Inteligência Artificial

Raoni F. S. Teixeira

Aula 5 - Agentes Lógicos

1 Introdução

Esta aula apresenta agentes lógicos e seu uso para resolver problemas com base em fatos e regras.

Diferentemente dos métodos de otimização tradicionais estudados anteriormente, agentes lógicos encontram soluções usando uma abordagem declarativa: o programador especifica **o que** deve ser feito, sem precisar detalhar **como** executar as tarefas.

2 Agente Lógico

Um agente lógico toma decisões usando conhecimento formalizado e regras. Ele utiliza um motor de inferência para deduzir novas informações a partir de fatos conhecidos, resolvendo problemas ou executando ações.

Os três componentes principais de um agente lógico são:

- **Base de Conhecimento (KB)**: Armazena fatos de forma estruturada.
  - Exemplo: "A água ferve a 100°C"; "Se está chovendo, o chão está molhado."
- **Regras de Inferência**: Permitem derivar conclusões a partir dos fatos.
  - Exemplo: "Se A implica B, e A é verdadeiro, então B também é verdadeiro."
- **Motor de Inferência**: Aplica as regras para deduzir novos fatos ou resolver problemas.

A Figura 1 ilustra como o motor de inferência usa fatos e percepções para atualizar a base de conhecimento.

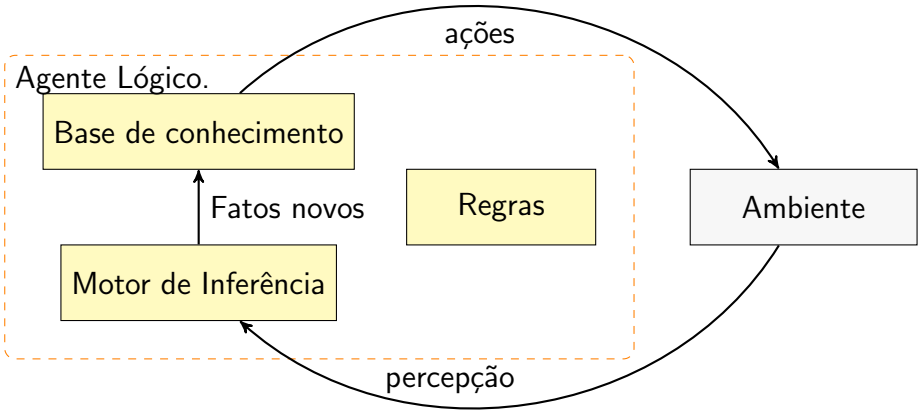

Figura 1: Agente Lógico usando as percepções e o que já é conhecido para inferir novos fatos e operar no ambiente.

3 Exemplo: Mundo de Wumpus

No clássico Mundo do Wumpus, um agente explora uma grade 4×4 em busca de ouro, enquanto evita perigos como o Wumpus (um monstro) e poços. O agente utiliza percepções para deduzir a localização de perigos e ouro:

- **Fedor**: Indica a proximidade do monstro Wumpus.
- **Brisa**: Indica a proximidade de um poço.
- **Brilho**: Indica a presença de ouro na célula.

A Figura 2 mostra um tabuleiro. O agente começa no canto inferior esquerdo (1,1) e utiliza percepções para deduzir onde estão os perigos e o ouro, aplicando as seguintes regras:

- Se há *fedor* em uma célula, o Wumpus está em uma célula adjacente.
- Se há *brisa* em uma célula, um poço está em uma célula adjacente.
- Se há *brilho*, o ouro está na mesma célula.

Enquanto explora, o agente atualiza sua base de conhecimento. Por exemplo:

1. Ao não detectar *fedor* ou *brisa* em (1,1), conclui que (1,2) e (2,1) são seguras.
2. Ao explorar (2,1) e perceber *fedor*, deduz que o Wumpus está em (2,2) ou (3,1).

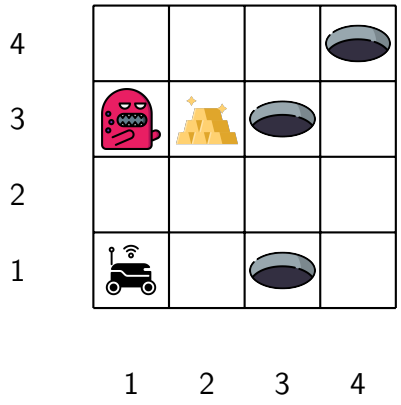

Figura 2: Mundo do Wumpus: o agente deve evitar poços e o monstro enquanto procura ouro.

3. Ao avançar para (1,2) e sentir *brisa*, confirma que (2,2) é segura e que o Wumpus está em (3,1).

Mais tarde, ao encontrar *brilho* em (3,2), o agente pega o ouro e retorna ao ponto inicial, utilizando o mesmo raciocínio lógico.

4 Lógica e Inferência

Os fatos e regras que permitem ao agente operar no ambiente são representados por meio da **Lógica de Primeira Ordem** (LPO). Diferentemente da lógica proposicional, que trabalha apenas com sentenças atômicas, a LPO introduz predicados com argumentos. Expressões como $P(t_1, \ldots, t_n)$ descrevem propriedades de objetos ou relações entre eles.

Por exemplo:

- Em $P(a)$, o predicado P indica uma característica do objeto a.
- Em $R(a, b)$, o predicado R expressa uma relação entre a e b.

Predicados organizam o conteúdo lógico das afirmações, mas não constituem proposições completas por si só — para isso, é preciso especificar os elementos envolvidos.

Nesse contexto, as **variáveis** (como x, y) funcionam como espaços reservados para elementos ainda não identificados. Elas contrastam com as **constantes** (como a, b), que se referem a objetos específicos. Assim:

- $P(x)$ pode significar "x é verde".

- $P(a)$ afirma de modo particular que a é verde.

A LPO também introduz dois quantificadores:

- **Quantificador universal** ($\forall$): afirma que uma proposição vale para todos os elementos de um domínio. Exemplo: $\forall x\, P(x)$ significa "todo x possui a propriedade P".
- **Quantificador existencial** ($\exists$): expressa que há pelo menos um elemento que satisfaz a condição. Exemplo: $\exists x\, P(x)$ significa "existe um x que possui a propriedade P".

Com esses elementos — predicados, variáveis, constantes e quantificadores — podemos construir afirmações precisas e genéricas sobre domínios estruturados.

A LPO também permite realizar inferências. Entre as regras mais comuns estão:

- **Modus Ponens**:
  - Se $A \to B$ e A é verdadeiro, então B é verdadeiro.
  - Exemplo:
    - Se está chovendo, então o chão está molhado.
    - Está chovendo.
    - Logo, o chão está molhado.
- **Modus Tollens**:
  - Se $A \to B$ e $\neg B$ é verdadeiro, então $\neg A$ é verdadeiro.
  - Exemplo:
    - Se há fogo, então há fumaça.
    - Não há fumaça.
    - Logo, não há fogo.

A inferência pode seguir dois caminhos principais:

Encadeamento para frente: parte de fatos conhecidos e aplica regras para deduzir novas conclusões. No mundo do Wumpus (Figura 2), por exemplo, podemos usar o *modus tollens* para concluir que a célula (2,2) é segura.

Encadeamento regressivo: parte de uma meta — como verificar se uma célula é segura — e tenta prová-la com base nas regras e fatos disponíveis.

Ao aplicar o encadeamento regressivo, o sistema segue um processo típico:

1. Busca uma regra cujo consequente corresponda à meta desejada. A estrutura geral é:

$$\text{Se premissa}_1 \wedge \text{premissa}_2 \wedge \cdots \wedge \text{premissa}_n, \text{ então conclusão.}$$

2. Examina cada premissa:
   - Se for um fato conhecido (por exemplo, "não há fedor em (1,1)"), a premissa é marcada como verdadeira.
   - Se não for conhecida, ela se transforma em uma nova meta a ser provada com base em outras regras ou fatos.

Esses mecanismos mostram como a LPO não apenas estrutura o conhecimento, mas também guia o raciocínio do agente de forma lógica e transparente. Seu poder está na combinação entre forma e função: predicados que estruturam, variáveis que generalizam, constantes que especificam e quantificadores que delimitam — todos operando em um sistema coerente de inferência.

4.1 Regras do Mundo do Wumpus em Lógica de Primeira Ordem

As regras do Mundo do Wumpus podem ser escritas em lógica de primeira ordem, permitindo generalizações sobre qualquer posição do tabuleiro. A seguir, apresentamos algumas das principais relações entre percepções e elementos do mundo.

- **Brisa indica possível poço em células adjacentes**:

$$\forall x \, \forall y \, (\text{brisa}(x, y) \rightarrow \exists x', y' \, (\text{adjacente}(x', y', x, y) \wedge \text{poço}(x', y')))$$

- **Ausência de brisa indica ausência de poço nas células adjacentes**:

$$\forall x \, \forall y \, (\neg\text{brisa}(x, y) \rightarrow \forall x', y' \, (\text{adjacente}(x', y', x, y) \rightarrow \neg\text{poço}(x', y')))$$

- **Fedor indica possível Wumpus em células adjacentes**:

$$\forall x \, \forall y \, (\text{fedor}(x, y) \rightarrow \exists x', y' \, (\text{adjacente}(x', y', x, y) \wedge \text{wumpus}(x', y')))$$

- **Ausência de fedor indica ausência de Wumpus nas células adjacentes**:

$$\forall x \, \forall y \, (\neg\text{fedor}(x, y) \rightarrow \forall x', y' \, (\text{adjacente}(x', y', x, y) \rightarrow \neg\text{wumpus}(x', y')))$$

- **Percepção de brilho indica ouro na mesma célula**:

$$\forall x \, \forall y \, (\text{brilho}(x, y) \rightarrow \text{ouro}(x, y))$$

- **Se uma célula é segura, então não contém poço nem Wumpus**:

$$\forall x \, \forall y \, (\text{segura}(x, y) \rightarrow \neg \text{poço}(x, y) \land \neg \text{wumpus}(x, y))$$

- **Uma célula é visitável se for segura e adjacente a uma célula visitada**:

$$\forall x \, \forall y \, (\text{visitável}(x, y) \leftrightarrow \text{segura}(x, y) \land \exists x', y' \, (\text{visitada}(x', y') \land \text{adjacente}(x, y, x', y')))$$

Essas representações formais são a base da implementação declarativa de agentes lógicos. Em vez de programar explicitamente cada ação do agente, descrevemos propriedades do mundo e relações entre elementos. A inferência — isto é, o raciocínio necessário para decidir o que fazer — fica a cargo da linguagem lógica.

5 Implementação em Prolog

Uma linguagem ideal para esse paradigma é o `Prolog`, que permite representar conhecimento por meio de fatos e regras. O interpretador, então, realiza inferências automáticas para responder a perguntas e tomar decisões.

5.1 Fatos e Regras em Prolog

Fatos descrevem propriedades conhecidas do ambiente:

```
brisa(2,1).              % Há brisa na célula (2,1)
adjacente(1,1,2,1).      % A célula (1,1) é adjacente à (2,1)
```

Regras definem condições sob as quais certos predicados devem ser considerados verdadeiros. Por exemplo:

```
poco(X,Y) :- brisa(A,B), adjacente(X,Y,A,B).
```

Essa regra afirma: *"(X,Y) é uma célula com poço se houver brisa em (A,B) e (X,Y) for adjacente a (A,B)"*.

5.2 O Significado do Operador :-

O operador `:-` expressa uma implicação lógica reversa:

$$\text{conclusão} \leftarrow \text{premissas}$$

Lê-se: "A conclusão é verdadeira se as premissas forem verdadeiras". Em termos lógicos, $B \rightarrow A$.

Exemplo: A regra abaixo:

```
avo(X,Z) :- pai(X,Y), pai(Y,Z).
```

significa: *"X é avô de Z se X é pai de Y e Y é pai de Z"*.

Elementos da sintaxe Prolog:

- A vírgula (',') representa conjunção lógica $\wedge$ (`and`).
- O ponto-e-vírgula (';') representa disjunção lógica $\vee$ (`or`).
- Um ponto final ('.') encerra cada cláusula.

5.3 Paradigma Declarativo

Ao programar em Prolog, focamos no **que** deve ser verdade, não em **como** alcançar essa verdade. O controle da busca é responsabilidade do interpretador.

Por exemplo:

```
poco(X,Y) :- brisa(2,1), adjacente(X,Y,2,1).
```

declara que uma célula `(X,Y)` contém um poço se:

- há brisa na célula `(2,1)`, e
- `(X,Y)` é adjacente a `(2,1)`.

5.4 Consultas

Com fatos e regras definidos, podemos fazer perguntas ao interpretador:

```
?- poco(X,Y).
```

Essa consulta solicita: *"Para quais valores de (X,Y) a presença de um poço pode ser inferida?"*

O Prolog busca satisfazer as premissas da regra associada ao predicado `poco/2`, retornando todos os pares possíveis que tornam a proposição verdadeira.

6 Problemas de Satisfação de Restrições

Problemas de Satisfação de Restrições (CSPs, do inglês *Constraint Satisfaction Problems*) são comuns em inteligência artificial. Neles, o objetivo é atribuir valores a variáveis de forma que certas restrições sejam satisfeitas. Entre os exemplos clássicos estão a coloração de mapas, o Sudoku, o agendamento de tarefas e o problema das 8 rainhas.

6.1 O Problema das 8 Rainhas

Nesse problema, buscamos posicionar 8 rainhas em um tabuleiro 8×8 de forma que nenhuma ataque outra. Ou seja, não pode haver duas rainhas na mesma linha, coluna ou diagonal.

O problema pode ser modelado como um CSP:

- Cada variável representa a posição da rainha em uma linha.
- O domínio de cada variável é o conjunto de colunas, de 1 a 8.
- As restrições impedem conflitos entre as rainhas.

Em Prolog, uma solução pode ser representada como uma lista: o i-ésimo elemento indica a coluna onde está a rainha da i-ésima linha. Por exemplo, a lista `[1, 5, 8, 6, 3, 7, 2, 4]` representa uma disposição onde nenhuma rainha se ataca.

Abaixo está uma implementação simples do problema em Prolog:

```
% Verifica se nenhuma rainha ataca outra
seguras([]).
seguras([Q|Outras]) :-
    seguras(Outras),
    segura(Q, Outras, 1).

% Verifica se Q está segura em relação às demais
segura(_, [], _).
segura(Q, [Q1|Outras], Dist) :-
    Q =\= Q1,
    abs(Q - Q1) =\= Dist,
    D1 is Dist + 1,
    segura(Q, Outras, D1).

% Gera permutações possíveis das colunas e testa segurança
solucao(Tabuleiro) :-
    permutation([1,2,3,4,5,6,7,8], Tabuleiro),
    seguras(Tabuleiro).
```

Essa solução mostra bem a força do paradigma declarativo de Prolog. Não programamos os passos da busca; apenas descrevemos as condições que definem uma solução. O predicado:

- `permutation/2` gera todas as permutações das colunas — evitando conflitos nas colunas.
- `seguras/1` verifica se não há ataques diagonais entre rainhas.
- `segura/3` testa se uma rainha está a uma distância segura das outras.

7 Discussão Final

Os agentes lógicos fazem parte da história da Inteligência Artificial. Entre as décadas de 1950 e 1980, eles representaram a principal abordagem para construir máquinas capazes de raciocinar. Essa visão nasceu da lógica matemática e da filosofia analítica, e foi formalizada por pioneiros como John McCarthy, Marvin Minsky, Allen Newell e Herbert Simon. Seu objetivo era claro e ousado: programar computadores para pensar como humanos, usando regras e fatos declarativos [McC59, NS56].

Esse paradigma, conhecido como simbólico ou "boa e velha IA" (*Good Old-Fashioned AI*), levou à criação dos primeiros sistemas especialistas, como o MYCIN na medicina [**?**]. Nessas aplicações, a lógica de primeira ordem (LPO) ofereceu precisão e generalidade. Com ela, era possível expressar fatos isolados e regras universais com quantificadores, o que ampliava o alcance do raciocínio automático. O exemplo do *Mundo do Wumpus*, por sua vez, permanece uma referência didática valiosa para demonstrar como agentes lógicos operam em ambientes parciais e incertos.

No entanto, a abordagem lógica logo revelou seus limites. A construção manual de bases de conhecimento é trabalhosa, a inferência pode ser computacionalmente cara, e a lógica clássica falha diante de incertezas comuns no mundo real. Além disso, os sistemas simbólicos eram frágeis e pouco adaptáveis. O fracasso de projetos ambiciosos nos anos 1980 levou ao chamado *inverno da IA*, um período de estagnação e ceticismo.

A partir dos anos 1990, métodos conexionistas — baseados em aprendizado estatístico e redes neurais — passaram a dominar a pesquisa em IA. Redes neurais e técnicas de aprendizado de máquina permitiram que a IA aprendesse diretamente com dados, evitando a necessidade de modelar o mundo explicitamente. E é isso que estudaremos nas próximas aulas.

8 Exercícios

1. Escreva em lógica de primeira ordem a seguinte regra: *"Se uma célula é segura, então todas as células adjacentes sem brisa nem fedor também são seguras."*

2. Modele em Prolog os seguintes fatos e regras:
   - Há brisa em (2,1), e fedor em (1,2);
   - Células adjacentes seguem a topologia de grade 4x4.
   - Escreva uma regra que infere a possibilidade de poço ou Wumpus nas células adjacentes.
3. Faça uma consulta em Prolog para saber quais células são potencialmente perigosas, com base nos fatos fornecidos.
4. Adapte o código das 8-rainhas para resolver o problema das **N rainhas**, generalizando o tabuleiro para qualquer valor de $N \geq 4$. Dica: use listas de tamanho N e o predicado `numlist/3` para gerar os domínios.

Referências

[McC59] John McCarthy. Programs with common sense. 1959.

[NS56] Allen Newell and Herbert A. Simon. The logic theory machine: A complex information processing system. *IRE Transactions on Information Theory*, 1956.