Assistente de Provas em Lógica de Primeira Ordem utilizando Q-Learning

André Luiz Feijó dos Santos[1]

[1]Departamento de Informática – Universidade Federal de Viçosa (UFV)
36570-900 – Viçosa – MG

andre.santos1@ufv.br

Abstract. *This work presents the development of an automated prover for First-Order Logic using the Q-Learning algorithm. The aim is to combine these two approaches to guide an agent in learning to correctly deduce logical sentences and infer new knowledge based on a defined set of rules. The results of this work indicated that reinforcement learning is not only an efficient technique in terms of time but also for understanding the context of each deduction, assisting in cascading deduction series.*

Resumo. *Este trabalho apresenta o desenvolvimento de um provador automatizado em Lógica de Primeira Ordem utilizando o algoritmo de Q-Learning. A proposta é combinar essas duas abordagens para conduzir um agente a aprender a deduzir corretamente sentenças lógicas e inferir novos conhecimentos com base em um conjunto de regras definidas. Os resultados desse trabalho indicaram que o aprendizado por reforço é uma técnica eficiente não apenas em tempo, como também para conhecer o contexto de cada decução, auxiliando em séries de deduções em cascata.*

1. Introdução

A Lógica de Primeira Ordem (LPO) é uma ferramenta poderosa que permite derivar conclusões a partir de premissas. Nesse contexto, surgiram diversos softwares conhecidos como Assistentes de Prova [3], incluindo Coq, Lean, Isabelle, entre outros. Esses *softwares* aplicam algoritmos mecânicos de resolução e unificação de predicados na Forma Clausal baseados em *backtracking*, o que pode ser computacionalmente custoso. O objetivo deste trabalho é implementar um modelo a partir de aprendizado por reforço capaz de realizar deduções [1].

2. Redução ao Absurdo

A Redução ao Absurdo (*Reductio Ad Absurdum*) é um método de prova lógica que é frequentemente utilizado para demonstrar a validade ou invalidez de uma afirmação. Esse método baseia-se na suposição de que a afirmação a ser provada é falsa e, em seguida, conduz à uma contradição ou absurdo, *i.e.*, uma conclusão incompatível com outras premissas ou conhecimentos estabelecidos. Se a suposição levar a uma contradição, conclui-se que a negação da afirmação original é falsa, o que implica que a afirmação original é verdadeira. Considere o seguinte exemplo clássico de aplicação de LPO:

[1]O projeto está disponibilizado em `https://github.com/andrefeijosantos/pyfol_lib`, junto à uma documentação e resultados mais detalhados. O projeto está na sua primeira vesão, que possui algumas simplificações que não afetam a implementação do Q-Learning. Essas limitações são dedscritas na documentação presente no GitHub

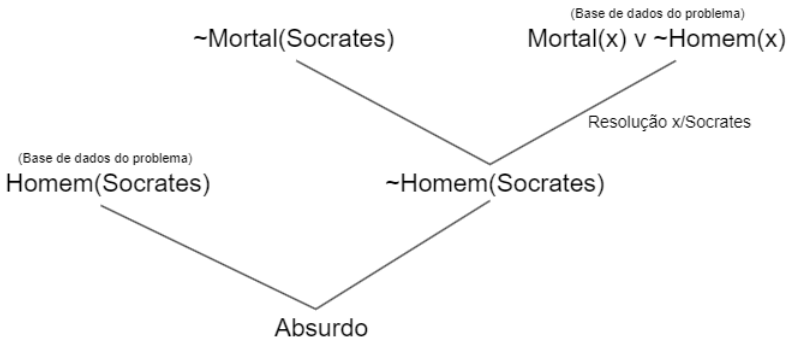

Figura 1. Exemplo de *backtracking* utilizado pelos ATPs

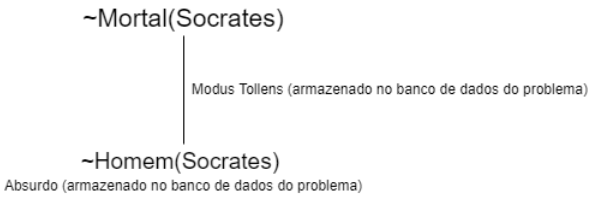

Figura 2. Exemplo de solução pelo método heurístico

Todos os Homens são mortais.
Sócrates é Homem.
Sócrates é mortal?

Este exemplo pode ser formalizado como a seguir:

$P \equiv \forall x, Homem(x) \Rightarrow Mortal(x)$
$H \equiv Homem(Socrates)$
$P, H \vdash Mortal(Socrates)$

A estrutura de prova utilizada por um ATP (*Automated Theorem Prover*) é apresentada de forma gráfica na Figura 1. A proposição $Mortal(Socrates)$ é negada e, a partir de então, são buscadas outras proposições que impliquem em uma Redução ao Absurdo, em um processo de *backtracking*. Já a Figura 2 apresenta o método de solução heurística. Nele, o banco de dados do problema não irá armazenar as proposições e predicados do contexto e, sim, as deduções que podem ser utilizadas e os vértices os quais implicam em falácias.

Em essência, os dois métodos são equivalentes. Porém, o convencional, partindo do predicado *P*, irá buscar por *Q* que ou seja possível encontrar uma fórmula do tipo $P \vee \neg P$ (e simplificá-la), ou $P \wedge \neg P$ (e, então, chegar em um absurdo). Já o método heurístico proposto aqui é "cego", *i.e.*, irá criar vizinhos de uma proposição e caminhar por eles de forma heurística até chegar em um absurdo. Evidentemente, caso não seja possível chegar no absurdo, a resposta do programa deve ser que não é possível provar (e não que a setença seja falsa). De qualquer forma, o fim do algoritmo pode ser ser dado, por:
1) *Time Out*: Não foi possível provar em tempo razoável;
2) Fim dos nós a serem visitados sem chegar a um absurdo;
3) Encerrar graciosamente indicando o caminho de prova.

3. Formulação

Seja S_P o conjunto de predicados inseridos e $G = (V, E)$ um grafo no qual V é um conjunto de vértices que representam proposições, e cada aresta $e = (v_i, v_j)$ indica que, a partir da proposição em v_i, deduz-se a representada por v_j. Seja v_1, $v_1 \in V$, o vértice de partida e S_{RAA} o conjunto de vértices finais, tal que:

$$\forall v_k, k > 1, v_k \in S_{RAA} \Leftrightarrow \exists P_i, P_i \in S_P, v_k = \neg P_i$$

É buscado, portanto, o caminho mais curto de v_1 para um vértice de S_{RAA}.

4. Metodologia de Busca e Prova Inteligente

A metodologia consiste em duas fases: uma de pré-processamento e outra de caminhar pelo grafo de decisão. Na primeira, uma estrutura de dados que mapeia uma predicado a todos os que podem ser deduzidos a partir dele é criada. Além disso, o conjunto S_{RAA} será populado com as negações das proposições de *P*. Depois, o agente inteligente explora o espaço de busca com o objetivo obter uma dedução válida.

Uma primeira forma tentar representar como um ser humano tenta derivar as proposições seria, justamente, derivando de forma aleatória, com a esperança de obter um vértice de S_{RAA} ao final. Esse método é totalmente estocástico e não é possível ter qualquer garantia de convergência, dependendo totalmente dos fatores probabilísticos de cada instância para obter sua eficiência média. Uma forma de incrementar essa metodologia de busca seria executá-la uma quantidade estipulada *n* de vezes (chamadas de episódios) e, quando a setença for provada, retornar a solução e abortar iterações não executadas, porém - ainda assim - há um algoritmo que é totalmente depente de fatores probabilísticos.

Para que o agente em questão seja de fato inteligente, o *Q-Learning* [1] acrescenta uma espécie de ”memória”a ele. Cada vez que o agente encerra uma busca, seja porque:

1. chegou em uma falácia lógica;
2. chegou em uma proposição que não pode mais deduzir nada;
3. chegou em um *Time Out*;
4. encontrou um ciclo,

ele retropropaga uma recompensa ou uma penalidade associada ao estado final. Nesse caso, apenas o ítem (1) irá propagar uma recompensa. Todos os outros irão retornar uma penalidade. Assim, ao conhecer a ”qualidade”dos nós visitados, é possível que ele não cometa sucessivos erros semelhantes.

Para essa busca com ”memória”, cinco parâmetros se fazem muito importantes:

1. α: Taxa de aprendizado, o quanto que o agente aprendeu com o episódio;
2. ϵ: Taxa de exploração, se irá explorar ou exploitar;
3. γ: Desconto de um determinado valor Q na retropropagação;
4. *R(s,a,s')* (por estado): Leve penalidade que o agente recebe logo ao chegar em um estado, força ele a preferir caminhos mais curtos;
5. *R(s,a,s')* (para estado terminal): Grande penalidade ou recompensa final, apresenta ao agente a qualidade de um caminho.

P:	P1, P2, P2, ..., Pn
Q:	Q1, Q2, Q2, ..., Qn
S:	S1, S2, S2, ..., Sn

Figura 3. Estrutura de dados que mapeia predicado atual em predicados dedutíveis

Homem:	Mortal
~Mortal:	~Homem

Figura 4. Mesma estrutura, aplicada na instância mortal-socrates

Além disso, ao contrário da segunda modelagem temporária proposta, caso sejam definidos *n* episódios, serão executados *n* episódios. Isso é uma boa estratégia quando se quer conhecer mais do espaço de busca, que é a raiz da chamada aqui de "Busca Inteligente". A ideia é que, se forem feitas deduções em sequência, é possível aproveitar resultados de deduções anteriores em posteriores. Sendo assim, uma segunda prova realizada em um contexto já desenvolvido não começa totalmente "cega", pois já conhecerá valores *Q* de alguns vértices, o que pode acarretar em maior eficiência.

5. Resultados

As métricas consideradas nesse trabalho foram, além da capacidade de realizar provas, a otimização resultante de deduções em castata e a capacidade do agente encontrar a dedução mais curta possível. Os exemplos testados estão disponíveis no repositório do projeto, junto com um link para o *Google Colaboratory* com explicações de cada caso de teste. Vale citar que todas as proposições que se desejou provar, de fato, eram verdadeiras no contexto, *i.e.* é possível construir deduções até elas. As subseções a seguir descrevem testes disponíveis no *GitHub* e os resultados obtidos.

5.1. Funcionamento do Assistente de Deduções

A instância *mortal-socrates* descreve a mesma dedução apresentada na seção Redução Ao Absurdo. Nela, são criados os predicados *Mortal* e *Homem* e a constante *Socrates*. Após isso, é assumida a proposição *Homem(Socrates)*, e sua negação, *i.e.* $\neg Homem(Socrates)$, é armazenada no conjunto S_{RAA}. Além disso, é assumida como verdade no contexto o predicado molecular $Homem(x_1) \Rightarrow Mortal(x_1)$. Como dito, as regras (condicionais) atribuídas a um ambiente de prova são armazenadas em um estrtura de dados que mapeia um predicado a todos os outros predicados que pode ser deduzidos a partir dele. As Figuras 3 e 4 apresentam como esses dados são armazenados.

Quando é, então, pedido para que o agente inteligente prove a sentença desejada, parte-se da negação da proposição que se deseja provar (nesse caso, $\neg Mortal(Socrates)$) e o agente inteligente começa a explorar as deduções que podem ser realizadas.

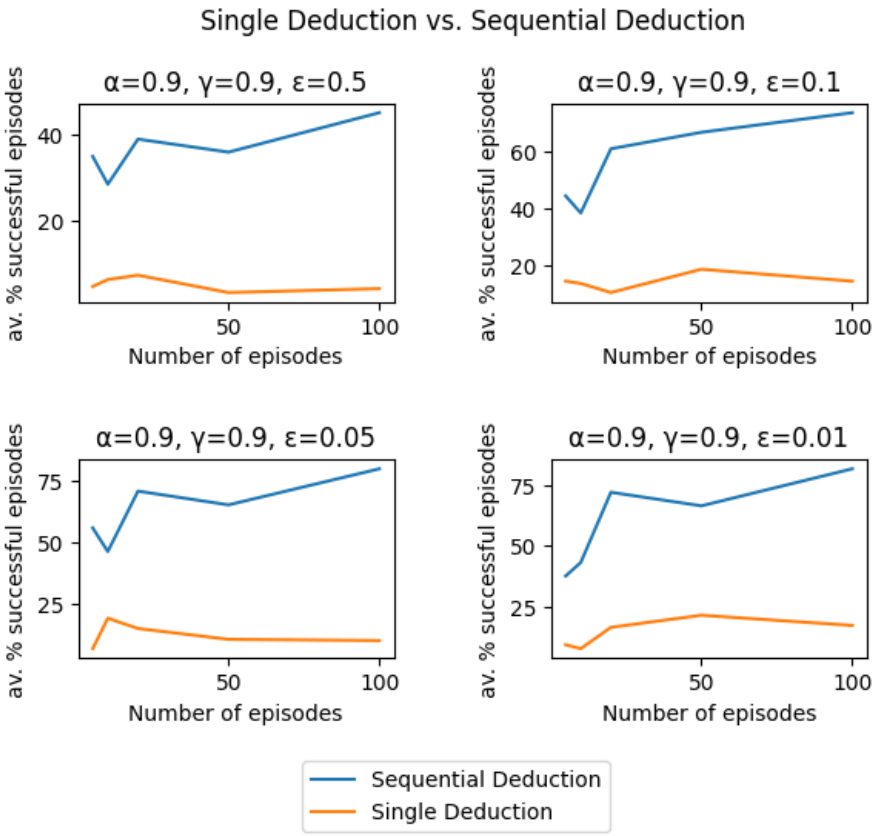

Figura 5. Gráficos da instância mortal-socrates2

5.2. Otimização em Deduções Sequenciais

A instância *mortal-socrates2* é uma dedução levemente mais robusta que *mortal-socrates*. Nessa instância, diversos outros predicados foram adicionados (alguns que contribuem para prova e outros, não); foi afirmado ¬ Irracional(Socrates) e deseja-se provar ¬ Divindade(Socrates), cuja sequência de derivações é: Divindade(Socrates) ⇒ DeusGrego(Socrates) ⇒ Imortal(Socrates) ⇒ ¬Mortal(Socrates) ⇒ ¬ Homem(Socrates) ⇒ Irracional(Socrates).

Para isso, duas abordagens foram testadas: na primeira, foi provado $Mortal(Socrates)$ e, após isso, utilizou-se desses resultados para provar $\neg Divindade(Socrates)$; na segunda, apenas desejou-se provar $\neg Divindade(Socrates)$, sem deduções intermediárias. Nesses testes, α e γ foram mantidos fixos, ambos em 0.9, pois foram valores que apresentaram resultados satisfatórios. Por outro lado, o parâmetro ϵ foi variado dentro do conjunto de opções $\{0.5, 0.1, 0.05, 0.01\}$. Cada um desses valores foram testados 100 vezes em 5, 10, 20, 50 e 100 episódios e foram colhidas as porcentagens médias de episódios com sucesso, *i.e.* episódios com deduções válidas.

Os gráficos a seguir apresentados os resultados obtidos dos testes. No eixo horizontal, estão as quantidades de episódios executados; no vertical, a porcentagem média de episódios realizados com êxito. As linhas azuis apresentam os resultados obtidos para deduções iterativas e as laranjas, para deduções únicas.

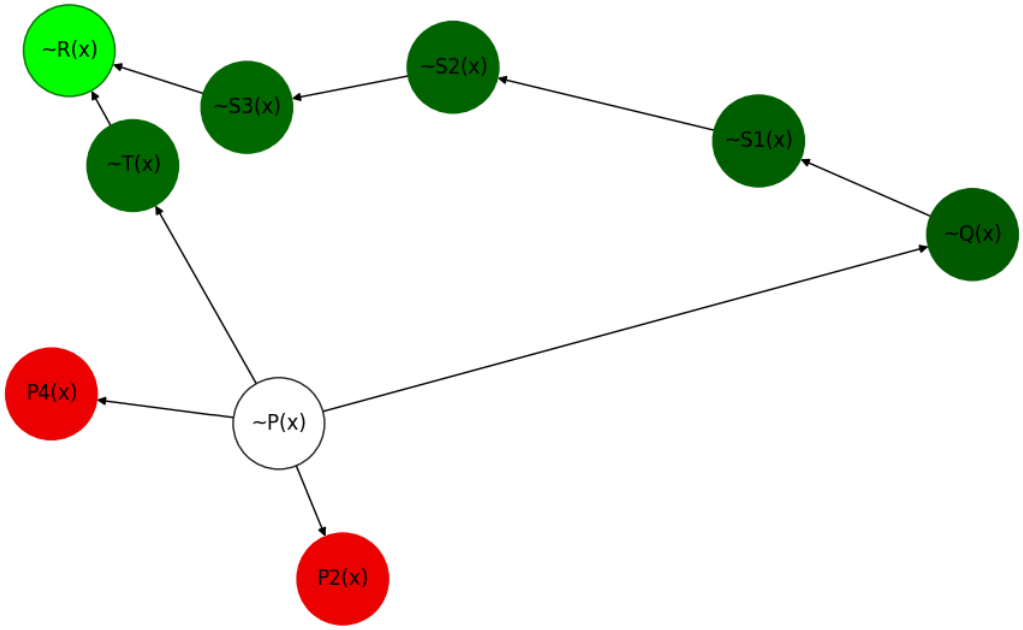

Figura 6. Grafo de deduções da instância shortest-path

5.3. Busca de Caminho Mínimo para Dedução

É interessante que um assistente de deduções consiga apresentar a dedução de menor esforço possível para o usuário. A instância *shortest-path* apresenta o comportamento do agente inteligente diante duas possibilidades de dedução, com uma exigindo maior quantidade de passos de outra.

A Figura 6 apresenta o grafo de deduções formado pelo agente inteligente. Note que $\neg R(x)$ é uma falácia lógica nesse contexto que pode ser deduzida pela sequências:

1. $P(x) \Rightarrow \neg T(x) \Rightarrow \neg R(x)$;
2. $P(x) \Rightarrow \neg Q(x) \Rightarrow \neg S_1(x) \Rightarrow \neg S_2(x) \Rightarrow \neg S_3(x) \Rightarrow \neg R(x)$.

Como nesse exemplo não existem estados dedutíveis a partir $T(x)$, $\neg Q(x)$, $\neg S_1(x)$, $\neg S_2(x)$, $\neg S_3(x)$, ou $\neg R(x)$, para todo teste em que ambos os caminhos são descobertos, os valores $V(\neg T(x)) = \sum T(\neg T(x), a, s')[R(\neg T(x), a, s') + \gamma V(s')]$ e $V(\neg Qx))$ (pela mesma equação) valem 4.5 e 0.59, respectivamente. Sendo assim, o agente inteligente considera como política seguir para $\neg T(x)$ quando se encontra em $\neg P(x)$.

5.4. Outras Oservações

As subseções anteriores apresentaram, detalhadamente, o funcionamento do agente inteligente desenvolvido em instâncias desenvolvidas com objetivos de verificar determinadas funcionalidades. Por fim, vale citar que outros testes estão diponíveis no repositório do projeto, os quais podemos verificar a execução de cada um. A Tabela 1 apresenta dados recolhidos de 100 execuções de cada um. Os dados de cada teste estão presentes em suas legendas. Nela, μ_p indica a média de execuções em que a sentença foi, de fato, provada, *i.e.* por pelo menos um episódio; μ_e, a porcentagem média de episódios com êxito e μ_n, o número médio de nós visitados e n, o máximo de nós que podem ser explorados a partir da negação da proposição que se deseja deduzir (sem contar o inicial). A instância

Resultados das Instâncias do Repositório							
Episódios	Instância	Tempo (ms)	μ_e (%)	μ_p (%)	μ_n	n	Tipo de Prova
-	*mortal-socrates*	0.000	100.00	100.00	1	1	Única
10	*mortal-socrates*2_1	0.000	7.19	31.00	5.56	6	Única
	mortal-socrates2_2	0.001	17.60	83.00	5.81	6	Sequencial
	shortest-path	0.000	63.00	100.00	5.42	9	Única
	example-4_1	0.001	8.29	44.00	11.93	16	Única
	example-4_2	0.001	30.39	95.00	13.53	16	Sequencial
20	*mortal-socrates*2_1	0.001	10.74	45.00	5.66	6	Única
	mortal-socrates2_2	0.002	17.20	97.00	5.97	6	Sequencial
	shortest-path	0.001	64.50	100.00	6.22	9	Única
	example-4_1	0.003	16.20	80.00	13.29	16	Única
	example-4_2	0.003	38.25	100.00	14.68	16	Sequencial
50	*mortal-socrates*2_1	0.004	11.07	61.00	4.88	6	Única
	mortal-socrates2_2	0.005	16.33	100.00	6	6	Sequencial
	shortest-path	0.004	63.00	100.00	6.87	9	Única
	example-4_1	0.007	27.37	100.00	14.24	16	Única
	example-4_2	0.007	43.83	100.00	16.00	16	Sequencial
100	*mortal-socrates*2_1	0.008	17.55	81.00	8.97	6	Única
	mortal-socrates2_2	0.009	16.83	100.00	6	6	Sequencial
	shortest-path	0.008	64.79	100.00	7	9	Única
	example-4_2	0.015	30.24	100.00	14.73	16	Única
	example-4_1	0.014	45.81	100.00	16.00	16	Sequencial

Tabela 1. Resultados obtidos com α = 0.9; γ = 0.9 e ϵ = 0.5

mortal-socrates foi deixada separada, pois - por se tratar de um caso muito simples - obteve sempre os mesmos resultados.

6. Conclusão

A ferramenta desenvolvida se mostrou simples de implementar, além de eficiente nos casos testados. De fato, os resultados apresentados pelo agente mostram que deduções em sequências retornam resultados mais interessantes que deduções únicas, especialmente quando há muitos estados a serem explorados. Sendo assim, vale observar que o agente desenvolvido se mostrou aplicável principalmente em softwares de dedução interativa, que prestam assistência no desenvolvimento de uma prova formal. Em suma, é interessante que esta abordagem de dedução seja ainda mais explorada e aperfeiçoada para que sejam abertas novas possibilidades de resolução heurística e eficiente de deduções formais por meio de Q-Learning em lógica de ordem zero; de primeira ordem ou até de ordem superiores.

Referências

[1] Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, 279–292 (1992). https://doi.org/10.1007/BF00992698

[2] NAWAZ, M. S. et al. A Survey on Theorem Provers in Formal Methods. Disponível em:¡https://arxiv.org/abs/1912.03028¿. Acesso em: 5 jul. 2023.

[3] Filip Marić A SURVEY OF INTERACTIVE THEOREM PROVING. [s.l: s.n.]. Disponível em: ¡http://elib.mi.sanu.ac.rs/files/journals/zr/26/zrn26p173-223.pdf¿. Acesso em: 5 jul. 2023.