ENIAC - 2017
XIV Encontro Nacional de Inteligência Artificial e Computacional

Realização: Organização: Promoção:

XIV Encontro Nacional de Inteligência Artificial e Computacional
2 a 5 de outubro de 2017
Uberlândia, MG, Brasil

Anais

Comissão Especial de Inteligência Artificial da Sociedade Brasileira de Computação (CEIA-SBC)
Comissão Especial de Inteligência Computacional da Sociedade Brasileira de Computação (CEIC-SBC)

COORDENADORES DO COMITÊ DE PROGRAMA
Aline Paes (UFF)
André Britto (UFS)

EDITORES
Aline Paes (UFF)
André Britto (UFS)
Gina M. B. de Oliveira (UFU)

REALIZAÇÃO
Sociedade Brasileira de Computação

ORGANIZAÇÃO
Faculdade de Computação - Universidade Federal de Uberlândia

PACTL-SYM: um planejador baseado em Verificação Simbólica de Modelos

Viviane Bonadia dos Santos[1], **Leliane Nunes de Barros**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
Rua do Matão, 1010 - CEP 05508-090 - São Paulo - SP

{vbonadia,leliane}@ime.usp.br

Resumo. *Planejamento clássico em inteligência artificial, cujo objetivo é encontrar um plano de ações que alcance um estado meta a partir de um estado inicial, faz suposição de ambientes completamente observáveis e ações com efeitos determinísticos. Uma extensão de planejamento clássico, chamada de* FOND *(Fully-Observable Non-Deterministic Problem), considera ações com efeitos não determinísticos e a geração de políticas que eventualmente terminam em becos-sem-saídas. Soluções conhecidas para esse problema são baseadas em técnicas de verificação de modelos sendo que a maioria delas utiliza a lógica temporal de tempo ramificado* CTL*. Para contornar as limitações da lógica* CTL*, que não leva em consideração as ações, foi proposta uma nova lógica, a lógica* α-CTL*, e um planejador para problemas* FOND*, chamado* PACTL*. Neste trabalho apresentamos a versão simbólica desse planejador, que representa estados e ações como fórmulas lógicas, utiliza técnicas de verificação simbólica de modelos e diagramas de decisão binária (*BDDS*).*

1. Introdução

Em inteligência artificial, planejamento consiste no processo de seleção de ações que ao serem executadas garantem atingir um conjunto de metas preestabelecidas $\mathcal{G}$ a partir de um estado inicial s_0 [Nau et al. 2004]. No planejamento clássico são feitas diversas suposições restritivas sobre o ambiente que o agente atua, entre elas: o ambiente é completamente observável, discreto, finito e determinístico, isto é, não existe incerteza sob os efeitos das ações executadas pelo agente. Uma classe de planejadores que lidam com problemas que possuem menos restrições considera ações com efeitos não determinísticos, chamados de FOND (*Fully-Observable Non-Deterministic Problem*).

A Figura 1 apresenta um exemplo esquemático de domínios de planejamento em que os nós representam estados do ambiente e as arestas representam as ações que ao serem executadas em um estado levam o agente a outro estado. A Figura 1(a) ilustra um domínio de planejamento clássico e 1(b) um domínio de planejamento não determinístico. Note que, ao executar a ação c no estado s_2 no domínio não determinístico, o agente pode ir para o estado s_3 ou para o estado s_4. A solução para um problema de planejamento clássico é uma sequência totalmente ordenada de ações; enquanto a solução para um problema FOND é um plano ramificado, isto é, uma política que associa uma ação a cada estado.

Um exemplo de planejador considerado estado-da-arte para problemas FOND é o planejador PRP (*Planner for Relevant Policies*) [Muise et al. 2012]. Apesar do PRP ser eficiente, ele não garante devolver a solução ótima, isto é, uma política que contém o

(a) Domínio determinístico. (b) Domínio não determinístico.

Figura 1. Domínio de planejamento em que s_0 é o estado inicial e $s_g \vDash \mathcal{G}$. Arestas rotuladas pela mesma ação representam efeitos não determinísticos.

caminho de menor distância para um estado meta. O PRP utiliza o planejador FD (*Fast Downward*) [Helmert 2006] (um planejador baseado em heurísticas considerado estado-da-arte em planejamento clássico) para resolver uma versão determinística do problema FOND e, em seguida, regride as ações do plano encontrado para obter uma solução mais geral para o problema FOND original. Em geral, o PRP devolve um plano subótimo.

Outro planejador que se destaca na área de planejamento não determinístico é o planejador MBP (*Model Based Planner*) [Bertoli et al. 2001], que utiliza técnicas de verificação simbólica de modelos. A ideia principal desta abordagem é resolver problemas de planejamento através de um modelo formal, dando garantia sobre a validade do plano gerado. Além disso, o MBP permite especificar metas complexas que não podem ser resolvidas pelo planejador PRP. Ao longo dos anos, diferentes métodos de verificação de modelos foram propostos sendo que, muitos deles, representavam explicitamente o espaço de estados através de tabelas. Contudo, o número de estados do modelo pode crescer exponencialmente, fazendo com que o tamanho das tabelas seja um fator limitante em sistemas realísticos. A verificação simbólica de modelos é uma abordagem usada para controlar o problema de explosão de estados em que: conjuntos de estados são representados por fórmulas lógicas e a exploração do espaço de estados é feita através de operações lógicas. Tais operações podem ser realizadas de forma eficiente utilizando diagramas de decisão binária (BDDS- *Binary decision diagram*) [Bryant 1992].

Diversos planejadores que utilizam técnicas de verificação simbólica se destacaram na Competição Internacional de Planejamento (IPC)[1] de 2014. Quatro dentre os cinco melhores colocados na categoria de planejadores clássicos ótimos (*Sequential Optimal track*) eram planejadores que utilizavam BDDS [Edelkamp et al. 2015]. O planejador GAMER [Kissmann and Edelkamp 2008] que competiu na categoria de planejadores FOND, no IPC de 2008, também utiliza BDDS.

Neste trabalho apresentamos um planejador baseado em verificação simbólica de modelos e na lógica α-CTL. A semântica α-CTL permite que o planejador resolva problemas da classe FOND raciocinando formalmente sobre as ações. Diferentemente do MBP que utiliza mecanismos extra-lógicos para extrair políticas, α-CTL permite um planejamento que faça verificação enquanto sintetiza um plano [Pereira and de Barros 2008]. Este artigo está organizado da seguinte forma: na Seção 2 apresentamos os conceitos de planejamento como verificação de modelos; na Seção 3 mostramos como representar estados e ações através de fórmulas lógicas bem como as operações fundamentais de um planejador baseado em verificação simbólica de modelos; na Seção 4 descrevemos o planejador PACTL-SYM; na Seção 5 mostramos um exemplo de planejamento no Robô de

[1] http://www.icaps-conference.org/index.php/Main/Competitions

Carga; e, finalmente, na Seção 6 apresentamos as conclusões e trabalhos futuros.

2. Fundamentos

2.1. O problema de planejamento FOND

Um domínio de planejamento é uma tupla $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$, sendo que: $\mathcal{S}$ é um conjunto finito de estados do ambiente; $\mathcal{A}$ é um conjunto finito de ações que o agente pode executar; e $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ é uma função de transição de estados que leva o agente de um estado $s \in \mathcal{S}$ para um possível conjunto de estados $X \subseteq \mathcal{S}$ através de uma ação $a \in \mathcal{A}$. Um problema de planejamento, por sua vez, é definido por uma tupla $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$ em que: $\mathcal{D}$ é o domínio de planejamento; $s_0 \in \mathcal{S}$ é o estado inicial do ambiente; e $\mathcal{G} \subseteq \mathcal{S}$ é o conjunto de estados meta.

Em decorrência da existência de ações com efeitos não determinísticos, a solução para um problema FOND pode resultar em múltiplos caminhos. Neste contexto, a solução para um problema de planejamento não determinístico é uma *política* π, isto é, um mapeamento de estados em ações. A política determina a ação mais apropriada para ser executada em cada estado e pode ser classificada em três tipos: *fraca*, *forte* e *forte-cíclica*. A política fraca é uma política que eventualmente pode alcançar um estado meta. Um exemplo de política fraca para o domínio da Figura 1(b) é o conjunto $\pi = \{(s_0, a), (s_3, d)\}$; a política forte é uma política que, a despeito do não determinismo, garante alcançar um estado meta. Por exemplo, na Figura 1(b) a política $\pi = \{(s_0, b), (s_2, c), (s_3, d), (s_4, b)\}$ é uma política forte; e a política forte-cíclica é uma política que sempre alcança um estado meta pressupondo que sua execução eventualmente conseguirá sair de todos os ciclos existentes. Na Figura 1(b), $\pi = \{(s_0, b), (s_2, c), (s_3, b), (s_4, d), (s_5, e)\}$ é um exemplo de política forte-cíclica.

2.2. Verificação de modelos usando a lógica CTL

Verificação de modelos é uma técnica de verificação formal que consiste na exploração de um sistema de transição finito (modelo) a fim de verificar a validade de uma dada propriedade. Dado um modelo $\mathcal{M}$, um estado s e uma propriedade φ, um verificador de modelos verifica $(\mathcal{M}, s) \vDash \varphi$, devolvendo sucesso quando $\mathcal{M}$ satisfaz φ a partir do estado s ou um contra-exemplo, caso contrário [Clarke et al. 1999]. Uma abordagem comum em verificação de modelos é a formalização do modelo $\mathcal{M}$ através de uma estrutura de Kripke. Seja $\mathbb{P}$ um conjunto finito não vazio de proposições atômicas, uma estrutura de Kripke sobre $\mathbb{P}$ é uma tupla $\mathcal{K} = \langle \mathcal{S}, T, \mathcal{L} \rangle$ em que: $\mathcal{S}$ é um conjunto finito não vazio de estados; $T : \mathcal{S} \to 2^{\mathcal{S}}$ é uma função de transição de estados (sem considerar as ações); e $\mathcal{L} : \mathcal{S} \to 2^{\mathbb{P}}$ é uma função de rotulação de estados. A propriedade a ser verificada, por sua vez, é formalizada através de uma fórmula da lógica CTL.

A lógica CTL (*Computation Tree Logic*) é uma lógica temporal de tempo ramificado que permite especificar propriedades quantificadas sobre caminhos em uma árvore de computação. Existem quatro operadores nesta lógica: $\bigcirc$ (próximo estado), $\Diamond$ (finalmente, em um estado futuro), $\Box$ (globalmente, em todos os estados) e $\sqcup$ (uma fórmula é válida até que outra seja verdade). Todo operador temporal é precedido de um quantificador existencial $\exists$ (existe algum caminho) ou universal $\forall$ (em todos os caminhos).

Definição 1 (Sintaxe CTL) *O conjunto de fórmulas* CTL *é definido indutivamente como:* $\varphi ::= \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \bigcirc \varphi_1 \mid \forall \bigcirc \varphi_1 \mid \exists \Box \varphi_1 \mid \forall \Box \varphi_1 \mid \exists \Diamond \varphi_1 \mid \forall \Diamond \varphi_1 \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$.

A semântica de uma fórmula CTL é dada em termos de uma estrutura de Kripke. De acordo com a semântica CTL dizemos, por exemplo, que $(\mathcal{M}, s) \vDash p$ é verdadeira se o estado s é rotulado com a proposição p; e $(\mathcal{M}, s) \vDash \forall \Diamond p$ é verdadeira se todos os caminhos na estrutura de Kripke, a partir de s, finalmente alcançam um estado que satisfaz p. A Figura 2 apresenta caminhos computacionais em modelos que satisfazem fórmulas da lógica CTL (por simplicidade omitimos o operador $\bigcirc$). Por exemplo, a Figura 2(e) ilustra que todos os caminhos de $\mathcal{M}$ a partir de s, finalmente alcançam um estado que satisfaz φ.

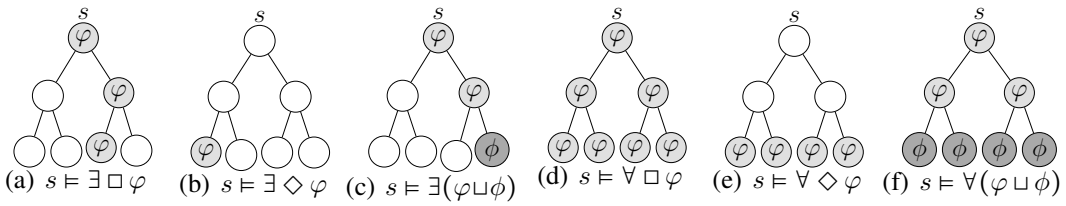


Figura 2. Semântica dos operadores na árvore computacional CTL. Nós em cinza claro representam estados do modelo que satisfazem φ e nós em cinza escuro estados que satisfazem ϕ.

Os algoritmos de verificação de modelos CTL baseiam-se na implementação de operações de pré-imagem. Dado um conjunto de estados $X \subseteq \mathcal{S}$, a operação de pré-imagem computa um conjunto de estados Y que alcançam X. Existem dois tipos de pré-imagem: a *pré-imagem fraca* (que computa os estados Y que podem alcançar X, contudo, pode existir uma ramificação a partir de algum estado de Y que não alcança X) e a *pré-imagem forte* (que computa os estados que garantidamente alcançam X).

$$\text{PREIMAGEMFRACA(X)} = \{s \in \mathcal{S} \mid T(s) \cap X \neq \varnothing\}$$
$$\text{PREIMAGEMFORTE(X)} = \{s \in \mathcal{S} \mid \varnothing \neq T(s) \text{ e } T(s) \subseteq X\}$$

Figura 3. Operações de pré-imagem de verificação de modelos em CTL.

Note que para verificar uma fórmula temporal do tipo $\exists \Diamond \varphi$, usamos a pré-imagem fraca e para a fórmula do tipo $\forall \Diamond \varphi$, usamos a pré-imagem forte. O Algoritmo 1 [Giunchiglia and Traverso 2000] apresenta uma função de um verificador de modelos que verifica se uma estrutura de Kripke satisfaz uma fórmula CTL do tipo $\exists \Diamond \varphi$ a partir de um estado inicial s. Por falta de espaço, os algoritmos que tratam os demais operadores CTL foram omitidos.

Algoritmo 1: VERIFICAEF($\mathcal{K}, s, \varphi$)

```
1  estadosAtuais ← ∅
2  proximosEstados ← ESTADOS(φ, K)  // estados da estrutura de Kripke que satisfazem φ
3  enquanto proximosEstados ≠ estadosAtuais faça
4      se s ∈ proximosEstados então
5          retorna Verdadeiro
6      fim
7      estadosAtuais ← proximosEstados
8      proximosEstados ← proximosEstados ∪ PREIMAGEMFRACA(proximosEstados)
9  fim
10 retorna Falso
```

2.3. Planejamento baseado em verificação de modelos com α-CTL

No planejamento baseado em verificação de modelos, o modelo descreve o domínio de planejamento $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$. Note que, ao invés de uma estrutura de Kripke, em planejamento temos um modelo cujas transições são rotuladas pelas ações. Enquanto um verificador de modelos verifica a validade de determinada propriedade em um modelo $\mathcal{M}$, um planejador procura um submodelo de $\mathcal{M}$ (política) que satisfaça a meta a partir do estado inicial. Assim, dado um problema de planejamento $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, $\mathcal{D}$ é um modelo de transição de estados rotulado pelas ações, e $\mathcal{G}$ é a propriedade φ que desejamos verificar no modelo a partir de s_0.

Neste contexto, os planejadores baseados em verificação de modelos que utilizam CTL necessitam de alguns mecanismos extra-lógicos para auxiliar na escolha de ações (dado que a estrutura de Kripke não contém ações). A lógica α-CTL é uma extensão de CTL que permite verificar a validade de uma propriedade e sintetizar políticas sem a necessidade de mecanismos extra-lógicos, uma vez que sua semântica leva em consideração as diferentes transições causadas pelos diferentes tipos de ações [Pereira 2007]. A lógica α-CTL é composta por quatro operadores temporais: $\odot$ (*sucessor imediato*), $\boxdot$ (*invariantemente*), $\diamondsuit$ (*finalmente*) e $\sqcup$ (*até que*) que devem, obrigatoriamente, ser precedidos por um quantificador de caminho: $\forall$ ou $\exists$ (o "ponto" nos operadores referem-se às ações).

Definição 2 (Sintaxe α-CTL) *Seja $p \in \mathbb{P}$ uma proposição atômica. A sintaxe de α-CTL é definida indutivamente como:*

$\varphi ::= \top \mid p \mid \neg p \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid \exists \odot \varphi \mid \forall \odot \varphi \mid \exists \boxdot \varphi \mid \forall \boxdot \varphi \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$.

A semântica de uma fórmula α-CTL é dada em termos de um sistema de transição de estados rotulados por ações, isto é, uma estrutura de Kripke em que as transições são rotuladas pelas ações. Por exemplo: (1) em α-CTL um modelo $\mathcal{M}$ satisfaz $\exists \odot p$ a partir de s se existe uma ação $a \in \mathcal{A}$ que, quando executada em s, possivelmente alcança um sucessor imediato s' que satisfaz p (isto é, algum efeito não determinístico de a leva ao estado s'); (2) em α-CTL o modelo $\mathcal{M}$ satisfaz $\forall \odot p$ a partir de s se existe uma ação $a \in \mathcal{A}$ que quando executada em s, necessariamente alcança um sucessor imediato s' que satisfaz p (isto é, todos os efeitos não determinísticos de a levam ao estado s').

Seja $\mathcal{M}$ um modelo de transição de estados rotulado por ações e $s \in \mathcal{S}$ o estado inicial. A Figura 4 ilustra a semântica dos operadores $\boxdot$, $\diamondsuit$ e $\sqcup$ precedidos pelo quantificador de caminho $\forall$. Note que as transições são rotuladas pelas ações `a`, `b`, `c` e `d`. Na figura 4(b), por exemplo, $s \vDash \forall \diamondsuit \varphi$ expressa que para todos os estados futuros, *alguma ação* leva a estados que finalmente satisfazem φ. Observe a diferença entre a semântica dos operadores $\Diamond$ e $\diamondsuit$ quando precedidos pelo operador $\forall$ (figuras 2(e) e 4(b) respectivamente). Em CTL a fórmula é satisfeita a partir de um estado s se todos os estados futuros, sem diferenciar as transições, finalmente alcançam um estado que satisfaz φ, enquanto em α-CTL a fórmula é satisfeita se existe uma ação em que todos os efeitos não determinísticos finalmente alcançam um estado que satisfaz φ. Uma descrição mais detalhada da semântica α-CTL pode ser encontrada em [Pereira 2007].

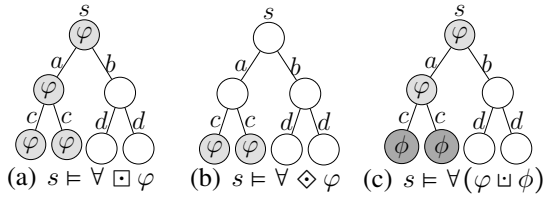


Figura 4. Semântica dos operadores $\boxdot$, $\diamondsuit$ e $\sqcup$ na árvore computacional α-CTL.

Com base na semântica da lógica α-CTL foi proposto o planejador PACTL (*Planejador Alpha* CTL) [Pereira 2007], que baseia-se na implementação de operações de pré-imagem. O cálculo da pré-imagem fraca de um conjunto de estados $X \subseteq \mathcal{S}$ (usado na síntese de política fraca) computa um conjunto de pares estado-ação que possivelmente alcançam X e o cálculo da pré-imagem forte (usado na síntese de política forte) computa um conjunto de pares estado-ação que garantidamente (a despeito do não determinismo) alcançam X. A Figura 5 apresenta as definições de pré-imagem baseadas em α-CTL.

$$\text{PRÉIMAGEMFRACA}_{actl}(\text{X}) = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A} \text{ e } \mathcal{T}(s,a) \cap X \neq \varnothing\}$$
$$\text{PRÉIMAGEMFORTE}_{actl}(X) = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A} \text{ e } \varnothing \neq \mathcal{T}(s,a) \subseteq X\}$$

Figura 5. Operações de pré-imagem α-CTL.

O Algoritmo 2 apresenta uma função do planejador PACTL que computa um submodelo $M_\pi \subseteq \mathcal{D}$ a partir do qual é possível extrair uma política fraca. Esta função recebe como parâmetro um problema FOND $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$, sendo φ a meta de planejamento $\mathcal{G}$ descrita em α-CTL. Esta função primeiramente determina o conjunto de estados que satisfazem φ (Linha 1). Em seguida, iterativamente, este conjunto é expandido através de operações de pré-imagem fraca até alcançar um ponto-fixo ($M_\pi \neq M'_\pi$), a partir do qual podemos garantir que todos os pares (s, a) do submodelo computado finalmente alcançam um estado $s \vDash \varphi$ através de algum efeito não determinístico de a. Em [Pereira 2007] são descritos algoritmos de planejamento para cada um dos operadores da lógica α-CTL.

Uma meta de planejamento pode ser especificada em α-CTL de três formas: $\exists \diamond\!\!\!\!\cdot\, \varphi$ se uma solução fraca é desejada; $\forall \diamond\!\!\!\!\cdot\, \varphi$ se uma solução forte é desejada; ou $\forall \boxdot \exists \diamond\!\!\!\!\cdot\, \varphi$ se uma solução forte-cíclica é desejada. Em [Pereira and de Barros 2008, Pereira 2007], foram tratadas fórmulas α-CTL que expressam metas de alcançabilidade estendidas, dadas por um par (φ_1, φ_2) em que φ_1 especifica uma condição que deve ser preservada ao longo da execução da política e φ_2 uma propriedade que deve ser satisfeita no final da execução da política. Por exemplo, $\forall(\varphi_1 \sqcup \varphi_2)$ expressa uma meta de alcançabilidade estendida quando uma solução forte é desejada.

Algoritmo 2: MODELOEF($\mathcal{P}$) *// síntese de políticas fracas*

```
1 M_π ← {s | s ∈ S e s ⊨ φ}  // conjunto de estados que satisfazem φ
2 M'_π ← ∅
3 enquanto M_π ≠ M'_π faça
4 |   M'_π ← M_π
5 |   C ← {s | ∃a ∈ A e (s,a) ∈ M_π}  // conjunto de estados pertencentes a M_π
6 |   I ← PRÉIMAGEMFRACA_actl(C)
7 |   M_π ← M_π ∪ I
8 fim
9 retorna M_π
```

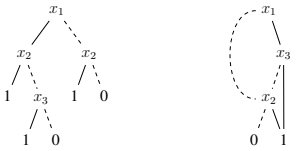
3. Planejamento baseado em verificação simbólica de modelos com α-CTL

O planejamento baseado em verificação simbólica de modelos faz buscas num espaço de conjuntos de estados descritos por fórmulas lógicas. Esses conjuntos de estados, em geral, são extremamente grandes. A representação deles usando diagramas de decisão binária resulta em uma representação compacta e que permite operações computacionalmente eficientes.

3.1. Diagramas de Decisão Binária

Diagramas de Decisão Binária (BDD - *Binary Decision Diagram*) são grafos direcionados acíclicos (DAG) que representam funções $f : \{0,1\}^n \rightarrow \{0,1\}$. Um BDD pode ser descrito como o resultado de simplificações em uma árvore de decisão. Uma árvore de decisão, por sua vez, é uma estrutura que possui nós de decisão rotulados com variáveis booleanas e nós terminais rotulados com 1 (verdadeiro) ou 0 (falso). Cada nó v de decisão possui dois nós sucessores: $l(v)$ que representa a atribuição do valor falso à variável v, e $h(v)$ que representa a atribuição do valor verdadeiro à v [Huth and Ryan 2004].

Embora árvores de decisão sejam muito úteis para representar funções booleanas, em geral, elas podem possuir muita redundância. Quando tais redundâncias são eliminadas, dá-se o nome para estrutura resultante dessas eliminações de *diagrama de decisão binária*, uma forma mais compacta para representação de fórmulas booleanas. O termo BDD, em geral, refere-se a diagrama de decisão binária reduzido ordenado (ROBDD - *Reduced Ordered Binary Decision Diagram*). Além da representação compacta, para toda expressão booleana existe um único ROBDD que o representa. Por simplicidade, quando a notação BDD é utilizada, faz-se referência a ROBDD. A Figura 6 ilustra um exemplo de uma árvore de decisão e de um BDD para fórmula $\varphi = (x_1 \vee x_2) \wedge (x_2 \vee x_3)$.



(a) Árvore de Decisão. (b) BDD.

Figura 6. Árvore de decisão para a fórmula $\varphi = (x_1 \vee x_2) \wedge (x_2 \vee x_3)$ e BDD que representa φ de forma compacta. A aresta tracejada representa $x_i = Falso$ e a aresta sólida representa $x_i = Verdadeiro$.

3.2. Regressão simbólica

Em planejamento baseado em verificação simbólica de modelos, os conjuntos de estados e ações do problema são representados através de fórmulas booleanas quantificadas (QBF - *Quantified Boolean Formulas*) [Büning and Bubeck 2009], e a busca através do espaço de estados é realizada fazendo transformações lógicas sobre as fórmulas que representam os conjuntos de estados. Seja $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$ um domínio de planejamento sobre um conjunto de proposições $\mathbb{P}$. Um estado $s \in \mathcal{S}$ é representado simbolicamente como:

$$\xi(s) = \bigwedge_{p \in \mathbb{P}(s)} p\,, \tag{1}$$

e um conjunto de estados $X \subseteq \mathcal{S}$ é representado como:

$$\xi(X) = \bigvee_{s \in X} \xi(s)\,. \tag{2}$$

Neste trabalho, ao invés de representarmos de forma simbólica o modelo de transição de estados rotulado pelas ações e proposições (como é feito pala maioria dos planejadores baseados em verificação de modelos), representamos o modelo através da descrição de ações STRIPS. Assim, podemos construir o espaço de estados apenas com a especificação de s_0, $\mathcal{G}$ e um conjunto de ações STRIPS. Uma ação STRIPS [Fikes and Nilsson 1971] $a \in \mathcal{A}$ é definida em termos de suas precondições e efeitos. As precondições da ação a, denotadas por $Precond(a)$, representam as proposições que devem ser verdadeiras no estado em que a será executada. Os efeitos, por sua vez, são subdivididos em dois conjuntos: efeitos positivos, denotados por $Ef^{+}(a)$, que representam as proposições que se

tornam verdadeiras após a execução de a; e efeitos negativos, denotados por $Ef^{-}(a)$, que representam as proposições que se tornam falsas após a execução de a. No caso das ações não determinísticas, os efeitos são representados por um conjunto de efeitos não determinísticos: $Efeitos(a) = \{e_1, ..., e_n\}$ onde cada efeito e_i possui seu próprio conjunto de efeitos positivos (Ef^{+}) e negativos (Ef^{-}).

No planejamento baseado em verificação simbólica de modelos é necessário representar as ações através de fórmulas QBF. Assim, uma ação não determinística a será representada por uma tupla $\langle \xi(Precond(a)); \xi(Efeitos(a)); \text{Mudanças}(a) \rangle$ tal que: $\xi(Precond(a))$ é a fórmula que representa $Precond(a)$ e $\xi(Efeitos(a))$ a fórmula que representa $Efeitos(a)$, respectivamente definidos pelas fórmulas 3 e 4 sendo p, t e r átomos da linguagem; e Mudanças é um conjunto de proposições que estão em $Efeitos(a)$.

$$\xi(Precond(a)) = \Big(\bigwedge_{p \in Precond(a)} p \Big). \tag{3}$$

$$\xi(Efeitos(a)) = \Big(\bigvee_{e_i \in Efeitos(a)} \Big(\bigwedge_{t \in Ef^{+}(a,e_i)} t \Big) \wedge \Big(\bigwedge_{r \in Ef^{-}(a,e_i)} \neg r \Big) \Big). \tag{4}$$

Para raciocinar sobre este modelo utilizamos operações de regressão pelas ações. A operação de regressão por uma ação $a \in \mathcal{A}$ é similar à operação de pré-imagem α-CTL (Figura 5), porém é feita sobre as fórmulas que descrevem as ações, isto é, em termos de precondições e efeitos. Com base na semântica de QBF, temos uma série de equivalências das fórmulas booleanas, dentre elas, duas que valem destacar e que serão utilizadas para computar a regressão são: $\forall x \varphi \equiv \varphi[0/x] \wedge \varphi[1/x]$; e $\exists x \varphi \equiv \varphi[0/x] \vee \varphi[1/x]$. Em tais equivalências, x é uma variável proposicional que ocorre na fórmula φ.

O cálculo da regressão fraca (REGRFRACA(X, a)) computa a fórmula que representa o conjunto de estados que podem alcançar o subconjunto $X \subseteq \mathcal{S}$ através da ação a. O cálculo da regressão forte (REGRFORTE(X, a)) computa a fórmula que representa o conjunto de estados que garantidamente alcançam o subconjunto $X \subseteq \mathcal{S}$ através da ação a [Menezes et al. 2014]. É possível provar que as regressões fraca e forte (Figura 7) equivalem a um passo da pré-imagem fraca e forte, respectivamente, (Figura 5) segundo a ação a [Rintanen 2008].Note que, nas operações de regressão fraca e forte os quantificadores $\exists$ e $\forall$ são da lógica QBF.

$$\text{REGRFRACA}(X, a) = \xi(Precond(a)) \wedge \exists \textit{Mudanças}(a).(\xi(X) \wedge \xi(Efeitos(a)))$$
$$\text{REGRFORTE}(X, a) = \xi(Precond(a)) \wedge \forall \textit{Mudanças}(a).(\xi(Efeitos(a)) \rightarrow \xi(X))$$

Figura 7. Operações simbólicas de regressão na lógica QBF.

4. O planejador PACTL-SYM

O PACTL-SYM (*Planejador Alpha*-CTL *Simbólico*) é um planejador baseado em verificação simbólica de modelos. Dado um problema de planejamento FOND $\mathcal{P} = \langle \mathcal{D}, s_0, \phi \rangle$ definido sobre um conjunto de proposições $\mathbb{P}$, em que $\mathcal{D}$ é dado pelas ações representadas conforme as Equações 3 e 4 e ϕ é a meta de planejamento especificada em α-CTL, o planejador, num primeiro momento, computa um submodelo $M_\pi \subseteq \mathcal{D}$. Se $s_0 \subseteq M_\pi$ então uma política π, extraída de M_π, é devolvida, caso contrário o planejador devolve falha.

O planejador PACTL-SYM raciocina sobre conjuntos de estados (representados conforme a Equação 2) e explora o espaço de estados utilizando operações de regressão fraca e forte (Figura 7). Computacionalmente os estados e ações são representados utilizando BDDS, sendo que cada proposição do problema é uma variável do BDD.

O Algoritmo 3 apresenta a função do planejador que computa um submodelo de uma solução fraca ($\exists \diamond \varphi$). A cada iteração i do algoritmo é computada a regressão por todas as ações $a \in \mathcal{A}$ e o resultado armazenado em uma camada i de estados e ações (par de BDD representando os estados computados pelas regressões e conjunto de ações que geraram estes estados). A primeira camada do submodelo armazena um BDD que representa os estados que satisfazem $\mathcal{G}$, a partir desta camada o algoritmo expande o submodelo, através do cálculo de regressão, de forma que cada nova camada possui um conjunto de estados que pode alcançar o conjunto de estados da camada anterior. O algoritmo para ao encontrar uma camada que contém o estado inicial ou quando alcança um ponto fixo. A divisão do submodelo em camadas permite uma representação bastante compacta dos conjuntos de estados e faz com que as operações da busca sejam mais eficientes. Para computar a regressão, o algoritmo chama a função auxiliar REGRESSAOFRACA (Algoritmo 4). Dado um conjunto de estados $\xi(X)$ esta função computa um conjunto de estados $\xi(Y)$ que podem alcançar $\xi(X)$. Para isso, esta função aplica a operação de regressão fraca, definida na Figura 7, para cada uma das ações do problema. Além dos estados que podem alcançar $\xi(X)$, esta função retorna um conjunto de ações que ao serem aplicadas em estados pertencentes a Y levam a estados pertencentes a X. A função VERIFICAINCONSISTENCIA (Linha 4) verifica se o estado gerado é inconsistente, isto é, se proposições mutuamente exclusivas são verdadeiras no estado gerado. Caso sejam, a função retorna $\perp$, senão o próprio estado é retornado. Assumimos que o conjunto de proposições mutuamente exclusivas é dado para o algoritmo.

Algoritmo 3: MODELOEF($\mathcal{P}$) *// síntese de políticas fracas*

```
1  ξ(M_π)^BDD ← ⋁_{s⊨φ} ξ(s) // conjunto de estados que satisfazem φ
2  ξ(M'_π)^BDD ← ⊥; i = 0
3  camadas_i.insere(ξ(M_π)^BDD, ∅)
4  enquanto ξ(s_0) ⊈ ξ(M_π)^BDD ou ξ(M_π)^BDD ≠ ξ(M'_π)^BDD faça
5      ξ(M'_π)^BDD ← ξ(M_π)^BDD
6      (ξ(Y)^BDD, acoes) ← PRÉIMAGEMFRACA(P, ξ(M_π)^BDD)
7      i = i + 1;   camadas_i.insere(ξ(Y)^BDD, acoes)
8      ξ(M_π)^BDD ← ξ(Y)^BDD
9  fim
10 retorna camadas
```

Algoritmo 4: REGRESSAOFRACA($\mathcal{P}, \xi(X)^{BDD}$) *// regressão simbólica*

```
1  ξ(Y)^BDD ← ⊥ ; acoes ← [ ]
2  para a ∈ A faça
3      ξ(I)^BDD ← ξ(Precond(a)) ∧ ∃Mudanças(a).(ξ(X) ∧ ξ(Efeitos(a)))
4      ξ(I)^BDD ←VERIFICAINCONSISTENCIA(ξ(I)^BDD)
5      se ξ(I)^BDD ≠ ⊥ então
6          ξ(Y)^BDD ← ξ(Y)^BDD ∨ ξ(I)^BDD
7          acoes.insere(a)
8      fim
9  fim
10 retorna (ξ(Y)^BDD, a)
```

A função de síntese de política forte ($\forall \diamond \varphi$) é similar à função de síntese de política fraca, porém, ao invés de chamar a função auxiliar REGRESSAOFRACA, é feita uma chamada para a função auxiliar REGRESSAOFORTE que substitui a operação de regressão

fraca (Linha 3) pela operação de regressão forte conforme apresentado na Figura 7.

Se o estado inicial está contido na última camada, então existe uma solução para o problema. Para extrair uma política realizamos uma busca para frente a partir do estado inicial utilizando operações de progressão. Dado um estado s a operação de progressão, denotada por $Progr(s, a, e_i)$ (Fórmula 5), computa o estado que pode ser alcançado, segundo o efeito não determinístico e_i da ação a, quando esta é executada em s.

$$Progr(s, a, e_i) = \exists \textbf{\textit{Mudanças}}(a, e_i).(\xi(s) \wedge \xi(Precond(a))) \wedge \xi(Efeitos(a, e_i)) \,. \quad (5)$$

Dado um submodelo com n camadas, se s_0 está contido no conjunto de estados da última camada (camada n), o algoritmo computa os estados S que s_0 pode alcançar considerando apenas as ações da camada n, sendo que S é um ou mais estados contidos no conjunto de estados armazenados na camada $n - 1$. Em seguida, o algoritmo computa os estados S' que podem ser alcançados a partir de S considerando apenas as ações armazenadas na camada $n - 1$ e assim sucessivamente até que o estado meta seja alcançado. A cada passo de progressão, para cada estado gerado é associado uma ação. O conjunto de pares estado-ação computados na busca progressiva representa a política extraída a partir do submodelo computado pelo Algoritmo 3. A Figura 12(a) ilustra um exemplo esquemático do funcionamento do algoritmo de extração da política.

Neste artigo nos concentramos em descrever o funcionamento dos algoritmos que computam soluções fracas e fortes. Resumidamente, para computar a política forte-cíclica, combinamos as operações de regressão fraca e forte de forma que, num primeiro momento o algoritmo computa, através da regressão fraca, um conjunto de estados que podem alcançar a meta. Em seguida, este conjunto é contraído, através da regressão forte, de forma que pares de estado-ação que podem levar para algum estado que não está presente em nenhuma camada sejam eliminados. Desta forma, todos os estados presentes no submodelo resultante eventualmente alcançam a meta.

5. Planejamento para o problema do Robô de Carga

Para exemplificar o funcionamento do planejador PACTL-SYM vamos considerar o domínio do Robô de Carga em que um robô deve transportar n caixas de uma localização para outra. Como exemplo, vamos considerar um cenário em que temos 2 salas (sala A e sala B) e 1 caixa que deve ser transportada de uma sala B para sala A (Figura 8).

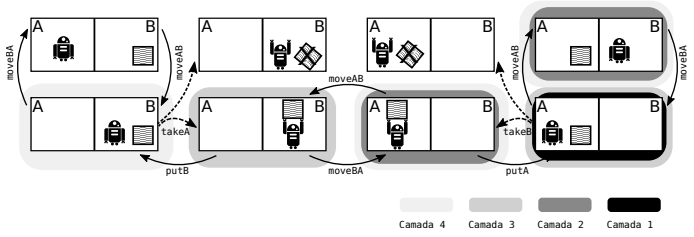


Figura 8. Domínio do robô de carga

Este problema pode ser representado pelo seguinte conjunto de proposições $\mathbb{P}$: $boxAtA$ (representa que a caixa está no chão da sala A); $boxAtB$ (a caixa está no chão da sala B); $boxOnr$ (a caixa está nos braços do robô); $boxOk$ (a caixa não está quebrada); $rAtA$ (o robô está na sala A); $rAtB$ (o robô está na sala B) e $rFree$ (o robô não está carregando nenhuma caixa). As proposições mutuamente exclusivas deste problema são: $\{rAtA, rAtB\}$ pois o robô não pode estar em duas salas ao mesmo tempo e

$\{boxAtA, boxAtB, boxOnr\}$ pois, ou a caixa está no chão de uma das salas, ou nos braços do robô. O robô pode executar seis ações, sendo elas: `moveAB`, move da sala A para sala B; `moveBA`, move da sala B para sala A; `putA`, descarrega a caixa na sala A; `putB`, descarrega a caixa na sala B; `takeA`, carrega a caixa na sala A; e `takeB`, carrega a caixa na sala B. A Figura 9 descreve as ações `moveBA`, `putA` e `takeB` de maneira simbólica. Note que, sempre que o robô tenta carregar uma caixa ele poderá derrubá-la no chão e quebrá-la ($\neg boxOk$) e assim o robô não poderá mais transportá-la para outra localização.

$moveBA:$	$\langle\ \xi(Precond(moveBA)) = rAtB; \xi(Efeitos(moveBA)) = rAtA \wedge \neg rAtB$ Mudanças = $\{rAtA, rAtB\}\rangle$
$putA:$	$\langle\ \xi(Precond(putA)) = boxOnr \wedge rAtA; \xi(Efeitos(putA)) = boxAtA \wedge rFree \wedge \neg boxOnr$ Mudanças = $\{boxAtA, rFree, boxOnr\}\rangle$
$takeB:$	$\langle\ \xi(Precond(takeB)) = boxAtB \wedge rAtB \wedge rFree \wedge boxOk;$ $\xi(Efeitos(takeB)) = (boxOnr \wedge \neg boxAtB \wedge \neg rFree) \vee \neg boxOk;$ Mudanças = $\{boxOnr, boxAtB, rFree, boxOk\}\rangle$

Figura 9. Ações simbólicas do domínio do robô de carga.

Seja o estado inicial $s_0 = (boxAtB \wedge boxOk \wedge rAtB \wedge rFree \wedge \neg boxAtA \wedge \neg boxOnr \wedge \neg rAtA)$ e $\mathcal{G} = \exists \diamond (rAtA \wedge boxAtA \wedge boxOk)$, a Figura 10 mostra o resultado da regressão fraca a partir de $\mathcal{G}$ considerando a ação `moveBA` e a Figura 11 apresenta o resultado das camadas construídas de acordo com o Algoritmo 3.

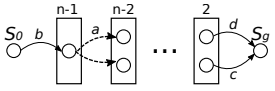
$\text{RegrFraca}(\xi(\mathcal{G}), moveBA) = \xi(Precond(moveBA)) \wedge \exists \textit{Mudanças}(moveBA).(\mathcal{G} \wedge \xi(Efeitos(moveBA)))$
$= rAtB \wedge \exists rAtA, rAtB.((boxAtA \wedge rAtA \wedge boxOk) \wedge (rAtA \wedge \neg rAtB))$
$= rAtB \wedge \exists rAtA, rAtB.(boxAtA \wedge rAtA \wedge boxOk \wedge \neg rAtB) = rAtB \wedge \exists rAtB.(boxAtA \wedge boxOk \wedge \neg rAtB)$
$= rAtB \wedge boxAtA \wedge boxOk$

Figura 10. Regressão fraca a partir de $\mathcal{G}$ segundo a ação $moveBA$.

A extração da política parte do estado inicial e realiza uma busca para frente através do cálculo de progressão (Fórmula 5), considerando apenas as ações da camada 4. A Figura 12 apresenta um exemplo de progressão a partir de s_0 segundo a ação `takeB` e o efeito não determinístico $e_i = (boxOnr \wedge \neg boxAtB \wedge \neg rFree)$. O par estado-ação resultante dessa operação é: $(\xi(s_0)$, `takeB`$)$. Quando `takeB` é executada em s_0 o robô pode ir para o estado $\xi(s) = rAtB \wedge \neg boxAtA \wedge \neg rAtA \wedge boxOnr \wedge \neg boxAtB \wedge \neg rFree$.

Camada	Conjunto de estados $\xi(X)$	Ações
1	$(boxAtA \wedge rAtA \wedge boxOk)$	$\varnothing$
2	$(boxAtA \wedge rAtB \wedge boxOk) \vee (boxOnr \wedge rAtA \wedge boxOk)$	`moveAB, putA`
3	$(boxAtA \wedge rAtA \wedge boxOk \wedge rFree) \vee (boxOnr \wedge rAtB \wedge boxOk)$	`moveAB, takeA, moveBA`
4	$(boxAtA \wedge rAtB \wedge boxOk) \vee (boxOnr \wedge rAtA \wedge boxOk) \vee$ $(boxAtB \wedge rAtB \wedge boxOk)$	`moveBA, putA, moveAB, takeB`

Figura 11. Submodelo construído pelo Algoritmo 3 para o Robô de Carga.



(a)

$Progr(s_0, takeB, e_i)$
$= \exists \textit{Mudanças}(takeB).(\xi(s) \wedge \xi(Precond(takeB))) \wedge \xi(Efeitos(takeB))$
$= rAtB \wedge \neg boxAtA \wedge \neg rAtA \wedge boxOnr \wedge \neg boxAtB \wedge \neg rFree$

(b)

Figura 12. a) Exemplo esquemático de extração da política a partir do submodelo computado. b) Estado alcançado a partir de s_0 segundo a ação `tabeB` considerando o efeito $e_i = (boxOnr \wedge \neg boxAtB \wedge \neg rFree)$.

6. Conclusão e trabalhos futuros

Um dos grandes desafios de planejamento automatizado é construir algoritmos eficientes que devolvam soluções ótimas. Nos últimos anos diversas técnicas para resolver problemas de planejamento da classe FOND foram propostas. Dentre elas, uma técnica que se

destaca é a baseada em verificação de modelos. A maioria dos planejadores dentro desta abordagem são baseados em CTL. Porém, ela não é adequada para formalizar algoritmos de síntese de política uma vez que sua semântica não considera os diferentes tipos de transições, causadas pelas diferentes ações do domínio. Neste trabalho apresentamos um planejador para problemas FOND, baseado em verificação simbólica de modelos e na lógica α-CTL, que permite construir algoritmos de síntese de maneira formal. A abordagem simbólica permite representar grandes conjuntos de estados e raciocinar sobre eles de forma eficiente com o uso de BDDS. Implementamos os algoritmos apresentados utilizando C++ e a biblioteca para manipulação de BDDS CUDD[2]. Trabalhos futuros envolvem executar testes em outros domínios e problemas e um estudo comparativo do nosso planejador com outros planejadores FOND, entre eles os MBP e o PRP.

Agradecimentos Este trabalho recebeu suporte do CNPq, CAPES-PROEX e FAPESP (projeto 2015/01587-0).

Referências

Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., e Traverso, P. (2001). MBP: a model based planner. *Workshop on Planning under Uncertainty and Incomplete Information*.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318.

Büning, H. K. e Bubeck, U. (2009). Theory of Quantified Boolean Formulas. *Handbook of Satisfiability*, páginas 735–760.

Clarke, E. M., Grumberg, O., e Peled, D. (1999). *Model checking*. MIT Press.

Edelkamp, S., Kissmann, P., e Torralba, A. (2015). BDDs strike back (in AI planning).

Fikes, R. E. e Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. IJCAI'71, páginas 608–620.

Giunchiglia, F. e Traverso, P. (2000). Planning as model checking. *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*.

Helmert, M. (2006). The fast downward planning system. *J. Artif. Int. Res.*

Huth, M. e Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press.

Kissmann, P. e Edelkamp, S. (2008). Gamer: Fully-observable non-deterministic planning via pddl-translation into a game.

Menezes, M. V., Barro, L. N., e Pereira, S. L. (2014). Symbolic regression for non-deterministic actions. *Learning and Nonlinear Models*, 12:98–114.

Muise, C. J., McIlraith, S. A., e Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. *ICAPS*.

Nau, D., Ghallab, M., e Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pereira, S. L. (2007). *Planejamento sob incerteza para metas de alcançabilidade estendidas*. Tese de doutorado, Universidade de São Paulo.

Pereira, S. L. e de Barros, L. N. (2008). A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems*, 16(3):327–344.

Rintanen, J. (2008). Regression for classical and nondeterministic planning. *European Conference on Artificial Intelligence*, páginas 568–572.

[2]http://vlsi.colorado.edu/ fabio/CUDD/html/index.html