AGENTES LÓGICOS: AGENTES QUE RACIOCINAM DE FORMA LÓGICA

4.1 INTRODUÇÃO:

Iremos discutir representação de conhecimento e processos de raciocínio sobre conhecimentos da vida $\Rightarrow$ campo da IA.

Agente baseado em conhecimento pode combinar conhecimento com estimulo corrente para inferir sobre aspectos escondidos do estado corrente para priorizar uma determinada ação durante o processo de seleção.

A razão final para estudar agentes baseados em conhecimento: flexibilidade (capacidade para lidar com novas tarefas, armazenar novos conhecimentos sobre a ambiente, capacidade de adaptar a mudanças no ambiente e atualizar o conhecimento associado).

Agente baseado em conhecimento precisa saber:

- O estado atual do mundo;
- Como inferir propriedades do mundo não captadas pelos seus sensores;
- Como o mundo evolui em função do tempo;
- O que ele necessita para alcançar os objetivos;
- O que suas ações realizam em várias circunstâncias.

4.2 UM AGENTE BASEADO EM CONHECIMENTO:

O componente central de um agente baseado em conhecimento é a sua base de conhecimento (*Knowledge base - KB*). Informalmente, uma KB é um conjunto de representações de fatos do mundo. Cada representação individual é chamada de sentença. As sentenças são expressas em uma linguagem de representação de conhecimento.

Devem existir mecanismos para adicionar novas sentenças a base de conhecimento e para consultar o que é sabido (conhecido): *TELL* e *ASK* (nome padrão).

Quando uma consulta *ASK* for feita a base de dados, a resposta deve seguir o que foi "dito" anteriormente (*TELL*).

Determinar respostas a partir do que foi "dito" anteriormente faz parte do ***mecanismo de inferência***: outro componente principal de um agente baseado em conhecimento.

```
function KB_AGENT (percept) return action
      static: KB, a Knowledge base
            t, a counter, initially 0, indicating time

      TELL (KB, MAKE_PERCEPT_SENTENCE (percept t))
      action ← ASK (KB, MAKE_ACTION_QUERY(t))
      TELL (KB, MAKE_ACTION_SENTENCE (action t))
      t ← t + 1
      return action
```

Onde:

- *MAKE_PERCEPT_SENTENCE:* cria uma sentença a partir do que foi percebido pelo agente em um dado instante de tempo.
- *MAKE_ACTION_QUERY:* retorna que sentença será perguntada em um dado momento (*ASK*).

⇒ Os detalhes do mecanismo de inferência estão encapsulados nas funções *TELL* e *ASK*.

Um agente baseado em conhecimento pode ser considerado semelhante a um agente com estado interno, visto anteriormente. No entanto, o primeiro não é um programa que calcula ações baseado somente em um estado interno variável.

Um agente baseado em conhecimento pode ser descrito em três níveis:

- Nível de conhecimento: é o nível mais abstrato, nós podemos descrever o agente dizendo o que ele conhece. Por exemplo, um agente motorista de táxi poderia saber que o Elevado Castelo Branco liga a Av: Bias Fortes a Pedro II. Se as funções *TELL* e *ASK* trabalharem corretamente, nós podemos usar o nível de conhecimento sem se preocupar com os níveis mais baixos;
- Nível lógico: nível onde o conhecimento é codificado em sentenças. Por exemplo, o agente motorista de táxi poderia ter em sua base de conhecimento a seguinte sentença: Liga(ECBranco, BF, PII).
- Nível de implementação: é o nível que mantém a arquitetura do agente, ou seja, a representação física das sentenças do nível lógico. Ex.: Liga(ECBranco, BF, PII) pode ser representada em uma base de conhecimento por uma lista de string, por uma entrada positiva (flag 1) numa tabela tridimensional, etc... A escolha da representação é muito importante para o desempenho do agente mas é irrelevante para o nível lógico e de conhecimento.

É possível construir um agente baseado em conhecimento "dizendo" o que ele precisa saber ⇒ abordagem declarativa (o conhecimento do especialista humano é capturado de forma declarativa).

Pode-se também criar mecanismos de aprendizagem a partir do que foi percebido no ambiente (série de estímulos).

4.3 O MUNDO DO WUMPUS

A principal motivação para usarmos o mundo do Wumpus como exemplo é possibilidade de desenvolvermos raciocínio lógico.

O Mundo do Wumpus é um jogo baseado num agente que explora uma caverna onde existe em alguma posição um wumpus (animal que devora qualquer um que entrar em seu ambiente). Além disso, alguns ambientes têm armadilhas e em um deles tem um amontoado de ouro, que deve ser apanhado pelo agente.

ESPECIFICANDO O PROBLEMA:

Semelhante ao ambiente do aspirador de pó, o mundo do Wumpus é uma matriz de quadrados onde cada posição pode conter agentes e objetos. O agente começa na posição [1,1], tem que achar o ouro e retornar para a posição [1,1] sem ser morto pelo wumpus (sem entrar na mesma posição).

Os estímulos possíveis são:

- Nas posições adjacentes (não diagonal) ao wumpus, agente perceberá cheiro (stench).
- Nas posições adjacentes (não diagonal) a armadilha, agente perceberá brisa (breeze).
- Na posição que tem ouro: agente perceberá brilho.
- Quando o agente for em direção ao muro ele perceberá a batida.
- Quando o wumpus morre, percebe-se um grito de qualquer lugar da caverna.

⇒ Assim, o que é percebido por um agente em um dado momento é representado por uma lista de cinco elementos. Exemplo: [cheiro, brisa, brilho, nada, nada], ou seja, existe cheiro, brisa e brilho mas não existe batida e grito. O agente não percebe sua localização.

As ações de movimento do agente são as mesmas do aspirador de pó, mais a ação de pegar o ouro [grab], de atirar [shoot] (só uma flecha disponível) e sair da caverna [climb].

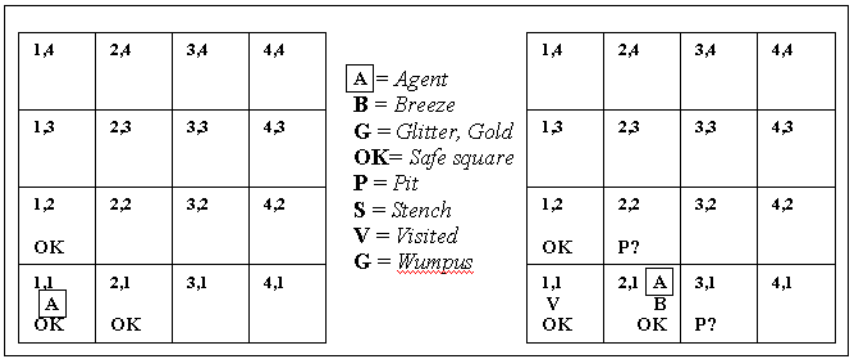
A localização do wumpus e do ouro é aleatória. São dados 1.000 pontos para cada vez que o agente consegue pegar o ouro e escapa. E 1 ponto de penalidade para cada ação tomada e 10.000 pontos para quando é morto.

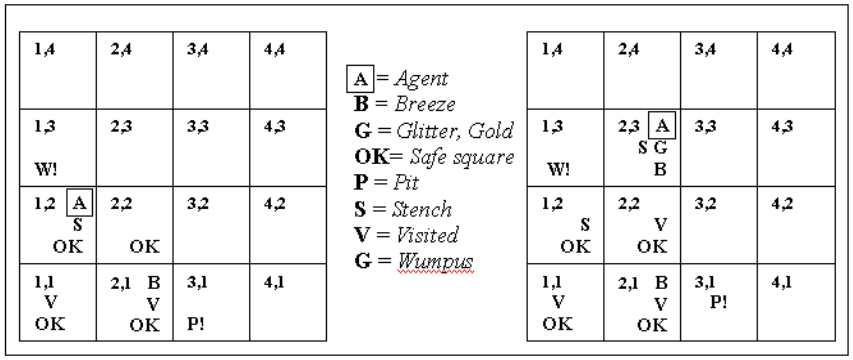
É possível aumentar a complexidade do ambiente colocando dois agentes explorando juntos (podendo se comunicar um com o outro), fazendo o wumpus se movimentar, tendo múltiplos ouros e/ou múltiplos wumpus.

AGINDO E RACIOCINANDO NO MUNDO DO WUMPUS

Temos as regras do mundo do Wumpus, mas não temos como o agente poderia agir

No primeiro passo temos a seguinte percepção: [*nada, nada, nada, nada, nada*] no segundo temos: [*nada, Brisa, nada, nada, nada*]





Como não tem cheiro nem brisa no campo [1,1] então o agente infere que [1,2] e [2,1] é seguro. O agente percebe brisa na posição [2,1] , logo ele infere que existe uma armadilha em [2,2] ou [3,1] (uma vez que em [1,1] ele já esteve e não tem). Como o quadrado [1,2] não foi visitado ainda, ele infere que é melhor visitá-lo do que arriscar. Na posição [1,2] ele nota um Cheiro, logo ele infere que na posição [1,3] possui um *Wumpus*, pois a posição [1,1] já foi visitada e a posição [2,2] não tem *Wumpus* pois se existisse, existeria cheiro na posição [2,1]. Ele também infere que a posição [2,2] não tem armadilha pois não tem vento em [1,2] e que na posição [3,1] tem uma armadilha. Assim por diante... Ao encontrar o ouro, basta ele voltar para o estado inicial pelos pontos já visitados.

Vamos estudar como os agentes lógicos realizam as inferências necessárias para realizarem uma determinada tarefa.

4.4 REPRESENTAÇÃO, RACIOCÍNIO E LÓGICA:

O objetivo da representação do conhecimento é expressar o conhecimento em uma forma computacionalmente tratável. A linguagem de representação é definida sobre dois aspectos:

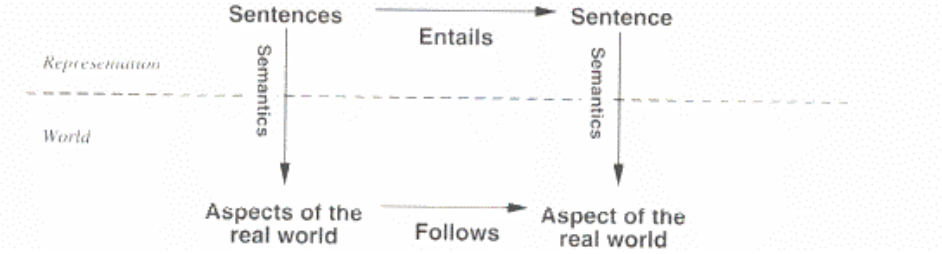
A **sintaxe** da linguagem descreve as configurações possíveis que podem construir sentenças. Normalmente a sintaxe é definida em termos de como as sentenças são representadas. Essas sentenças por sua vez derivarão em representações computacionais internas.

A **semântica** determina os fatos do mundo para os quais as sentenças se referem. A ligação entre os fatos do mundo real e as sentenças é dada pela semântica da linguagem.

$\Rightarrow$ a propriedade de um fato estar seguido de outros fatos, ou seja, ter como conseqüência outros fatos, deve ser espelhada por sentenças acarretando (*entails*) em outras sentenças.

$\Rightarrow$ o processo de gerar novas sentenças verdadeiras a partir de sentenças verdadeiras é chamando de *entailment:* utilizaremos inferência lógica.

Sentença x Fatos:



Na notação matemática, "a base de conhecimento KB *entails* α", é dado por KB |- α.

Um processo de inferência pode possuir dois comportamentos distintos:

- Dada uma base de conhecimento *KB,* ele gera novas sentenças α derivadas de *KB*; ou,
- Dada uma base de conhecimento *KB* e uma sentença α, ele diz se α pode ser derivada a partir de *KB*.

O registro de um processo de inferência que somente deriva em sentenças verdadeiras é chamado de PROVA.

REPRESENTAÇÃO:

Linguagem de Programação X Linguagem Natural:

Com linguagem de programação descrevemos algoritmos e estruturas de dados, como por exemplo o Mundo do Wumpus pode ser representado como sendo uma matriz 4x4. No entanto descrever "existe uma armadilha em [2,2] ou [3,1]" ou "existe um Wumpus em alguma posição" é complicado. Descrever informações que não se tem certeza sobre como são as coisas não é trivial. Enquanto que descrever somente informações sobre possibilidades de como elas devem ser ou como elas não devem ser, não são expressivas suficientes (linguagem natural).

$\Rightarrow$ Uma boa linguagem de representação de conhecimento deve combinar aspectos de linguagem de programação e de linguagem natural ao mesmo tempo: ser capaz de representar precisamente tudo que queremos (linguagem natural) de forma não ambígua e independente de contexto (linguagem de programação).

Algumas linguagens de representação de conhecimento foram projetadas para tentar combinar os dois aspectos. Ex.: lógica de 1º ordem.

SEMÂNTICA:

Dizemos que uma sentença significa determinada coisa de acordo com uma dada interpretação (uma sentença não significa nada por ela mesma).

Na prática todas as linguagens de representação impõem um relacionamento sistemático entre as sentenças e os fatos. O valor verdade de uma sentença depende da sua interpretação e do estado atual do mundo.

INFERÊNCIA:

Existem muitas maneiras para projetar sistemas inferências lógicas. Uma sentença é válida (tautologia) se e somente se ela é válida para todas as interpretações possíveis. Ex.: "Existe uma sujeira na posição [1,1] ou não existe uma sujeira na posição [1,1]". Já o exemplo: "Existe uma área livre na minha frente ou existe um muro na minha frente", não é válida por si mesma (depende da interpretação) $\Rightarrow$ "Se toda posição está livre ou tem um muro, então..."

Uma sentença é "satisfeita" (*satisfable*) se ela for verdadeira para pelo menos uma interpretação, e não "satisfeita" se ela for falsa para todas as interpretações. Ex.: "Existe um muro na minha frente ou não existe muro na minha frente". Programas agentes não sabem a interpretação do que está sendo utilizado, mas consegue inferir a partir da base de conhecimento.

4.5 LÓGICA

Lógica é constituída de:

1. Um sistema formal para descrever os estados das coisas: a sintaxe da linguagem descreve como fazer sentenças e a semântica como as sentenças descritas se relacionam com as coisas do mundo;

2. Uma teoria para prova, ou seja um conjunto de regras de inferência que define o *"entailments"* do conjunto de sentenças.

Iremos nos concentrar em dois tipos de lógica: Proposicional ou booleana e lógica de primeira ordem.

Na lógica proposicional, símbolos representam proposições (fatos), por exemplo *"the wumpus is dead"* deve ser uma proposição verdadeira ou não. Proposições podem ser combinadas usando conectores lógicos gerando sentenças mais complexas.

A lógica de primeira ordem representa o mundo em termos de objetos e predicados dos objetos (propriedades dos objetos e relação entre eles) e também usa conectores e qualificadores permitindo que as sentenças sejam escritas sobre várias coisas. Lógica de primeira ordem tende a ser capaz de capturar uma boa idéia do que sabemos sobre o mundo.

As lógicas variam sob dois pontos de vista: o ponto de vista ontológico, que provê a natureza da realidade, ou seja o que o mundo é constituído, e sob o ponto de vista epistemiológico, que diz quais são os possíveis estados de conhecimento (ou seja, valores possíveis para as coisas existentes no mundo), como pode ser observado na tabela abaixo:

Linguagem	**Ponto de Vista Ontológico (o que existe no mundo)**	**Ponto de Vista Epistemiológico (o que o agente acredita sobre os fatos)**
Lógica Proposicional	fatos	*true/false/unknow*
Lógica de Primeira Ordem	fatos, objetos, relações	*true/false/unknow*
Lógica Fuzzy	fatos com grau de certeza $\in$ [0,1]	*valor no intervalo*

4.6 LÓGICA PROPOSICIONAL

SINTAXE:

A sintaxe da lógica proposicional é simples: os símbolos são as constantes *true* e *false*, proposições P e Q, conectivos lógicos $\neg$, $\wedge$, $\vee$, $\Leftrightarrow$ e $\Rightarrow$ e parênteses "( )". Todas as sentenças são construídas a partir da combinação desses símbolos .

Conjunção Lógica: $P \vee Q$ Ex.: nublado e chove

Disjunção Lógica: $P \wedge Q$ Ex.: nublado ou chove

Implicação Lógica: $P \Rightarrow Q$ Ex.: se chove então nublado

chove é a premissa e nublado é a conclusão

Equivalência Lógica: $P \Leftrightarrow Q$ Ex.: eleitor se e somente se vota

Negação Lógica: $\neg P$ Ex.: não chove

Os parênteses são usados para impor uma ordem de avaliação, caso estes não existam, vale a seguinte ordem: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Ex.:$\neg P \vee Q \wedge R \Rightarrow S$ equivale a $((\neg P) \vee (Q \wedge R)) \Rightarrow S$

Gramática que define a sintaxe da Lógica Proposicional:

```
Sentence -> AtomicSentence | ComplexSentence
AtomicSentence -> True | False | Symbol
Symbol -> P | Q | R ...
ComplexSentence -> ¬ Sentence
                 | ( Sentence ∧ Sentence )
                 | ( Sentence ∨ Sentence )
                 | ( Sentence ⇒ Sentence )
                 | ( Sentence ⇔ Sentence )
```

SEMÂNTICA:

Quando uma sentença se refere a um fato verdadeiro do mundo real, então ela recebe o valor verdadeiro. Por exemplo:

Se P for "Paris é capital da França" então seu valor é *true*

Se P for "Rio de Janeiro é capital do Brasil" então seu valor é *false*.

Tabela Verdade:

As tabelas verdades podem ser utilizadas para verificar a validade de uma sentença, ou seja, dizer se uma sentença pode ser *"entailed"* a partir de outra.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$
F	F	V	F	F	V
F	V	V	F	V	V
V	F	F	F	V	F
V	V	V	V	V	V

Se todas as linhas forem verdadeiras, então a conclusão pode ser deduzida da premissa. Exemplo, P pode ser derivado (*"entailed"*) a partir de $(P \vee Q) \wedge \neg Q$, como pode ser observado na tabela a seguir.

P	Q	P∨Q	(P∨Q)∧¬Q	((P∨Q)∧¬Q)⇒P
F	F	F	F	V
F	V	V	F	V
V	F	V	V	V
V	V	V	F	V

Regras de Inferência para Lógica Proposicional:

A notação $\frac{P}{Q}$ significa que Q foi inferido a partir de P, e é o mesmo que P |- Q. a seguir são apresentadas algumas regras de inferência, ou seja padrões de inferência que podem ser aplicados para derivar algumas conclusões. O mais usado é o Modus Ponens, definido por:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Onde lê-se *"se α então β, dado que α ocorreu então pode-se concluir β"*

Modus Ponens:	$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$
And-Elimination:	$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$
And-Introduction:	$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$
Or-Introduction:	$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$
Double-Negation Elimination:	$\frac{\neg\neg\alpha}{\alpha}$
Resolution:	$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$

Complexidade da Inferência na Lógica Proposicional: para construir uma tabela verdade para verificar a inferência (prova) envolvendo *n* proposições são necessárias 2^n linhas.

4.7 UM AGENTE PARA O MUNDO DO WUMPUS

Dado a seguinte situação:

4,1	4,2	4,3	4,4
3,1	3,2	3,3	3,4
2,1 **A** S OK	2,2 OK	2,3	2,4
1,1 V OK	1,2 B V OK	1,3	1,4

Onde os seguintes estímulos foram percebidos:

$\neg$ S1,1 $\quad$ $\neg$ B1,1

$\neg$ S1,2 $\quad$ B1,2

S2,1 $\quad$ $\neg$ B2,1

O agente deve ter algum conhecimento sobre o ambiente. Como por exemplo: o agente deve saber que se não tem cheiro em uma posição então não existe Wumpus nas posições adjacentes.

Em lógica proposicional, é necessário ter regras para cada posição do mundo, mas as regras importantes no momento, e que compõem a base de conhecimento, juntamente com os estímulos percebidos, são:

R1: $\neg$ S1,1 $\Rightarrow \neg$W1,1 $\wedge \neg$W1,2 $\wedge \neg$W2,1

R2: $\neg$ S2,1 $\Rightarrow \neg$W1,1 $\wedge \neg$W2,1 $\wedge \neg$W2,2 $\wedge \neg$W3,1

R3: $\neg$ S1,2 $\Rightarrow \neg$W1,1 $\wedge \neg$W1,2 $\wedge \neg$W2,2 $\wedge \neg$W1,3

R4: S2,1 $\Rightarrow$ W3,1 $\vee$ W2,1 $\vee$ W2,2 $\vee$ W1,1

PROCURANDO O WUMPUS:

1 – Aplicando o Modus Ponens para $\neg$ S1,1 e a sentença R1, temos:

$$\neg \text{ W1,1} \wedge \neg \text{ W1,2} \wedge \neg \text{ W2,1}$$

2 – Aplicando a Simplificação de Conjunções[1] temos:

$\neg$ W1,1 $\quad$ $\neg$ W1,2 $\quad$ $\neg$ W2,1

3 – Aplicando o Modus Ponens para $\neg$ S2,1 e a sentença R2 e então aplicando Simplificação de Conjunções temos (como 1 e 2):

[1] Representado na tabela anterior como sendo *And-eliminator.*

$\neg$ W1,1 $\neg$ W2,2 $\neg$ W2,1 $\neg$ W3,1

4 – Aplicando Modus Ponens para $\neg$ S1,1 e a sentença R4, temos:

$$W3,1 \vee W2,1 \vee W2,2 \vee W1,1$$

5 – Aplicando o Silogismo Disjunto[2] onde A é W3,1 $\vee$ W2,1 $\vee$ W2,2 e B é w1,1 (do passo 4), sendo $\neg$ w1,1 derivado do passo 3, temos:

$$W3,1 \vee W2,1 \vee W2,2$$

6 - Aplicando o Silogismo Disjunto no passo anterior onde A é W3,1 $\vee$ W2,1 e B é w2,2 (do passo 5), sendo $\neg$ w2,2 derivado do passo 3, temos:

$$W3,1 \vee W2,1$$

7 – Aplicando o Silogismo Disjunto no passo anterior onde A é W3,1 e B é w2,1 (do passo 6), sendo $\neg$ w2,1 derivado do passo 2, temos:

$$W3,1 \Rightarrow \text{Existe um Wumpus !!!!! (deduzir sem saber o que é)}$$

TRADUZINDO DEDUÇÃO EM AÇÃO:

Na figura a seguir é ilustrado o algoritmo de um agente baseado em conhecimento representado com lógica proposicional.

```
function PROPOSITIONAL_KB_AGENT (percept) returns an action

    static:  KB; a knowledge base
             t  ; a counter, initially 0

    TELL (KB, MAKE_PERCEPT_SENTENCE (percept, t)
    for each action in the list of possible actions do
             if ASK (KB, MAKE_ACTION_QUERY (t, action)) then
             t ← t + 1
             return action
    end
```

Infelizmente a lógica proposicional não apresenta respostas do tipo "que ação devo tomar?". Ela apenas responde perguntas do tipo "devo ir em frente?" ou "devo virar a direita?".

[2] Representado na tabela anterior como sendo *Resolution.*

Assim devemos fazer um loop perguntando por todas as ações possíveis (se devo ou não realizá-la).

As regras apresentadas anteriormente são válidas somente para um determinado ponto naquele determinado instante de tempo. Temos que escrever regras diferentes para diferentes espaços de tempo, logo se tivermos 100 passos nós necessitaremos de 6.400 regras... Considerando que uma ação simples teria 64 regras (11 posições com 4 orientações possíveis).

$\Rightarrow$ Com lógica de primeira ordem é possível reduzir essas 6.400 regras...

4.8 LÓGICA DE PREDICADOS DE PRIMEIRA ORDEM

Diferença (ponto de vista ontológico): representa o mundo por meio de objetos (coisas com identificadores individuais) e propriedades que os diferenciam uns dos outros. Sobre esses objetos existem relações, que podem ser também funções.

Exemplos:

Objetos: pessoas, casas, números, teorias, cores, . . .

Relações: irmão de, maior que, dentro, parte, tem cor, . . .

Propriedades: verde, vermelho, quadrado, redondo, . . .

Funções: pai de, melhor amigo, um a mais que, . . .

Considere o seguinte exemplo: *"Um mais dois é igual a três"*

Objetos: um, dois, três e um mais dois[3]

Relação: igual

Função: mais (*um mais dois* é o nome do objeto que obtemos quando aplicamos a função mais para o objeto *um* e *dois*. Outro nome para este objeto é três)

"Na posições vizinhas do wumpus tem cheiro"

Objetos: Wumpus, posição

Propriedade: cheiro

Relação: vizinhança

[3] Ao aplicarmos a função *mais* para os objetos *um* e *dois*. O mesmo que *três*.

SINTAXE E SEMÂNTICA:

Em lógica de primeira ordem tem-se que toda expressão é uma sentença (que representa um fato), objetos são representados por termos, constantes, variáveis e funções são utilizados para construir **termos**, e quantificadores e predicados para construir sentenças.

Gramática da Lógica de Primeira Ordem:

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \\
&\mid \textit{Sentence Connective Sentence} \\
&\mid \textit{Quantifier Variable}, \ldots \textit{Sentence} \\
&\mid \neg \textit{Sentence} \\
&\mid (\textit{Sentence}) \\
\\
\textit{AtomicSentence} &\rightarrow \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term}, \ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\\
\textit{Connective} &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{Before} \mid \textit{HasColor} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLegOf} \mid \cdots
\end{aligned}
$$

Onde cada constante faz referência a apenas a um objeto.**Termo** é uma expressão lógica que corresponde a <u>um</u> objeto. Constantes são termos. Pode-se usar função para construir um termo. Ex: paide *(João)*. **Sentença atômica** é a composição de termos. Ex: *Irmão(Ricardo, João) e Casado(Pai(Ricardo),Mãe(João))*. Uma sentença atômica é **verdadeira** se existir uma relação entre os objetos referenciados. Já **Sentenças complexas** é a combinação de sentenças atômicas com os conectivos lógicos (mesmos da lógica proposicional)**.** Ex: *Irmão(Ricardo, João)* $\wedge$ *Filho(João, Pedro), MaisVelho(João,30)* $\Rightarrow$ $\neg$ *MaisNovo(João,30)* [4] e *MaisVelho(João,30)* $\vee$ $\neg$ *MaisVelho(João,30)* (tautologia). Para expressar um objeto em particular, nós utilizamos os predicados unários: *"Sócrates é homem"*$\Rightarrow$ *homem(Sócrates).*

Os **Quantificadores** expressam propriedades sobre um conjunto de objetos e podem ser **Universal** (todo), por exemplo *"Todo gato é mamífero"*,ou **Existencial** (existe algum objeto no mundo que satisfaz a uma determinada condição), por exemplo:

4 Não é uma Tautologia, depende da interpretação de MaisVelho e MaisNovo)

$\exists$ x (animal(x) $\wedge$racional(x)

$\exists$ x [gosta(x, Maria) $\wedge \neg$ gosta(x, Mãe(Maria))]

$\exists$ x, y (irmao(x,y) $\wedge$ gosta(x,y)

Normalmente, quando se usa o $\forall$ o conectivo principal é o $\Rightarrow$. Exemplo:

"Todo cão é mamífero" $\forall x\ [\ cão(x) \Rightarrow mamífero(x)\ ]$

"O homem é um animal racional" $\forall x\ [\ homem(x) \Rightarrow (\ animal(x) \wedge racional(x))\ ]$

Normalmente, quando se usa $\exists$ o conectivo principal é o $\wedge$, por exemplo:

"Existem homens irracionais" $\exists x\ [\ homem(x) \wedge \neg racional(x\ )\ ]$

"Alguns alunos gostam de lógica" $\exists x\ [\ aluno(x) \wedge gosta(x, logica)\ ]$

"Alguém temido por todos detesta lógica" $\exists x\ [\ \forall y\ teme(y,x)\ \wedge gosta(x, logica)\ ]$

Propriedades dos quantificadores:

$\forall x\ \forall y\ S \equiv \forall y\ \forall x\ S$	Exemplos:	
$\exists x\ \exists y\ S \equiv \exists y\ \exists x\ S$	$\exists x\ \forall y$ *ama(x,y)*	*"Existe alguém que ama todo mundo"*
$\exists x\ \forall y\ S \neq \forall y\ \exists x\ S$	$\forall y\ \exists x$ *ama(y,x)*	*"Alguém é amado por todos"*

Relação entre o $\exists$ e $\forall$:

$\forall x\ \neg$ *gosta(x,berinjela)* é equivalente a $\neg\exists x$ *gosta(x,berinjela)*, ou seja *"Todo mundo não gosta de berinjela"* é equivalente a *"Não existe alguém que gosta de berinjela*".

$\forall x$ *gosta (x, sorvete)* é equivalente a $\neg\exists x\ \neg$ *gosta (x, sorvete)*, ou seja, *"Todo mundo gosta de sorvete"* é equivalente a *"Não existe alguém que não goste de sorvete"*

Igualdade:

O símbolo de igualdade verifica se dois termos referem-se ao mesmo objeto, por exemplo:

"Ana ama uma única pessoa" $\exists x\ [ama(Ana,x) \wedge \forall y\ (ama(Ana,y) \Rightarrow x = y\ )]$

"José ama no mínimo duas pessoas" $\exists x, y\ [ama(Jose,x) \wedge ama(Jose,y) \wedge \neg\ (\ x = y\ )]$

Sentenças quantificadas e não quantificadas:

$\neg\ \forall x\ P \equiv \exists x\ \neg P$
$\neg\exists x\ P \equiv \forall x\ \neg P$
$\forall x\ P \equiv \neg\exists x\ \neg P$
$\exists x\ P \equiv \neg\ \forall x\ \neg P$

$\Rightarrow$ quantificadas

$\neg\ (P1 \wedge P2) \equiv \neg P1 \vee \neg P2$
$\neg\ (P1 \vee P2) \equiv \neg P1 \wedge \neg P2$
$P1 \wedge P2 \equiv \neg\ (\ \neg P1 \vee \neg P2)$
$P1 \vee P2 \equiv \neg\ (\neg P1 \wedge \neg P2)$

$\Rightarrow$ não quantificadas

LÓGICA DE ORDEM SUPERIOR:

Em lógica de predicados de ordem superior (*higher order logic)* podemos quantificar relações e funções, por exemplo:

$$\forall x, y\ (x = y) \Leftrightarrow (\forall p\ p(x) \Leftrightarrow p(y))$$

Significa *"para todo objeto x e y, x será igual a y se e somente se, x e y aplicado a qualquer função forem equivalentes".*

Lógica de ordem superior tem o poder de representação muito maior que lógica de primeira ordem, porém para alguns problemas não é possível construir algoritmos capazes de tratá-los (decisão) quando representado em lógica de ordem superior.

UTILIZANDO LÓGICA DE PRIMEIRA ORDEM:

Considere o seguinte exemplo: domínio de conjuntos.

Constante: ConjuntoVazio

Predicados: *Membro* e *Subconjunto.*

Funções: *Interseção, União e Adição* (adiciona um elemento ao conjunto)

1 – Os conjuntos ou são vazios ou foram construídos pela adição de alguma coisa a um conjunto (recursiva):

$$\forall s\ Conjunto(s) \Leftrightarrow (s=ConjuntoVazio) \vee (\exists x,s_2\ Conjunto(s_2) \wedge s=Adição(x,s_2))$$

2 - Não tem como decompor o conjunto vazio em um conjunto menor:

$$\neg\exists x,s\ Adição(x,s) = ConjuntoVazio$$

3 - Adicionar um elemento que já existe no conjunto não afeta nada:

$$\forall x,s\ Membro(x,s) \Leftrightarrow s=Adição(x,s)$$

4 - Somente os membros de um conjunto foram adicionados a dele (recursiva):

$$\forall x,s\ Membro(x,s) \Leftrightarrow \exists y,s_2\ \ (s=Adição(y,s_2) \wedge ((x=y) \vee Membro(x,s_2)))$$

5 - Um conjunto é um subconjunto de *S,* se e somente se todos membros deste conjunto estão contidos em *S:*

$$\forall s_1, s_2\ Subconjunto(s_1, s_2) \Leftrightarrow (\forall x\ \ Membro(x,s_1) \Rightarrow Membro(x, s_2))$$

6 - Um conjunto é igual a outro se e somente se um for subconjunto do outro e vice versa:

$$\forall s_1, s_2 \; (s_1 = s_2) \Leftrightarrow (Subconjunto(s_1, s_2) \wedge Subconjunto(s_2, s_1))$$

7 - Um objeto é membro da Interseção de dois conjuntos se e somente se ele for membro dos dois conjuntos:

$$\forall x, s_1, s_2 \; Membro(x, Interseção(s_1, s_2)) \Leftrightarrow Membro(x, s_1) \wedge Membro(x, s_2)$$

8 - Um objeto é membro da união de dois conjuntos se e somente se ele for membro de um dos conjuntos:

$$\forall x, s_1, s_2 \; Membro(x, União(s_1, s_2)) \Leftrightarrow Membro(x, s_1) \vee Membro(x, s_2)$$

4.9 INTERAGINDO COM UMA BASE DE CONHECIMENTO

Se quisermos adicionar a definição Mãe a base de conhecimento, podemos escrever:

TELL(*KB*,($\forall m,c \; Mãe(c) = m \Leftrightarrow Mulher(m) \wedge Filho(c,m)$))

Considere que o seguinte foi percebido por sensores:

TELL(*KB*,$Mulher(Ana) \wedge filho(Leo,Ana) \wedge Filho(Maria,Leo)$)

Consultando a BC...

ASK (KB, avo(Ana,Maria)

Resposta: Sim

Para perguntar quem é filho de quem, usa-se ***queries*** com variáveis existenciais, por exemplo "existe um x tal que ..." ⇒ perguntas desse tipo são resolvidas através do mecanismo de substituição (se existir mais que uma resposta possível, uma lista de substituição é retornada).

ASK(*KB*, $\exists x \; Filho(x,Ana)$)

Resposta: Sim { x / Leo }

4.10 AGENTES LÓGICOS PARA O MUNDO DO WUMPUS

Agentes baseado em simples reflexo agem de acordo com suas percepções;

Agentes baseados em modelos (estado interno) constrói uma representação interna do mundo e utiliza-a para agir;

Agentes baseados em objetivos tentam direcionar seus esforços para atingir seus objetivos.

```
function KB-AGENT (percept) returns an action
      static KB, a knowledge base
         t, a counter, initially 0, indicating time

         TELL (KB, MAKE-PERCEPT-SENTENCE(percept, t))
         action ← ASK (KB, MAKE-ACTION-QUERY(t))
         TELL(KB, MAKE-ACTION-QUERY(t)))
         t ← t + 1
   return action
```

A sentença do estímulo deve incluir tanto o estímulo como o tempo no qual ele ocorre (caso contrário o agente não conseguiria definir quando foi dito/percebido aquilo). Assim, uma sentença típica captada em `MAKE-PERCEPT-SENTENCE(percept, 5)` seria:

estimulo([Cheiro,Brisa, Brilho, - , -],5)

Considerando que as ações possíveis para o agente são *Vire(esq), vire(dir), anda, pega, larga, atira, e sai.* A função `MAKE_ACTION_QUERY(t)` produziria $\exists a$ *Ação(a,5)* e *ASK* retorna uma lista de substituição que poderia ser { *a/atira*}.

UM AGENTE DE REFLEXO SIMPLES

Regras que refletem ações instintivas. Isto é feito através de regras que ligam a percepção à ação.

Percepções:

$\forall s,b,u,c,t$ *Estimulo([Cheiro, b ,g, u, c],t)* $\Rightarrow$ *Cheiro(t)*

$\forall s,b,u,c,t$ *Estimulo([s, Brisa, g, u, c],t)* $\Rightarrow$ *Brisa(t)*

$\forall s,b,u,c,t$ *Estimulo([s, b, Brilho, u, c],t)* $\Rightarrow$ *NoOuro(t)*

Indicando a ação (reflexo): $\forall t$ *NoOuro(t)* $\Rightarrow$ *Ação(Pega,t)*

Problemas:

- Agente não tem memória (estado interno);
- Como saber se já está com o ouro? $\forall t$ *[NoOuro(t)* $\wedge \neg$ *com(Ouro, t)* $\Rightarrow$ *Ação (pega,t)]*

com(Ouro,t) não pode ser observado pelo agente!!!!! $\Rightarrow$ Necessário anotar mudanças no mundo.

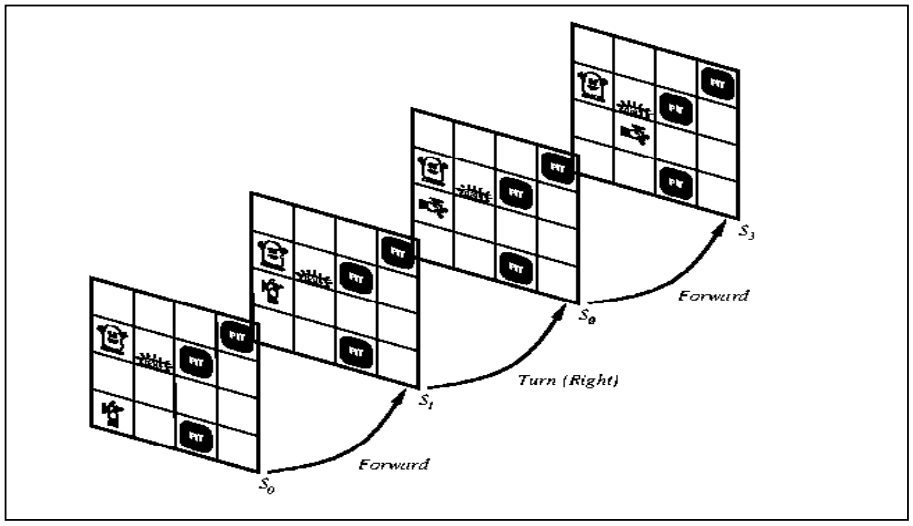
- Outro exemplo: agente não sabe em qual posição está, como saber quando sair?

Representando alterações no mundo...

Cálculo das situações (situacional):

O agente toma decisões baseado no que foi percebido no passado e reescrevendo as sentenças sobre o estado atual do mundo. Estas sentenças são alteradas a cada novo fato percebido no mundo e para cada ação realizada. A maneira mais fácil de representar trocas é apagar a sentença que diz *"agente em [1,1]"* e substituir pela a sentença *"agente em [1,2]"* $\Rightarrow$ mas nesse caso o passado é perdido.

$\Rightarrow$ Assim, um argumento em cada predicado representa a situação em que ele se aplica, a menos que ele se aplique a toda situação. Exemplo: Podemos escrever que um agente estava em [1,1] no instante S_0 e estava em [1,2] no instante S_1. Assim, ao invés de local(agente,[i,j]), diz-se local(Agente,[1,1],S_0) $\wedge$ *local(Agente,[1,2],*S_1*)*



Resultado(anda,S_0*)* = S_1

Resultado(vira(dir), S_1*)* = S_2

Resultado(anda, S_2*)* = S_3

Exemplo:

1 - Representar que o agente deve carregar um objeto portável sempre que ele estiver presente:

Portável(Gold)

$\forall s$ *NoOuro(s)* $\Rightarrow$ *com(Ouro,s)*

$\forall x,s$ *com(x,s)* $\wedge$ *Portável(x)* $\Rightarrow$ *com(x,Resultado(pega,s))*

2 - Não carregar nada após a ação de soltar um objeto:

$\forall x,s$ $\neg$*com(x,Resultado(larga,s))*

3 - Se um agente está carregando alguma coisa e ele não soltou então ele estará carregando no próximo estado:

$$\forall a,x,s\ com(x, s) \wedge (a \neq larga) \Rightarrow com(x,Resultado(a,s))$$

4 - Se um agente não estava carregando nada e ele não pegou ou não pode pegar nada então ele continuará não carregando nada:

$$\forall a,x,s\ \neg com(x, s) \wedge (a \neq pega \vee \neg(com(x,s) \wedge Portável(x))) \Rightarrow \neg com(x,Resultado(a,s))$$

Axiomas:

Axiomas de efeito: muda após ação:

Exemplos:

$$\forall s\ [NoOuro(s) \Rightarrow com(ouro,Resultado(pega,s))]$$

$$\forall s, x\ \neg com(x, Resultado(larga,s))$$

Frame axioms: não muda após ação

Exemplos:

$$\forall a, x, s\ [com(x, s) \wedge (a \neq larga) \Rightarrow com\ (x, Resultado(a, s))]$$

Problema: *como lidar com a "não mudança"* $\Rightarrow$ evitar re-inferir fatos que não mudam $\Rightarrow$ Axiomas de estado sucessor:

P verdadeiro após $\Leftrightarrow$ *ação faz P verdadeiro OU*

P já verdadeiro e

nenhuma ação o faz falso

Exemplo:

$$\forall a,x,s\ com(x,Resultado(a,s)) \Leftrightarrow [(a=pega \wedge com(x,s) \wedge Portável(x)) \vee (com(x,s) \wedge a \neq larga)]$$

Mantendo a localização:

Local Inicial: local(Agente, [1,1], S_0)

Direção: 0 no eixo x, 90 para cima,...

Orientação(Agente,S_0)=0

Local a frente dada uma direção do agente:

$\forall x,y\ LocalFrente([x,y],0)=[x+1,y]$

$\forall x,y\ LocalFrente([x,y],90)=[x,y+1]$

$\forall x,y\ LocalFrente([x,y],180)=[x-1,y]$

$\forall x,y\ LocalFrente([x,y],270)=[x,y-1]$

Adjacência:

$\forall l_1, l_2\ Adjacente(l_1, l_2) \Leftrightarrow \exists d\ l_1 = LocalFrente(l_2, d)$

Detalhes do mapa (ambiente 4X4):

$\forall x,y\ parede([x,y]) \Leftrightarrow (x=0 \vee x=5 \vee y=0 \vee y=5)$

O agente troca de posição somente indo pra frente, e quando não existir parede:

$\forall a,d,p,s\ Local(p,l,Resultado(a,s)) \Leftrightarrow [\ (a=anda \wedge l=LocalAFrente(p,s) \wedge \neg Parede(l)) \vee (Local(p,l,s) \wedge a \neq anda)]$

Deduzindo Propriedades "ocultas" (não Explícitas):

Regra de **Diagnóstico**: infere causas a partir do efeito

$\forall l\ ventando(l) \Rightarrow \exists l_1\ local(armadilha, l_1) \wedge Adjacente(l_1, l)$

Regra **Causal**: infere efeito a partir da causa

$\forall l_1, l_2\ local(armadilha, l_1) \wedge Adjacente(l_1, l_2) \Rightarrow ventando(l_2)$

Como deduzir que existe um Wumpus na vizinhança mas não se sabe a posição exata:

$\forall l_1, s\ Cheirou(l_1) \Rightarrow (\exists l_2\ Local(Wumpus, l_2, s) \wedge (l_2 = l_1 \vee Adjacente(l_1, l_2))$

Definição de "Ok":

$\forall x,t\ [Ok(x) \Leftrightarrow \neg Local(Wumpus,x,t) \wedge \neg Local(Armadilha,x,t)]$

Verificar que as posições vizinhas a uma posição sem vento e cheiro estão OK:

$\forall x,y,g,u,c,s\ Estimulo([-,-,g,u,c],t) \wedge Local(Agente,x,s) \wedge Adjacente(x,y) \Rightarrow OK(y)$

$\Rightarrow$ Sobre esta representação são utilizado mecanismos de inferências...

Preferências sobre ações a serem tomadas:

Ações:

- Ótimas incluem pegar o ouro e sair da caverna;
- Boas incluem mover para um quadrado que está OK que não foi visitado.
- Médias incluem mover para um quadrado que está OK que já foi visitado.
- De risco incluem mover para um quadrado que não sabemos se podemos morrer.
- Suicidas consiste em mover para um quadrado que sabemos que contém uma armadilha ou um wumpus.

$\Rightarrow$ Devemos fazer ações ótimas quando possíveis, boas quando as ótimas não forem possíveis e assim por diante:

$\forall a,s\ Otima(a,s) \Rightarrow Ação(a,s)$

$\forall a,s\ Boa(a,s) \wedge (\neg \exists b\ Otima(b,s)) \Rightarrow Ação(a,s)$

$\forall a,s\ Media(a,s) \wedge (\neg \exists b\ Otima(b,s) \vee Boa(b,s)) \Rightarrow Ação(a,s)$

$\forall a,s\ Media(a,s) \wedge (\neg \exists b\ Otima(b,s) \vee Boa(b,s) \vee Media(b,s)) \Rightarrow Ação(a,s)$

4.11 INFERÊNCIA COM LÓGICA DE PRIMEIRA ORDEM

Considere o seguinte fato:

Jack tem um cachorro

Todo dono de animais é um admirador de animais

Quem gosta de animal não mata animais

Jack ou a Curiosidade matou o gato, que se chamava Tuna

Deve-se provar se **"A curiosidade matou o gato?"**

Modelando em fórmulas lógicas tem-se:

$\exists x\ Dog(x) \wedge Owns(Jack,x)$

$\forall x\ (\exists y\ Dog(y) \wedge Owns(x,y)) \Rightarrow AnimalLover(x)$

$\forall x\ AnimalLover(x) \Rightarrow \forall y\ Animal(y) \Rightarrow \neg Kills(x,y)$

$Kills(Jack,Tuna) \vee Kills(Curisity,Tuna)$

$Cat(Tuna)$

$\forall x\ Cat(x) \Rightarrow Animal(x)$

Convertendo para a forma normal:

A1. Dog(D)

A2. Owns(Jack,D)

B. $Dog(y) \wedge Owns(x,y) \Rightarrow AnimalLover(x)$

C. $AnimalLover(x) \wedge Animal(y) \wedge Kills(x,y) \Rightarrow False$

D. $Kills(Jack,Tuna) \vee Kills(Curisity,Tuna)$

E. Cat(Tuna)

F. $Cat(x) \Rightarrow Animal(x)$

Deve-se provar por contradição, assim supor que *"a Curiosidade não matou tuna"*. Note que é feito um processo de substituição de objetos até que se chegue a um resultado falso, ou seja se *"A Curiosidade não matou Tuna"* é falso, então *"A Curiosidade matou Tuna"* é verdadeiro. Essa lógica implementada durante o processo de prova é mostrado na figura a seguir: