre o domínio *Pegsol-Row5* e um desempenho ruim sobre o domínio *Pegsol*.

5 CONCLUSÃO

Planejamento Automatizado é a tarefa de propor uma sequência de ações que levam um agente a alcançar um objetivo. A solução para um problema de planejamento é uma sequência de ações que permite que o agente inteligente, a partir do estado inicial, alcance um estado meta. No entanto, nem sempre é possível obter uma solução para um problema de planejamento. Dizemos, nestes casos, que o problema de planejamento não possui solução.

Problemas sem Solução são desafiadores até mesmo para planejadores estado-da-arte. Assim, em 2016 foi organizada, pela primeira vez, a UIPC, uma modalidade da competição internacional de planejamento com foco em algoritmos capazes de detectar se um problema de planejamento possui ou não solução.

A *Verificação Simbólica de Modelos* é uma abordagem formal que pode ser utilizada para detectar de forma eficiente se um problema de planejamento possui ou não solução. Nesta abordagem, os estados e transições do domínio de planejamento são representados por meio de Diagramas de Decisão Binária (BDDs), estruturas capazes de armazenar de forma eficiente uma grande quantidade de estados (até 2^{100} estados). A abordagem de verificação simbólica de modelos mostrou-se eficiente na UIPC-2016, na qual um planejador que utiliza tal técnica (planejador `SymPA`) alcançou o segundo lugar na competição.

Este trabalho mostrou o desempenho do planejador como verificação simbólica de modelos baseado na lógica α-CTL para quatro domínios *benchmarks* da UIPC, a saber: *sliding-tiles*, *pegsol*, *pegsol-row5* e *bottleneck*. Por meio dos experimentos realizados, concluímos que o algoritmo obteve um bom desempenho nos domínios *sliding-tiles* e *bottleneck*, conseguindo reportar corretamente se um problema era ou não solucionável em todos os problemas apresentados. Por outro lado, não obteve um bom desempenho nos domínios *pegsol* e *pegsol-row5*, decidir em tempo hábil se um problema era ou não solucionável para a maioria dos problemas. No entanto, concluímos que o planejador baseado em verificação de modelos α-CTL obteve desempenho similar para estes domínios quando comparado ao desempenho obtido pela maioria dos planejadores participantes da UIPC-2016.

A análise experimental também permitiu observar que:

- o limite do tamanho do espaço de estados que o algoritmo consegue tratar é por volta de 2^{100} estados (BURCH et al., 1992);
- é fundamental que restrições de proposições mutualmente excludentes sejam adicionadas às descrições dos problemas, uma vez que estas podem reduzir drasticamente o espaço de

busca;

- as invariantes são proposições que poderiam ser descartadas da representação dos estados, uma vez que não são modificadas pelas ações. Notamos que o algoritmo teve um comportamento pior em domínios com uma grande quantidade de proposições invariantes. Uma análise prévia deste conjunto de proposições poderia ser uma estratégia adotada para reduzir o espaço de busca;
- é necessário elaborar estratégias que possam reduzir o tamanho dos BDDs, de forma a tornar mais eficiente a busca por uma solução. As estratégias de controle do espaço de estados implementadas pelo planejador `Sympa` foram recentemente publicadas em (TORRALBA et al., 2017).

Como trabalhos futuros temos: a realização de experimentos para os outros 11 domínios *benchmarks* da UIPC-2016 e; a avaliação do planejador como verificação simbólica de modelos baseado em α-CTL em domínios não-determinísticos.

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. **UIPC 2016 Planner Abstracts**, 2016.

BRYANT, R. E. Graph-based algorithms for boolean function manipulation. **IEEE Trans. Comput.**, IEEE Computer Society, Washington, DC, USA, v. 35, n. 8, p. 677–691, 1986. ISSN 0018-9340.

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: **Logic of Programs, Workshop**. London, UK: Springer-Verlag, 1982. p. 52–71. ISBN 3-540-11212-X.

CLARKE, E. M.; GRUMBERG, O.; PELED, D. **Model checking**. [S.l.]: MIT press, 1999.

DIMOPOULOS, Y.; GEREVINI, A.; HASLUM, P.; SAETTI, A. The benchmark domains of the deterministic part of IPC-5. **Abstract Booklet of the competing planners of ICAPS-06**, p. 14–19, 2006.

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

EROL, K.; NAU, D. S.; SUBRAHMANIAN, V. S. Complexity, decidability and undecidability results for domain-independent planning. **Artificial intelligence**, Elsevier, v. 76, n. 1, p. 75–88, 1995.

FIKES, R. E.; NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. **Artificial intelligence**, Elsevier, v. 2, n. 3-4, p. 189–208, 1971.

GIUNCHIGLIA, F.; TRAVERSO, P. Planning as model checking. **Recent Advances in AI Planning**, Springer, p. 1–20, 2000.

GNAD, D.; STEINMETZ, M.; HOFFMANN, J. Django: Unchaining the power of red-black planning. In: **UIPC 2016 Planner Abstracts**. [S.l.: s.n.], 2016. p. 19–23.

GNAD, D.; TORRALBA, Á.; HOFFMANN, J.; WEHRLE, M. Decoupled search for proving unsolvability. In: **UIPC 2016 Planner Abstracts**. [S.l.: s.n.], 2016. p. 16–18.

GÖBELBECKER, M.; KELLER, T.; EYERICH, P.; BRENNER, M.; NEBEL, B. Coming up with good excuses: What to do when no plan can be found. In: **Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)**. [S.l.: s.n.], 2010. p. 81–88.

HASLUM, P. Adapting h++ for proving plan non-existence. **UIPC 2016 Planner Abstracts**, 2016.

HASLUM, P.; SLANEY, J.; THIÉBAUX, S. et al. Incremental lower bounds for additive cost planning problems. In: **ICAPS**. [S.l.: s.n.], 2012. v. 12, p. 74–82.

HELMERT, M. The fast downward planning system. **Journal of Artificial Intelligence Research**, v. 26, p. 191–246, 2006.

HOFFMANN, J. Ff: The fast-forward planning system. **AI magazine**, v. 22, n. 3, p. 57, 2001.

HUTH, M.; RYAN, M. **Lógica em ciência da computação**: modelagem e argumentação sobre sistemas. **Rio de Janeiro, 2a edição edition. tradução e revisão técnica Valéria de Magalhães Iório**, 2008.

KAUTZ, H. A.; SELMAN, B. et al. Planning as satisfiability. In: **ECAI**. [S.l.: s.n.], 1992. v. 92, p. 359–363.

KOROVIN, K.; SUDA, M. Iproverplan: a system description. **UIPC 2016 Planner Abstracts**, 2016.

KRIPKE, S. Semantical considerations on modal logic. **Acta Philosophica Fennica**, v. 16, p. 83–94, 1963.

MCDERMOTT, D.; GHALLAB, M.; HOWE, A.; KNOBLOCK, C.; RAM, A.; VELOSO, M.; WELD, D.; WILKINS, D. **Coorientador-the planning domain definition language**. 1998.

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

MENEZES, M. V.; BARROS, L. N.; PEREIRA, S. L. Symbolic regression for non deterministic actions. **Learning and Nonlinear Models (to appear)**, 2014.

MUISE, C.; LIPOVETZKY, N. Unsolvability international planning competition. 2016. Available from <http://unsolve-ipc.eng.unimelb.edu.au/>. Aceeso em: 30/08/2017.

NEWELL, A.; SHAW, J. C.; SIMON, H. A. Report on a general problem solving program. In: PITTSBURGH, PA. **IFIP congress**. [S.l.], 1959. v. 256, p. 64.

NILSSON, N. **Principles ofArtificial Intelligence**, palo alto, ca: Tioga. 1980.

PEREIRA, S. L. **Planejamento sob incerteza para metas de alcancabilidade estendidas**. Tese (Doutorado) — Instituto de Matemática e Estatística da Universidade de São Paulo, 2007.

PEREIRA, S. L.; BARROS, L. N. A logic-based agent that plans for extended reachability goals. **Autonomous Agents and Multi-Agent Systems (AAMAS)**, Springer Netherlands, v. 16, n. 3, p. 327–344, June 2008. ISSN 1387-2532 (Print) 1573-7454 (Online). Disponível em: <http://www.springerlink.com/content/er325t66t260396u/>.

POMMERENING, F.; SEIPP, J. Fast downward dead-end pattern database. **Unsolvability International Planning Competition: planner abstracts**, 2016.

RUSSELL STUART, P. N. **Artificial intelligence: A modern approach**. [S.l.]: Prentice Hall, 2009.

SANNER, S. Relational dynamic influence diagram language (rddl): Language description. **Unpublished ms. Australian National University**, p. 32, 2010.

SEIPP, J.; POMMERENING, F.; SIEVERS, S.; WEHRLE, M.; FAWCETT, C.; ALKHAZRAJI, Y. Fast downward aidos. In: **UIPC 2016 Planner Abstracts**. [S.l.: s.n.], 2016. p. 28–38.

STAALMARCK, G. Quantified boolean formulas. In: SPRINGER. **Computer Aided Verification: 11th International Conference, CAV'99, Trento, Italy, July 6-10, 1999, Proceedings**. [S.l.], 2003. p. 23.

STEINMETZ, M.; HOFFMANN, J. Clone: A critical-path driven clause learner. **UIPC 2016 Planner Abstracts**, 2016.

TORRALBA, A. Sympa: Symbolic perimeter abstractions for proving unsolvability. **UIPC 2016 Planner Abstracts**, 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, A.; HOFFMANN, J.; KISSMANN, P. Ms-unsat and simulation-dominance: Merge-and-shrink and dominance pruning for proving unsolvability. In: **UIPC 2016 Planner Abstracts**. [S.l.: s.n.], 2016. p. 12–15.

APÊNDICE A – DEFINIÇÃO EM PDDL DOS DOMÍNIOS DE PLANEJAMENTO

Este apêndice mostra a definição dos domínios e dos problemas de planejamento explicados no capítulo 4.

```
(define (domain pegsolitaire-netbenefit)
  (:requirements :strips )
  (:predicates
    (location ?x)
    (location ?y)
    (location ?z)
    (location ?l)
    (IN-LINE ?x ?y ?z)
    (occupied ?l)
    (free ?l)
  )
  (:action jump
    :parameters( ?from ?over ?to)
    :precondition
      (and (location ?from) (location ?over)
      (location ?to)
      (IN-LINE ?from ?over ?to)
      (occupied ?from)
      (occupied ?over)
      (free ?to))
    :effect
      (and (not (occupied ?from)) (not (occupied ?over))
      (not (free ?to))
      (free ?from)
      (free ?over)
      (occupied ?to))
     )
)
```

Figura 19 – Descrição na linguagem *PDDL* do domínio *Pegsol*.

(define (problem pegsolitaire-netbenefit-103-1)
(:domain pegsolitaire-netbenefit)
(:objects
pos-0-2 pos-0-3 pos-0-4 pos-1-2 pos-1-3 pos-1-4 pos-2-0 pos-2-1 pos-2-2 pos-2-3 pos-2-4 pos-2-5 pos-2-6 pos-3-0 pos-3-1 pos-3-2 pos-3-3 pos-3-4 pos-3-5 pos-3-6 pos-4-0 pos-4-1 pos-4-2 pos-4-3 pos-4-4 pos-4-5 pos-4-6 pos-5-2 pos-5-3 pos-5-4 pos-6-2 pos-6-3 pos-6-4)
(:init
(location pos-0-2) (location pos-0-3) (location pos-0-4) (location pos-1-2) (location pos-1-3) (location pos-1-4) (location pos-2-0) (location pos-2-1) (location pos-2-2) (location pos-2-3) (location pos-2-4) (location pos-2-5) (location pos-2-6) (location pos-3-0) (location pos-3-1) (location pos-3-2) (location pos-3-3) (location pos-3-4) (location pos-3-5) (location pos-3-6) (location pos-4-0) (location pos-4-1) (location pos-4-2) (location pos-4-3) (location pos-4-4) (location pos-4-5) (location pos-4-6) (location pos-5-2) (location pos-5-3) (location pos-5-4) (location pos-6-2) (location pos-6-3) (location pos-6-4)
(IN-LINE pos-0-2 pos-0-3 pos-0-4) (IN-LINE pos-0-4 pos-0-3 pos-0-2) (IN-LINE pos-0-2 pos-1-2 pos-2-2 (IN-LINE pos-2-2 pos-1-2 pos-0-2) (IN-LINE pos-0-3 pos-1-3 pos-2-3) (IN-LINE pos-2-3 pos-1-3 pos-0-3) (IN-LINE pos-0-4 pos-1-4 pos-2-4) (IN-LINE pos-2-4 pos-1-4 pos-0-4) (IN-LINE pos-1-2 pos-1-3 pos-1-4) (IN-LINE pos-1-4 pos-1-3 pos-1-2) (IN-LINE pos-1-2 pos-2-2 pos-3-2) (IN-LINE pos-3-2 pos-2-2 pos-1-2) (IN-LINE pos-1-3 pos-2-3 pos-3-3) (IN-LINE pos-3-3 pos-2-3 pos-1-3) (IN-LINE pos-1-4 pos-2-4 pos-3-4) IN-LINE pos-3-4 pos-2-4 pos-1-4) (IN-LINE pos-2-0 pos-2-1 pos-2-2)(IN-LINE pos-2-2 pos-2-1 pos-2-0) (IN-LINE pos-2-0 pos-3-0 pos-4-0) (IN-LINE pos-4-0 pos-3-0 pos-2-0) (IN-LINE pos-2-1 pos-2-2 pos-2-3) (IN-LINE pos-2-3 pos-2-2 pos-2-1) (IN-LINE pos-2-1 pos-3-1 pos-4-1) (IN-LINE pos-4-1 pos-3-1 pos-2-1) (IN-LINE pos-2-2 pos-2-3 pos-2-4) (IN-LINE pos-2-4 pos-2-3 pos-2-2) (IN-LINE pos-2-2 pos-3-2 pos-4-2) (IN-LINE pos-4-2 pos-3-2 pos-2-2) (IN-LINE pos-2-3 pos-2-4 pos-2-5) (IN-LINE pos-2-5 pos-2-4 pos-2-3) (IN-LINE pos-2-3 pos-3-3 pos-4-3) (IN-LINE pos-4-3 pos-3-3 pos-2-3) (IN-LINE pos-2-4 pos-2-5 pos-2-6) (IN-LINE pos-2-6 pos-2-5 pos-2-4) (IN-LINE pos-2-4 pos-3-4 pos-4-4) (IN-LINE pos-4-4 pos-3-4 pos-2-4) (IN-LINE pos-2-5 pos-3-5 pos-4-5) (IN-LINE pos-4-5 pos-3-5 pos-2-5) (IN-LINE pos-2-6 pos-3-6 pos-4-6) (IN-LINE pos-4-6 pos-3-6 pos-2-6) (IN-LINE pos-3-0 pos-3-1 pos-3-2) (IN-LINE pos-3-2 pos-3-1 pos-3-0) (IN-LINE pos-3-1 pos-3-2 pos-3-3) (IN-LINE pos-3-3 pos-3-2 pos-3-1) (IN-LINE pos-3-2 pos-3-3 pos-3-4) (IN-LINE pos-3-4 pos-3-3 pos-3-2) (IN-LINE pos-3-2 pos-4-2 pos-5-2) (IN-LINE pos-5-2 pos-4-2 pos-3-2) (IN-LINE pos-3-3 pos-3-4 pos-3-5) (IN-LINE pos-3-5 pos-3-4 pos-3-3) (IN-LINE pos-3-3 pos-4-3 pos-5-3) (IN-LINE pos-5-3 pos-4-3 pos-3-3) (IN-LINE pos-3-4 pos-3-5 pos-3-6) (IN-LINE pos-3-6 pos-3-5 pos-3-4) (IN-LINE pos-3-4 pos-4-4 pos-5-4) (IN-LINE pos-5-4 pos-4-4 pos-3-4) (IN-LINE pos-4-0 pos-4-1 pos-4-2) (IN-LINE pos-4-2 pos-4-1 pos-4-0) (IN-LINE pos-4-1 pos-4-2 pos-4-3) (IN-LINE pos-4-3 pos-4-2 pos-4-1) (IN-LINE pos-4-2 pos-4-3 pos-4-4) (IN-LINE pos-4-4 pos-4-3 pos-4-2) (IN-LINE pos-4-2 pos-5-2 pos-6-2) (IN-LINE pos-6-2 pos-5-2 pos-4-2) (IN-LINE pos-4-3 pos-4-4 pos-4-5) (IN-LINE pos-4-5 pos-4-4 pos-4-3) (IN-LINE pos-4-3 pos-5-3 pos-6-3) (IN-LINE pos-6-3 pos-5-3 pos-4-3) (IN-LINE pos-4-4 pos-4-5 pos-4-6) (IN-LINE pos-4-6 pos-4-5 pos-4-4) (IN-LINE pos-4-4 pos-5-4 pos-6-4) (IN-LINE pos-6-4 pos-5-4 pos-4-4) (IN-LINE pos-5-2 pos-5-3 pos-5-4) (IN-LINE pos-5-4 pos-5-3 pos-5-2) (IN-LINE pos-6-2 pos-6-3 pos-6-4) (IN-LINE pos-6-4 pos-6-3 pos-6-2) (free pos-0-2) (free pos-0-4)
(free pos-1-2) (free pos-1-3) (free pos-1-4) (free pos-2-0) (free pos-2-1) (free pos-2-5) (free pos-2-6) (free pos-3-0) (free pos-3-3) (free pos-3-5) (free pos-4-0) (free pos-4-1) (free pos-4-3) (free pos-4-4) (free pos-4-5) (free pos-4-6) (free pos-5-2) (free pos-5-3) (free pos-5-4) (free pos-6-2) (free pos-6-3) (free pos-6-4) (occupied pos-0-3) (occupied pos-2-2) (occupied pos-2-3) (occupied pos-2-4) (occupied pos-3-1) (occupied pos-3-2) (occupied pos-3-4) (occupied pos-3-6) (occupied pos-4-2))
(:goal
(free pos-0-2) (free pos-0-3) (free pos-0-4) (free pos-1-2) (free pos-1-3) (free pos-1-4) (free pos-2-0) (free pos-2-1) (free pos-2-2) (free pos-2-3) (free pos-2-4) (free pos-2-5) (free pos-2-6) (free pos-3-0) (free pos-3-1) (free pos-3-2) (occupied pos-3-3) (free pos-3-4) (free pos-3-5) (free pos-3-6) (free pos-4-0) (free pos-4-1) (free pos-4-2) (free pos-4-3) (free pos-4-4) (free pos-4-5) (free pos-4-6) (free pos-5-2) (free pos-5-3) (free pos-5-4) (free pos-6-2) (free pos-6-3) (free pos-6-4)))

Figura 20 – Descrição de um problema do domínio Pegsol.

```
(define (domain pegsolitaire-invasion)
  (:requirements :typing )
  (:predicates
    (location ?x)
    (location ?y)
    (location ?z)
    (location ?l)
    (IN-LINE ?x ?y ?z)
    (occupied ?l)
    (free ?l)
  )
  (:action jump
    :parameters( ?from ?over ?to)
    :precondition
      (and (location ?from) (location ?over)
      (location ?to)
      (IN-LINE ?from ?over ?to)
      (occupied ?from)
      (occupied ?over)
      (free ?to))
    :effect
      (and (not (occupied ?from)) (not (occupied ?over))
      (not (free ?to))
      (free ?from)
      (free ?over)
      (occupied ?to)))
)
```

Figura 21 – Descrição na linguagem *PDDL* do domínio *Pegsol-Row5*.

```
(define (problem pegsolitaire-invasion-1)
   (:domain pegsolitaire-invasion)
   (:objects
      pos-0-0 pos-0-1 pos-0-2 pos-0-3 pos-0-4 pos-0-5
   )
   (:init
      (location pos-0-0) (location pos-0-1) (location pos-0-2)
      (location pos-0-3) (location pos-0-4) (location pos-0-5)
      (IN-LINE pos-0-0 pos-0-1 pos-0-2)
      (IN-LINE pos-0-1 pos-0-2 pos-0-3)
      (IN-LINE pos-0-2 pos-0-3 pos-0-4)
      (IN-LINE pos-0-3 pos-0-4 pos-0-5)
      (IN-LINE pos-0-2 pos-0-1 pos-0-0)
      (IN-LINE pos-0-3 pos-0-2 pos-0-1)
      (IN-LINE pos-0-4 pos-0-3 pos-0-2)
      (IN-LINE pos-0-5 pos-0-4 pos-0-3)
      (free pos-0-1)
      (free pos-0-2)
      (free pos-0-3)
      (free pos-0-4)
      (free pos-0-5)
      (occupied pos-0-0)
   )
   (:goal
      (and
         (occupied pos-0-5)
      )
   )
)
```

Figura 22 – Descrição de um problema do domínio *Pegsol-Row5*.

```
(define (domain strips-sliding-tile)
   (:requirements :strips)
   (:predicates
      (tile ?x) (xposition ?x) (yposition ?x) (at ?t ?x ?y) (blank ?x ?y)
      (inc ?p ?pp) (dec ?p ?pp)
    )
   (:action move-up
      :parameters (?t ?px ?py ?by)
      :precondition (and
         (tile ?t) (xposition ?px) (yposition ?py) (yposition ?by)
         (dec ?by ?py) (blank ?px ?by) (at ?t ?px ?py))
      :effect (and (not (blank ?px ?by)) (not (at ?t ?px ?py))
         (blank ?px ?py) (at ?t ?px ?by))
    )
   (:action move-down
      :parameters (?t ?px ?py ?by)
      :precondition (and
         (tile ?t) (xposition ?px) (yposition ?py) (yposition ?by)
         (inc ?by ?py) (blank ?px ?by) (at ?t ?px ?py))
      :effect (and (not (blank ?px ?by)) (not (at ?t ?px ?py))
         (blank ?px ?py) (at ?t ?px ?by))
    )
   (:action move-left
      :parameters (?t ?px ?py ?bx)
      :precondition (and
         (tile ?t) (xposition ?px) (yposition ?py) (xposition ?bx)
         (dec ?bx ?px) (blank ?bx ?py) (at ?t ?px ?py))
      :effect (and (not (blank ?bx ?py)) (not (at ?t ?px ?py))
         (blank ?px ?py) (at ?t ?bx ?py))
    )
   (:action move-right
      :parameters (?t ?px ?py ?bx)
      :precondition (and
         (tile ?t) (xposition ?px) (yposition ?py) (xposition ?bx)
         (inc ?bx ?px) (blank ?bx ?py) (at ?t ?px ?py))
      :effect (and (not (blank ?bx ?py)) (not (at ?t ?px ?py))
         (blank ?px ?py) (at ?t ?bx ?py))
    ) )
```

Figura 23 – Descrição na linguagem *PDDL* do domínio *Sliding-Tiles*.

```
(define (problem p3-3-puzzle-problem-100-1390833173294931248)
   (:domain strips-sliding-tile)
   (:objects
      t1 t2 t3 t4 t5 t6 t7 t8
      p1 p2 p3
   )
   (:init
      (tile t1) (tile t2) (tile t3) (tile t4)
      (tile t5) (tile t6) (tile t7) (tile t8)
      (xposition p1) (xposition p2) (xposition p3)
      (yposition p1) (yposition p2) (yposition p3)
      (inc p1 p2) (inc p2 p3)
      (dec p2 p1) (dec p3 p2)
      (blank p1 p3)
      (at t1 p1 p1) (at t2 p2 p1) (at t5 p3 p1) (at t6 p1 p2)
      (at t4 p2 p2) (at t7 p3 p2) (at t8 p2 p3) (at t3 p3 p3)
   )
   (:goal
      (and
         (at t1 p2 p1) (at t2 p3 p1) (at t3 p1 p2) (at t4 p2 p2)
         (at t5 p3 p2) (at t6 p1 p3) (at t7 p2 p3) (at t8 p3 p3)
      )
   ) )
```

Figura 24 – Descrição de um problema do domínio *Sliding-Tiles*.

```
(define (domain Resources)
  (:requirements :strips )
  (:predicates
    (location ?loc)
    (person ?p)
    (person-at ?p ?loc)
    (connected ?loc1 ?loc2)
    (active ?loc)
  )
  (:action move
    :parameters( ?p ?l1 ?l2)
    :precondition
      (and (person ?p) (location ?l1)
      (location ?l2)
      (connected ?l1 ?l2)
      (active ?l2)
    :effect
      (and (person-at ?p ?l2))
      (not (person-at ?p ?l1))
      (not (active ?l2)))
)
```

Figura 25 – Descrição na linguagem *PDDL* do domínio *Bottleneck*.

```
(define (problem grid-prob-4-1)
    (:domain Resources)
    (:objects
        p0 p1 p2 p3
        loc-0-0 loc-0-1 loc-0-2 loc-0-3 loc-1-0 loc-1-1 loc-1-2 loc-1-3
        loc-2-0 loc-3-0 loc-3-1 loc-3-2 loc-3-3
    )
    (:init
        (person p0) (person p1) (person p2) (person p3)
        (location loc-0-0) (location loc-0-1) (location loc-0-2)
        (location loc-0-3) (location loc-1-0) (location loc-1-1)
        (location loc-1-2) (location loc-1-3) (location loc-2-0)
        (location loc-3-0) (location loc-3-1) (location loc-3-2)
        (location loc-3-3)
        (active loc-1-0) (active loc-1-1) (active loc-1-2)
        (active loc-1-3) (active loc-2-0) (active loc-3-0)
        (active loc-3-1) (active loc-3-2) (active loc-3-3)
        (connected loc-0-0 loc-1-0) (connected loc-0-1 loc-1-1)
        (connected loc-0-2 loc-1-2) (connected loc-0-3 loc-1-3)
        (connected loc-1-0 loc-2-0) (connected loc-2-0 loc-3-0)
        (connected loc-1-0 loc-0-0) (connected loc-1-1 loc-0-1)
        (connected loc-1-2 loc-0-2) (connected loc-1-3 loc-0-3)
        (connected loc-2-0 loc-1-0) (connected loc-3-0 loc-2-0)
        (connected loc-0-0 loc-0-1) (connected loc-0-1 loc-0-2)
        (connected loc-0-2 loc-0-3) (connected loc-1-0 loc-1-1)
        (connected loc-1-1 loc-1-2) (connected loc-1-2 loc-1-3)
        (connected loc-3-0 loc-3-1) (connected loc-3-1 loc-3-2)
        (connected loc-3-2 loc-3-3) (connected loc-0-1 loc-0-0)
        (connected loc-0-2 loc-0-1) (connected loc-0-3 loc-0-2)
        (connected loc-1-1 loc-1-0) (connected loc-1-2 loc-1-1)
        (connected loc-1-3 loc-1-2) (connected loc-3-1 loc-3-0)
        (connected loc-3-2 loc-3-1) (connected loc-3-3 loc-3-2)
        (person-at p0 loc-0-0) (person-at p1 loc-0-1)
        (person-at p2 loc-0-2) (person-at p3 loc-0-3)
    )
    (:goal
       (and
          (person-at p0 loc-3-0)
          (person-at p1 loc-3-1)
          (person-at p2 loc-3-2)
          (person-at p3 loc-3-3)
       )
    )
)
```

Figura 26 – Descrição em *PDDL* de um problema do domínio *Bottleneck*.