universidade de aveiro

deti departamento de electrónica, telecomunicações e informática

Inteligência Artificial

Teóricas

Resumos

Noções de programação declarativa

Programação funcional em Python

Tópicos de Inteligência Artificial

Representação do conhecimento

Resolução automática de problemas

Gonçalo Matos, 92972

Licenciatura em Engenharia Informática

3.º Ano | 1.º Semestre | Ano letivo 2020/2021

Última atualização a 28 de janeiro de 2021

Este documento é baseado nos *slides* teóricos do professor Luís Seabra Lopes, de onde a maioria das imagens são retiradas. Algumas são ainda capturas de ecrã das aulas assíncronas.

Sempre que for mencionada "bibliografia" como fonte, refere-se a consulta do livro Artifical Intelligence: a modern aproach, de Stuart Russel e Peter Norvig.

Fontes adicionais são referenciadas no início dos capítulos onde foram utilizadas.

Índice

1. Noções de programação declarativa

Slides teóricos
Fonte adicional: Stackoverflow sobre programação declarativa, consultado em 10/10/2020

Caracteriza-se por **programação imperativa** um fluxo de operações explicitamente sequenciado de operações com foco na forma como as tarefas são executadas (instruções), podendo alterar o conteúdo em memória e ainda realizar análise de casos (ifs), ciclos e ter associados sub-programas.

Em contraste, a **programação declarativa** abstrai-se da implementação focando-se apenas na descrição do que se pretende fazer enquanto faz uso de dois paradigmas: o **funcional**, baseado em funções (lambda) e o **lógico**, baseado em na lógica de primeira ordem sobre predicados.

SQL é uma linguagem declarativa, uma vez que nos seus comandos apenas descrevemos o que queremos obter. Assim, o programador é abstraído da forma como as operações são executadas na prática, ficando essa tarefa a cargo do compilador.

	Funcional	Lógico
Fundamentos	Lambda calculus	Lógica de primeira ordem
Conceito central	Função	Predicado
Mecanismos	Aplicação de funções Unificação uni-direccional Estruturas decisórias	Inferência lógica (resolução SLD) Unificação bi-direccional
Programa	Um conjunto de declarações de funções e estruturas de dados	Um conjunto de fórmulas lógicas (factos e regras)

A sua origem data da segunda metade do século XX, mas ainda hoje é amplamente usada e até está em crescimento em áreas como a Inteligência Artifical.

No entanto, a sua alta abstração torna por vezes a sua escrita complexa. Deve por isso haver um processo de análise mental antes de começar a programar uma solução.

Perceber o problema > Desenhá-lo > Escrevê-lo > Rever e testar

2. Programação funcional em Python

Slides teóricos

Criada no final dos anos 90, o **Python** é uma linguagem interpretada, interativa e funcional que tem como principais características a legibilidade, simplicidade (sem prejuízo da utilidade), modularidade e facilidade de integração, que permitem um rápido desenvolvimento. Apresenta-se como uma linguagem **multi-paradigma**.

Funcional Expressões lambda, funções de ordem superior, listas com sintaxe simples, iteradores…

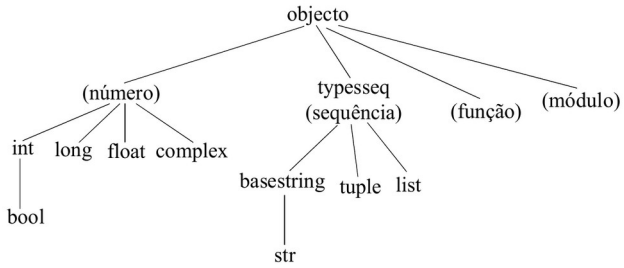
Orientada a objetos Classes, objetos, métodos, herança…

Imperativa Atribuição, sequências, condicionais, ciclos…

Comparativamente ao JAVA revela-se menos consisa porque os espaços são sintaticamente relevantes e o código, não compilado para código nativo (interpretado), demora mais tempo a executar (diferença residual), no entanto apresenta uma maior facilidade de desenvolvimento e legibilidade.

Objetos

Cada objeto é caracteriza-do por uma **referência**, um **tipo** e um **valor**.



Alguns destes são imutáveis, como as **str** e **tuple**.

Para determinar o tipo de um objeto utiliza-se a função pré-definida **type()**.

Variáveis

As variáveis guardam objetos, mas não são declaradas (inicializadas apenas) nem têm tipos associados (o tipo é do objeto!). Com exceção das funções quase tudo pode ser atribuído a uma variável.

Apesar de ser uma característica das linguagens imperativas e pouco comum nas funcionais, o valor das variáveis pode ser alterado.

Operações

As operações são maioritariamente realizadas com os operadores matemáticos habituais, salvo algumas exceções.

//	Quociente da divisão inteira
%	Resto da divisão inteira
**	Potência

Variáveis

```
# É possível fazer atribuições simultâneas

a = b = c = 10
```

Instruções de atribuição

A atribuição é uma referência para o objeto original, não uma cópia!

```
# Podemos decompor estruturas na atribuição

(a, b) = (10, 20)
```

Operadores lógicos

Na conjunção e disjunção, o segundo elemento só é avaliado de for necessário para determinar o resultado.

```
if False and a: # a não vai ser avaliado
```

Acesso a sequências

É possível extrair "fatias" das sequências. A indexação é circular, pelo que podemos aceder a índices negativos.

```
# [inf:sup] retorna a sequência entre os índices inf e sup-1 (uma cópia da lista)

# Para fazer uma cópia integral de uma lista usa-se [:]
```

Funções

Os parâmetros são passados por referência, não cópias!

```
# Podem ter parâmetros por defeito, que podem ser omitidos na chamada

def abc(arg1, arg2=True, arg3=None):

    pass

# Quando só queremos passar um dos args por defeito devemos indicar o seu nome
```

```
abc(2, arg3=4)
```

Expressões lambda

São expressões cujo valor é uma função. Funções que recebem expressões lambda como entrada e as retornam como saída chamam-se **funções de ordem superior**.

```
lambda x:x+1

# Podem ser atribuídas a variáveis

m = lambda x:x+1

m(1) # Output: 2

# Podem por isso ser passadas como argumentos de funções



# Podem ainda ser produzidas por outras funções

def incrementar(n):

   return lambda x:x+n

suc = incrementar(“a”)

print(suc(“bc”)) # Output: bca

# Outro exemplo

# Dadas 3 funções de 2 entradas retorna função de 3 entradas na forma h(f(x,y), g(y,z))

f = lambda f,g,h: lambda x,y,z: h(f(x,y), g(y,z))
```

Reduzir uma lista a um valor

Muitos dos procedimentos aplicados a listas consistem em combinar a cabeça da lista com o resultado da chamada recursiva sobre os restantes elementos, retornando um valor “neutro” pré-definido para a lista vazia.

```
# Dada uma lista, uma função de combinação e um elemento neutro

def reduzir(lista, fcomb, neutro):

  if lista == []:

    return neutro

  return fcomb(lista[0], reduzir(lista[1:], fcomb, neutro)
```

Listas de compreensão

São mecanismos compactos para processar os elementos de uma lista, podendo ser aplicadas não só a estas como também a tuplos ou cadeias de caracteres (str). Podem funcionar com a função pré-definida **map()** ou **filter()**.

```
# <expr> for <var> in <seq> if <condition>

[x**2 for x in [2,3,4]]       # Output: [4,9,16]

map(lambda x:x**2, [2,3,4])       # Same

[x for x in [1,2,3] if x%2==0]# Output: 2

filter(lambda x:x%2==0, [1,2,3])   # Same

# Podem percorrer várias sequências

# Dada uma lista de listas e uma função, aplica a função a cada um dos elementos das listas, retornando a concatenação das listas resultantes.

[f(y) for x in lista for y in x]
```

Classes

Tal como nas restantes linguagens Orientadas a Objetos (OO), em Python uma classe define um conjunto de objetos caracterizados por diversos atributos e métodos e organizam-se numa hierarquia que permite a herança.

Para mais info ver https://github.com/gmatosferreira/CheatSheets/blob/master/PythonObjectOriented.md

Tal como nas variáveis normais, os atributos das classes não são declarados!

Listas

Esta classe implementa a noção de lista em Python de uma forma bastante simplifica, oferencendo um conjunto abrangente de operações que as permitem manipular.

list.append(x) – acrescenta **x** ao fim da lista
list.extend(L) – acrescenta elementos da lista **L** no fim da lista
list.insert(i,x) – insere **x** na posição **i**
list.remove(x) – remove a primeira ocorrência de **x**
list.index(x) – remove a posição da primeira ocorrência de **x**
list.sort() – ordena a lista (modifica a lista)

3. Tópicos de Inteligência Artificial

Slides teóricos e aulas assíncronas
Fonte adicional: UC Berkley sobre vertentes da AI, consultado em 25/10/2020

Segundo a sua definição lexical, **inteligência** é a capacidade de pensar, raciocionar, adquirir e aplicar conhecimento.

O seu estudo envolve várias vertendes, desde a já referida aquisição, representação e armazenamento de conhecimento, a geração de comportamento inteligente, a origem das motivações, emoções e prioridades, como as perceções dão origem a entidades simbólicas, como raciocionar sobre o passado, planear o futuro, como surge a ilusão, a crença, esperança, amor…

Historicamente, estas vertententes resumem-se a quatro grandes abordagens, definidas por Russel & Norvig (1995).

Pensar como o ser humano	Pensar racionalmente
Agir como o ser humano	Agir racionalmente

As **humanas** focam-se no ser humano, desenvolvendo uma ciência empírica[1], baseada na hipótese e na experiência. Por outro lado, a abordagem **racional** envolve uma combinação de matemática e engenharia.

No início estava muito focada na imitação das capacidades do ser humano, no entanto, mais recentemente tem-se voltado mais para a racionalidade como bitola (referência).

Atualmente é maioritariamente utilizada para a resolução de problemas concretos e pouco abrangentes, foncando-se na melhor utilização dos recursos de forma racional.

Mas voltando à sua história, as bases da inteligência artifical remontam ao século IV a.C.. Foi nesta altura que nasceu a **lógica** e os fundamentos do pensamento racional. Já mais perto do século XX surgiu o movimento do **empirismo**, que defende que nada é compreendido sem antes ter estado nos sentidos.

A psicologia também contribuiu com o **comportamentalismo**, que estuda o comportamento com base na observação de estímulos e reações, seguida pela sua oposição, o **cognitivismo**, que defende que o comportamento é influenciado pela forma de pensar e não se resume a uma resposta a estímulos.

Em 1940, com o surgimento dos computadores esta área foi-se desenvolvendo, tendo sido oficalmente batizada em 1956 e desde então evoluido até à forma como hoje é conhecida.

1 Que é baseado na experiência.

Agir como o ser humano

Fonte adicional: Univ. Stanford sobre a sala chinesa, consultado em 25/10/2020

O **teste de Turing** (1950) pretendeu estebelecer a definição operacional para o comportamento inteligente de nível humano, que consistia em submeter o artefacto a um interrogatório através de um terminal de texto. Se o humano não conseguir concluir que está a interrogar uma máquina, então conclui-se que o artefacto é inteligente.

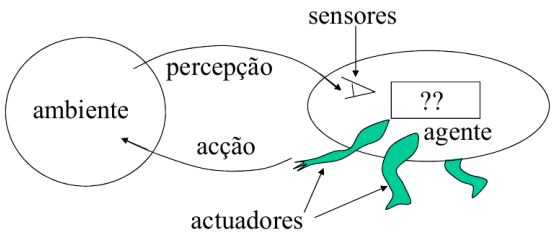
Esta teoria foi refutada por **Searle**, um filósofo americano, com o argumento da **Sala Chinesa**. Neste, descreveu o cenário de colocar um humano que não percebe chinês fechado dentro de uma sala com um livro escrito na sua língua que ensine chinês a ser interrogado por outro humano através de papéis passados por baixo da porta.

Searle defendeu que apesar de não perceber os papéis entregues por baixo da porta, o humano iria conseguir traduzi-los através do livro e elaborar respostas. Fora da sala, o interrogador iria ter a ilusão de que o interrogado sabia chinês. No entanto, nem o humano, nem a sala, nem o livro percebem chinês. Logo, deduziu que o sistema não compreende chinês.

Foi esta a premissa para a sua argumentação de que os computadores apenas fazem uso de regras sintáticas para manipular o texto, sem ter qualquer entendimento do seu significado ou semântica. Para ele a inteligência é um processo biológico, que apenas pode ser simulado pelas máquinas, mas nunca adquirido.

Agentes

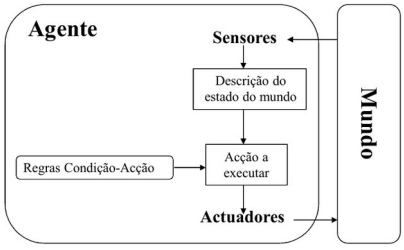
Definidos como uma **entidade** com capacidade de obter informação sobre o seu ambiente (sensores) e de executar (atuadores) ações em função dessa informação por Russel e Norvig.

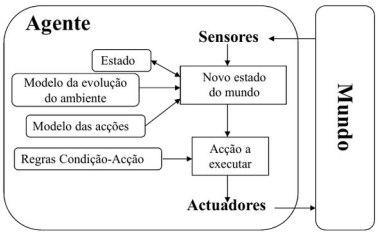


Agentes reativos

Também conhecidos por agentes de **estímulo-resposta**, este tipo de agente apresenta um conjunto de sensores, através dos quais recebe uma perceção do estado do mundo, sobre a qual aplica um conjunto de regras (**regras de condição-ação**) e executa as ações correspondentes. Um agente com estass características diz-se **agente reativo simples**.

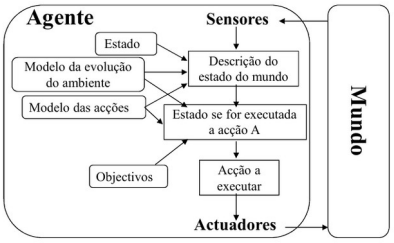
Existem ainda **agentes reativos com estado interno**, que funcionam de forma semelhante ao anterior, mas para além dos sensores, fazem uso de um estado interno e do histórico de ações anteriores para construir a perceção do estado do mundo.

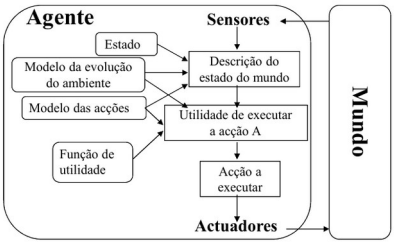




Agentes deliberativos

Contrariamente aos anteriores, executam as ações com base em **objetivos** ou **função de utilidade**.





A **função de atividade** é uma função que avalia a relevância de cada estado para o agente.

Características do mundo de um agente

Fonte adicional: JavaTPoint sobre ambientes do agente, consultado em 25/10/2020

Visto como se define um agente e os seus tipos, falta perceber como se caracteriza o seu ambiente. Este pode apresentar várias propriedades.

Acessível (vs. não acessível)

O agente tem acesso a toda a informação sobre o ambiente, ou pelo menos a toda a informação relevante para o processo de decisão.

Deterministico (vs. estocástico)

O resultado de uma ação é apenas e totalmente determinado pelo estado atual e pelos efeitos esperados da ação.

Episódico (vs. não episódico)

Cada par perceção-ação é indepente dos outros.

Dinâmico (vs. estático)

O ambiente pode sofrer mutações sem intervenção do agente enquanto este delibera.

Continuo (vs. discreto)

O ambiente apresenta um número infinito de perceções e ações que podem ser feitas (evolui de forma contínua).

Um agente táxi opera num ambiente dinâmico, enquanto que um agente que jogue palavras-cruzadas opera num ambiente estático.

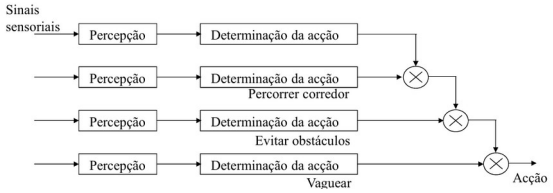
Para um agente carro de condução autónoma, a estrada é um ambiente contínuo. Já para um agente que jogue xadrez, o tabuleiro é um ambiente discreto.

Arquiteturas de agentes

Subsunção

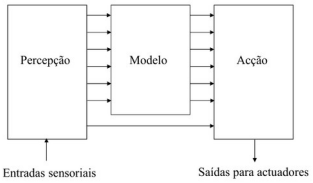
Esta arquitetura procura estabelecer a ligação entre a perceção e a ação em vários níveis, organizados em **camadas**, criando agentes com comportamento simultâneamente reativo e deliberativo.

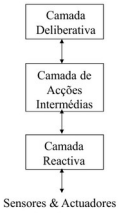
A camada mais baixa é a mais reativa, diminuindo a reatividade e aumentando o peso da deliberação à medida que se sobe nas camadas. A ação do agente resulta da fusão das decisões das várias camadas.



Três torres

O processo de tomada de decisão é simbólico e feito a partir de um modelo, mas tem algumas ações reativas.





Três camadas

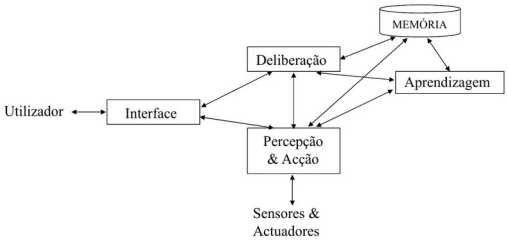
Para além das camadas deliberativas e reativas tem uma camada de ações intermédias, que é o intermediário entre as duas, produzindo uma alteração qualitativa no estado do mundo.

CARL

Esta arquitetura (anos 2000) veio introduzir dois novos conceitos.

Interfaces Facilitam a interação por pessoas não especializadas.

Aprendizagem Para funcionar num ambiente humano não estruturado vai ter de aprender para se ir adapatando às condições do mundo real. Está assente em **memória**.


MEMÓRIA
Deliberação
Aprendizagem
Utilizador
Interface
Percepção
& Acção
Sensores &
Actuadores

4. Representação do conhecimento

Slides teóricos e aulas assíncronas

Para poderem tomar decisões, os agentes (especialmente os deliberativos) precisam de ter algum conhecimento sobre o mundo em que se inserem.

Redes semânticas

As **redes semânticas** são representações gráficas do conhecimento que facilitam a legibilidade.

Relações definidas

Tal como na lógica de primeira ordem, nas redes semânticas o conhecimento representa-se através de relações entre entidades (de uma forma mais simples e visual).

Sub-tipo, sub-conjunto ou sub-classe $A \subset B$

Membro ou instância $A \in B$

Relação objeto-objeto $R(A, B)$

Relação conjunto-objeto $\forall x, x \in A \Rightarrow R(x, B)$

Relação conjunto-conjunto $\forall x, x \in A \Rightarrow \exists y (y \in B \wedge R(x, B))$

No entanto, apesar de também suportarem operações de **negação**, **disjunção** e **quantificação**, geralmente estas não são contemplatadas nas redes semânticas por razões computacionais.

Estas relações podem ainda ser repsentadas visualmente.

Herança

A herança pode ser representada em relações de **sub-tipo**, herdando todas as propriedades dos tipos mais abstratos dos quais descende, ou de **instância**, herdando as todas as propriedades do tipo a que pertence.

Aplica-se um **raciocínio não monotónico**[2] devido ao estabelecimento dos valores por defeito, propriedades comuns a todos os membros desse tipo e do cancelamento de herança, que ocorre quando um sub-tipo altera um valor por defeito herdado de um tipo do qual descende.

2 Que tem sempre o mesmo tom ou cujo tom não varia.

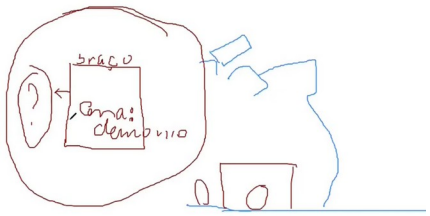
Sistemas de frames

Fonte adicional: Universidade Federal de Pernambuco, consultados em 10/11/2020

Um sistema de frames[3] é um conjunto de frames organizados hierarquicamente, onde cada um descreve um objeto complexo através de um conjunto de atributos, que descrevem conhecimento **declarativo** sobre objetos e eventos e **procedimental** sobre como recuprar a informação ou calcular valores.

Alguns destes, designados por **demónios**, são procedecimentos cuja execução é disparada automaticamente quando certas operações de leitura ou escrita são efetuadas.

Por exemplo um braço robótico que adquire conhecimento do mundo através de um sensor visual (uma câmara de vídeo) pode ter um sistema de *frames* que tem um atributo *cena* que descreve o mundo.



Na representação interna deste robot, quando é necessário tomar uma decisão que vai ter por base a representação semântica do conhecimento do braço é feito um pedido de consulta do valor da cena. Quando este pedido é feito, é desencadeada uma chamada ao **procedimento demónio** que vai capturar a cena, processá-la e atualizar os valores, finalmente retornados.

Relação com UML

Facilmente se identificam pontos na UML que fazem lembrar redes semânticas.

Classe Conjunto de objetos que partilham os mesmos atributos, operações, relações e semântica.

Atributo Propriedade de uma classe

Operação/método Implementação de um serviço qye pode ser executado por qualquer objeto da classe.

Nestes diagramas podem ainda ser criadas **instâncias** de classes, que são concretizações que se diferenciam pelo valor dos atributos, herdando no entanto os atributos e métodos da sua classe.

Redes semânticas	*UML*
subtipo(SubTipo,Tipo)	Generalização em diagramas de classes
membro(Obj,Tipo)	Diagramas de objectos
Relação Objecto/Objecto	Associação, agregação e composição em diagramas de objectos
Relação Objecto/Tipo	não tem
Relação Tipo/Tipo	Associação, agregação e composição em diagramas de classes

3 Visualmente uma entidade é representado por um quadrado, em inglês *frame*.

Indução vs. Dedução

A **dedução** permite a partir de um conjunto de permissas consideradas verdadeiras, por mecanismos de inferências lógica, tirar conlusões verdadeiras.

No entanto, para adquirir conhecimento novo é necessário haver **indução**, ou seja, a capacidade de inferir regras gerais a partir de observações concretas.

Nas redes semânticas a inferência pode ser vista como uma herança de baixo para cima.

Pode ser feita uma inferência de que "os gatos (normalmente) gostam de leite" depois de observar os gatos Tareco e Pirata a gostarem de leite.

A partir das instâncias de Gato **inferimos** que a classe Gato gosta de leite.

Lógica proposicional e de primeira ordem

Apontamentos para as aulas de IA, Luís Botelho, ISCTE-UL, sobre lógica de primeira ordem

A lógica é uma linguagem, e como qualquer outra linguagem, se decompõee em…

Sintaxe Descreve o conjunto de frases ou fórmulas que é possível escrever (fórmulas bem formadas).

Semântica Estabelece a relação entre as frases escritas e os factos que representam.

Regras de inferência Permitem manipular frases, gerando umas a partir das outras. São a base do processo de raciocínio.

A **lógica proposicional** é baseada em proposições, frases declarativas que podem ser verdadeiras ou falsas, que podem ser representadas por variáveis proposicionais que tomam o seu valor de verdade. As fórmulas são compostas por uma ou mais variáveis proposicionais ligadas por conectivas lógicas.

As proposições não dependem da interpretação, uma vez que, contrariamente aos predicados, as proposições não têm argumentos.

"A neve é branca." é uma proposição, que é verdadeira.

Em contraste temos a **lógica de primeira ordem**, onde há uma distinção entre os objetos e as relações (predicados).

Além de variáveis, usadas para representar termos não especificados e constantes escalares, os objetos podem ainda tomar a forma de expressões funcionais, que se formam pela aplicação de uma função aos seus argumentos. Os objetos podem ser considerados como expressões funcionais cuja aridade[4] é zero.

O nome da relação aplica-se sempre ao seu primeiro elemento (argumento). Os seus argumentos só podem ser termos (nunca relação de relação).

A noção "termo" engloba quer objetos, quer expressões funcionais.

4, 3, Rui e Paulo são **objetos**.

4 Número de argumentos

Potencia(4,3) é uma **expressão funcional**.
Pai(Rui, Paulo) é um **predicado**.

Conectivas lógicas (LP / LPO)

Comuns às duas lógicas abordadas, servem para combinar frases lógicas elementares por forma a obter frases mais complexas.

∧ Conjunção ∨ Disjunção ⇒ Implicação ¬ Negação

Quantificadores (LPO)

Para quantificar as variáveis da lógica de primeira ordem temos os **quantificadores universais** e os **quantificadores existenciais**.

∀ x A ∃ x A

Se A é uma fórmula bem formada (FBF), então a sua quantificação também é uma FBF.

Gramática (LPO)

Fórmula → *FórmulaAtómica*
| *Fórmula Conectiva Formula*
| *Quantificador Variável*, ... *Fórmula*
| '¬' *Fórmula*
| '(' *Fórmula* ')'
FórmulaAtómica → *Predicado* '(' *Termo* ',' ...)' | *Termo* '=' *Termo*
Termo → *Função* '(' *Termo* ',' ...)' | *Constante* | *Variável*
Conectiva → '⇒' | '∧' | '∨' | '⇔'
Quantificador → '∃' | '∀'
Constante → A | X1 | Paula | ...
Variável → a | x | s | ...
Predicado → Portista | Cor | ...
Função → Registo | Mãe | ...

Por convenção, todos os elementos começam com letra maiúscula, menos as variáveis.

Interpretações de fórmulas

Na **lógica proposicional**, a interpretação de uma fórmula é a atribuição de valores de verdade ou falsidade às várias proposições que nela ocorrem.

A fórmula $A \wedge B$ tem quatro interpretações possíveis que resultam da combinação dos valores de verdade e falsidade de A e B.

Já na **lógica de primeira ordem**, a interpretação de uma fórmula é o estabelecimento de uma correspondência entre as constantes que ocorrem na fórmula e os objetos do mundo, funções e relações que essas contantes representam.

Uma interpretação **satisfaz** uma fórmula sse a fórmula toma o valor verdadeiro para essa interpretação, sendo neste caso designada por **modelo da fórmula**.

Assim, uma fórmula diz-se **satisfatível** se existe uma interpretação que a satisfaz.

Uma fórmula cujo valor é verdadeiro para qualquer interpretação, diz-se uma **tautologia**.

Regras de substituição

Estas regras permitem substituir uma fórmula por outra equivalente.

Leis de DeMorgan
$\neg (A \wedge B) \equiv \neg A \vee \neg B$
$\neg (A \vee B) \equiv \neg A \wedge \neg B$

Dupla negação:
$\neg\neg A \equiv A$

Definição da implicação:
$A \Rightarrow B \equiv \neg A \vee B$

Transposição:
$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$

Comutação
$A \wedge B \equiv B \wedge A$
$A \vee B \equiv B \vee A$

Associação:
$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
$(A \vee B) \vee C \equiv A \vee (B \vee C)$

Distribuição:
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Há ainda as **leis DeMorgan**, que apenas se aplicam à lógica de primeira ordem.

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

Demonstração automática de teoremas

Nem sempre as regras de inferência mais fáceis de utilizar pelos humanos são as mais indicadas para uma máquina aplicar. É assim fundamental estandardizar as representações colocando-las na **forma clausal**.

Literal Fórmula atómica (predicato positivo), ou a sua negação

Cláusula Disjunção de literais

Forma Normal Conjuntiva (CNF) Conjunção de cláusulas

Forma clausal Representação da CNF através de um conjunto de cláusulas

Um **literal** é p ou ~p,
Uma **cláusulas** é l1 v l2 v … v ln
Uma **CNF** é c1 & c2 & … & cn
Uma **forma clausal** é { c1, c2, …, cn }

Conversão para a CNF

A **conversão** das fórmulas proposicionais e de primeira ordem **para a CNF** faz-se com recurso às regras de substituição.

Lógica proposicional	Lógica de primeira ordem
• Remover implicações • Reduzir âmbito da aplicação das negações • Associar e distribuir até chegar à CNF	• Renomear variáveis de forma a que cada quantificador tenha uma variável diferente $\forall x \forall y (p(x,y) \Rightarrow \forall y q(y,y))$ $\forall a \forall b (p(a,b) \Rightarrow \forall c q(c,c))$ • Remover implicações • Reduzir âmbito da aplicação das negações • Skolemização ◦ Substituir todas as ocorrências de cada variável quantificada existencialmente por uma função cujos argumentos são as variáveis dos quantificadores universais exteriores $\forall a \forall b (p(a,b) \wedge \exists c \neg q(c,c))$ $\forall a \forall b (p(a,b) \wedge \neg q(f(a,b), f(a,b)))$ • Remover quantificadores universais $p(a,b) \wedge \neg q(f(a,b), f(a,b))$

	• Associar e distribuir até chegar à CNF • Renomear variáveis de forma a que uma variável não apareça em mais do que uma fórmula $p(a_1, b_1), \neg q(f(a_2, b_2), f(a_2, b_2))$

Regras de inferência

Modus Ponens: $\{ A, A \Rightarrow B \} \vdash B$

Modus Tolens: $\{\neg B, A \Rightarrow B \} \vdash \neg A$

Silogismo hipotético: $\{A \Rightarrow B, B \Rightarrow C \} \vdash A \Rightarrow C$

Conjunção: $\{ A, B \} \vdash A \wedge B$

Eliminação da conjunção: $\{A \wedge B \} \vdash A$

Disjunção: $\{ A, B \} \vdash A \vee B$

Silogismo disjuntivo (ou resolução unitária):
$\{ A \vee B, \neg B \} \vdash A$

Resolução: $\{ A \vee B, \neg B \vee C \} \vdash A \vee C$

Dilema construtivo:
$\{ (A \Rightarrow B) \wedge (C \Rightarrow D), A \vee C \} \vdash B \vee D$

Dilema destrutivo:
$\{ (A \Rightarrow B) \wedge (C \Rightarrow D), \neg B \vee \neg D \} \vdash \neg A \vee \neg C$

Instanciação universal:
$\{\forall x\ P(x) \} \vdash P(A)$

Generalização existencial
$\{P(A) \} \vdash \exists x\ P(x)$

Consequências lógicas e provas

A regras de inferência permitem-nos obter **consequências lógicas** (A) a partir de um conjunto de fórmulas ($\Delta = \{ A_1, \ldots, A_n \}$), representada esta noção por **|=**.

$$\Delta \models A$$

Esta consequência é obtida se A tomar o valor verdadeiro em todas as interpretações para as quais cada uma das fórmulas em **Δ** toma também o valor verdadeiro.

A **prova** de uma fórmula A_n é dada pela sequência de fórmulas $\Delta = \{ A_1, \ldots, A_n \}$ tal que cada A_i pode ser inferida a partir das fórmulas anteriores $A_1, \ldots, A_{i-1}$. Neste caso, escreve-se:

$$\Delta \models A_n$$

Correção e completude

Estes dois conceitos estão interligados, uma vez que em vez de considerarmos as interpretações que satisfazem **Δ** para validar que A é satisfeito em todos estes casos, podemos provar A a partir de **Δ**.

Tirar consequências lógicas a partir da enumeração das várias interpretações é difícil em lógica proposicional e ainda mais em lógica de primeira ordem, em que os objetos são figuras do mundo, que dependendo do seu domínio pode ter dimensões gigantes.

Um **sistema de inferência** então diz-se **correcto** e **completo** se permite tirar consequências lógicas sem ter de analisar caso a caso as várias interpretações (por força bruta).

Correção Se todas as fórmulas que gera são consequências lógicas.

Completude Se permitir gerar todas as consequências lógicas.

As regras de inferência descritas acima são corretas, porque todas as fórmulas que gera são consequências lógicas, mas não são completas, uma vez que não permitem gerar todas as consequências lógicas.

Metateoremas

Das consequências lógicas podem ainda retirar-se dois **metateoremas**.

$$\text{Teorema da dedução}: Se \{A_1, ... A_n\} \models B, então\ A_1 \wedge ... \wedge A_n \Rightarrow B$$

$$\text{Redução ao absurdo}: Se\ \Delta\ é\ satisfazível, e\ \Delta \cup \neg A\ não\ é\ satisfazível, então\ \Delta \models A$$

Se Δ |= A, o que pelo teorema da dedução nos permite deduzir que Δ implica A e A não for verdade, então Δ nunca vai poder ser verdade, pelo modus tolens.

A **resolução** é uma regra de inferência correta, uma vez que gera uma consequência lógica, mas não é completa, uma vez que não é possível dela derivar todas as consequências lógicas, como $A \wedge B \models A \vee B$.

No entanto, a **refutação por resolução** é um mecanismo de inferência completo, uma vez que através do teorema da redução ao absurdo se prova que a negação da consequência lógica é inconsistente com a premissa. Para tal, basta derivar Falso.

No caso anterior, de $A \wedge B \wedge \neg(A \vee B)$ deriva-se Falso.

Este mecanismo é universal e muito mais eficiente do que fazer a enumeração de todas as interpretações.

Substituições e unificação

Em ambas as lógicas pode ser necessário substituir as variáveis de forma a fazer alguma inferência sobre determinada fórmula.

A **substituição** representa-se por $s = \{t_1/x_1, ..., t_n/x_n\}$ e aplicada à fórmula W denota-se **Ws** ou **SUBST(W,s)**.

Dadas duas fórmulas A e B tais que As=Bs, s diz-se **substituição unificadora** e As e Bs **unificadas**. A substituição unificadora **mais geral** é a mais simples (menos extensa) que permita a unificação.

Por exemplo para inferir a resolução de $\{P(A,x) \vee Q(x,B), \neg Q(C,y) \vee R(z,D)\}$ tem de se aplicaar a **substituição** $s = \{x/C, y/B\}$, de forma a **unificar** Q(x,B) e Q(C,y). A inferência passa a ser aplicável apenas a este caso particular da substituição.

Foi aplicada a **substituição unificadora mais geral**, uma vez que foram alteradas apenas as variáveis estritamente necessárias à unificação. Podia ter-se substituído z, mas não se fez, porque não era necessário.

Conclui-se então que na lógica de primeira ordem que $\{A \vee B, \neg C \vee D\} \models SUBST(A \vee D, g)$, em que B e C são unificáveis com g a sua substituição unificadora mais geral.

Esta resolução é correta, não completa, mas, tal como na lógica proposicional, a **refutação por resolução** é completa.

Refutação por resolução

Este é um mecanismo de inferência completo, que através do teorema da redução ao absurdo se prova que a negação da consequência lógica é inconsistente com a premissa. Para tal, basta derivar Falso.

Resolução com cláusulas de Horn

A **refutação por resolução** é correta e completa, mas apesar de melhor que avaliar todas as interpretações da inferência, continua a ser pouco eficiente.

Para várias cláusulas com muitos literais em cada uma vão haver múltiplos casos em que a resolução pode ser aplicada entre um literal e outros de outras formulas, o que vai gerar uma explosão de casos e um problema tendencialmente mais difícil de resolver com o aumento do seu tamanho.

Uma delas são as **cláusulas de Horn**, uma cláusula que tem no máximo um literal positivo.

$$\neg A \vee B \vee \neg C \quad (=) \quad A \wedge C \Rightarrow B$$

O problema da combinatória da **refutação por resolução** vai ser minimizado, uma vez nas **cláusulas de Horn** cada cláusula vai ter menos possibilidades de resolução. O seu literal positivo pode ser resolvido com um negativo noutra fórmula e o inverso para os seus literais negativos.

Estas cláusulas permitem a aplicação de algoritmos com complexidade temporal linear.

Engenharia do conhecimento

A **base de conhecimento** é um conjunto de declarações que existem sobre aspetos de um certo domínio, que podem ser factos e regras do funcionamento, designadas genericamente por frases.

Com base na automizatização e simplificação do conhecimento estudadas anteriormente, as **regras** serão cláusulas de Horn e os **factos** cláusulas sem literais negativos.

"O Sócrates é homem." é um facto.

"Todos os homens são mortais." é uma regra.
Cláusula de Horn porque Homem => Mortal, que pode escrever-se como ~Homem v Mortal.

A **engenharia do conhecimento**, por sua vez, é o processo de preparar uma representação para uma determinada aplicação, construindo para este fim bases de conhecimento. Não é mais do que a modelação de um sistema, que resulta numa base de dados que para além de factos tem também regras. Consiste nos seguintes passos:

1. Estudar o domínio de aplicação (entrevistas com peritos par adquirir conhecimento);
2. Identificar os objetos, conceitos e relações que será necessário representar;

Sunstantivos **comuns** representam **conceitos** (classes ou tipos).
Substantivos **próprios** designam **objetos** (instâncias).
Verbo **ser** identifica **relações de instânciação** (objeto ou tipo) e **generalização** (subtipo e tipo).
Verbo **ter** e **conter** indica **composição**.

Não devem ser considerados substantivos irrelevantes. Quando dois ou mais substantivos distintos se referem ao mesmo conceito, deve ser escolhido o mais representativo ou mais adequado, procurando sempre uniformizar e agregar ao máximo com conceitos mais abstatos.

3. Definir **ontologia**[5]: escolher vocabulário para as entidades, funções e relações;

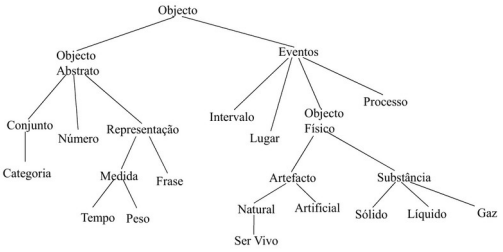
A **ontologia** é o vocabulário sobre um domínio conjugado com as suas relações e tem como objetivo captar a essência da organização do conhecimento num domínio.

4. Codificar conhecimento geral sobre o domínio, na forma de **axiomas** (regras);
5. Codificar descrições para problemas concretos, interrogar o sistema e obter respostas;
6. Caso o domínio seja tão complexo que não é praticável codificar à mão todo o conhecimento, pode usar-se aprendizagem automática.

Ontologia geral

Num projeto complexo com muitos aspetos do mundo real, a variedade de noções vai ser maior e vai envolver mais noções, que se organizam hierarquicamente.

Categorias, tipos ou classes; **medidas numéricas**; **objetos compostos**; **tempo**, **espaço** e mudanças; **eventos** e processos (eventos contínuos); **objetos físicos**; **substâncias**; **objetos abstratos** e crenças



No diagrama vemos a ontologia geral de uma classe Objecto.

5 Em informática é um conjunto estruturado de termos e conceitos que representam conhecimento sobre o mundo.

Troca de conhecimento entre agentes

De forma a permitir a troca de conhecimento entre agentes foi criada a linguagem **KIF**, **Knowledge Interchange Format**, que se destaca por ter uma semântica puramente **declarativa**[6] e permitir a representação de **meta-conhecimento**.

Tendo por base a lógica, representa o mundo através de objetos e relações entre eles, podendo os primeiros ser concretos ou abstratos e primitivos ou compostos.

A existência dos objetos pode ser conhecida, presumida ou suposta.
Uma relação é uma lista arbitrária de objetos.

Componentes da linguagem: Expressões

Termos Objetos primitivos ou compostos

Constantes, variáveis individuais, expressões funcionais, termos lógicos, caracteres e citações

Uma **expressão funcional** pode ser uma função ou uma estrutura de dados, ambas representadas por *functor arg1..n*.
As listas são uma expressão funcional, com functor listof: *listof arg1 … argn*.

As citações podem ser feitas com *(quote TERMO)* ou *'TERMO'*.

Frases Expressões com valor lógico

Constantes true ou false, (in)equações (comparação), frases relacional (predicados), lógica e quantificada

Frases relacionais: *(relação t1 … tn)*
Frases lógicas: *(not/and/or/</>/… t1 … tn)*
Frases quantificadas: *(forall/exists var1 … varn frase)*

Definições Frases verdadeiras por definição

Objetos, funções e relações (predicados)

Meta-conhecimento

Importante para o exame!

O **meta-conhecimento** formaliza conhecimento sobre o conhecimento. Para isto, recorre-se ao mecanismo da **citação** que permite tratar expressões como objetos, fazendo referência à sua representação em vez do seu significado.

Por exemplo *acredita joão 'material lua queijo'* representa conhecimento sobre a crença do João em que a lua é feita de queijo.

Conformação

O facto de ser uma linguagem **altamente expressiva** leva a que o KIF tenda a sobrecarregar os sistemas de geração de inferência, pelo que foi necessário definir várias **dimensões de conformação**, havendo para cada uma um **perfil de conformação**, que consiste numa seleção de níveis de conformação.

6 Foca-se mais no "que" do que no "como" é feito.

A conformação consiste na remoção da linguagem dos ingredientes que não sejam necessários.

- Lógica - atómica, conjuntiva, positiva, lógica, baseada em regras (de Horn ou não, recursivas ou não)
- Complexidade dos termos - termos simples (constantes e variáveis), termos complexos
- Ordem - *proposicional*, *primeira ordem* (contem variáveis, mas os functorese as relações são constantes), *ordem superior* (os functores e relações podem ser variáveis)
- Quantificação - conforme se usa ou não
- Meta-conhecimento - conforme se usa ou não

A conformação é apresentada da esquerda para a direita com uma escala decrescente de restrições (as da esquerda são as mais restritivas e por isso tornam a linguagem mais simples).

O nível de restrições adotadas em cada dimensão pode variar.

Conclui-se que esta é uma linguagem utópica[7], que tem um elevado grau de liberdade extremamente ambicioso, que na prática não funciona muito bem.

Apesar de ter como objetivo a troca de conhecimento entre agentes, não representa o tipo de mensagem, pelo que uma pergunta e uma afirmação correspondem ao mesmo conteúdo semântico. O KIF limita-se à representação do conteúdo.

Este suporte é dado pelo protocolo de comunicação KQML (Knowledge Query and Manipulation Language), que permitia a especificação do tipo de mensagem.

7 Sistema ou plano que parece irrealizável.

Redes de Bayes

Enquanto que na lógica temos elementos com valor de verdade absoluto (verdadeiro ou falso), nas **redes de crença bayesianas**, há a possibilidade de representar conhecimento impreciso em termos de um conjunto de variáveis aleatórias e respetivas dependências.

Com imprecisão deve-se à introdução da probabilidade do valor de verdade de cada proposição.
As dependências são expressas através de probabilidade condicionada.
A rede é um grafo acíclico.

As proposições no contexto probabilístico são variáveis aleatórias.

Para trabalhar com estas redes é necessário algum conhecimento de base sobre probabilidades. Destacam-se as fórmulas mais importantes.

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b) \qquad P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

P(a|b) é a probabilidade de 'a' dado/sabendo que 'b', ou seja, a probabilidade de 'a' ser verdade, sabendo que 'b' o é.

Para n variáveis vão haver 2^n possíveis atribuições de valores de verdade às variáveis. A probabilidade de ocorrer uma dada combinação de valores de todas as variáveis da rede é dada pela **probabilidade conjunta**.

$$P(x_1 \wedge x_2 \wedge \ldots \wedge x_n) = \prod_{i=1}^{n} P(x_i | P_{pais}(x_i))$$

A **probabilidade individual** é a probabillidade do valor específico (V ou F) de uma variável. Obtem-se somando as probabilidades conjuntas das situações em que essa variável tem esse valor específico, que vão ser 2^n para n variáveis das quais depende na rede bayesiana ascendente.

$$P(x_i = v_i) = \sum_{a_j \in (v,f)\, j=1,\ldots,k} P(xi \wedge a_1 \wedge a_k)$$

Seja x_i uma qualquer variável da rede, v_i o valor de x_i que se pretende calcular e a_k o conjunto das vari+aveis da rede que são ascendentes de x_i.

5. Resolução automática de problemas

Slides teóricos e aulas assíncronas

Um **problema** define-se como um **objetivo** cuja solução não é imediata, pelo que requer a pesquisa de uma solução.

A sua formulação inclui a descrição de um **ponto de partida**, um conjunto de **transições entre estados**, uma função que diz se um estado **satisfaz um objetivo** e por vezes ainda uma função para avaliar o **custo de uma solução**.

A **pesquisa** é o processo que de forma recursiva ou iterativa, executa transições sucessivas entre estados até atingir um que satisfaça o objetivo.

Pesquisa em árvore

Fonte adicional: GeeksforGeeks sobre tipos de pesquisa, consultada em 30/10/2020

Ao longo da pesquisa entre o estado inicial e o objetivo, as transições sucessivas entre estados estudadas vão originar ramificações que formam uma árvore.

Pesquisa informada. O algoritmo tem informação sobre o estado do objetivo através de uma função de avaliação (que geralmente informa da proximidade (relativa ou não) ao objetivo).

```
pesquisa_informada(Problema,FuncAval) retorna a Solução, ou 'falhou'
Estratégia ← estratégia de gestão de fila de acordo com FuncAval
pesquisa_em_arvore(Problema,Estratégia)
```

Pesquisa não informada. O algoritmo só conhece o estado inicial, transições e objetivo.

De forma genérica, o algoritmo desta pesquisa, descreve-se da seguinte forma:

```
pesquisa(Problema,Estratégia) retorna a Solução, ou 'falhou'
    Árvore ← árvore de pesquisa inicializada com o estado inicial do Problema
    Ciclo:
        se não há candidatos para expansão, retornar 'falhou'
        Folha ← uma folha escolhida de acordo com Estratégia
        se Folha contem um estado que satisfaz o objectivo
            então retornar a Solução correspondente
            senão expandir Folha e adicionar os nós resultantes à Árvore
    Fim do ciclo;
```

Em programação, a árvore pode ser implementada através de um **fila**, inicializada com o estado inicial e à qual são acrescentados os estados expandidos de acordo com a estratégia definida. A forma como a estratégia é definida distingue os vários métodos de pesquisa.

Pesquisa em profundidade

Nesta pesquisa não informada, a partir do nó raiz, cada um dos seus ramos é explorado ao máximo, antes de retroceder.

```
Implementado através de método que adiciona nós à cabeça da lista (LIFO)
```

Podem ser implementadas pequenas variações, como sem repetição de estados, de forma a prevenir ciclos infinitos, com limite de visitas em profundidade, ou com limite crescente, que consiste em resolver o problema para um limite e caso não seja encontrada solução aumentar esse limite (e assim sucessivamente).

```
pesquisa_em_profundidade(Problema) retorna a Solução, ou 'falhou'
retornar pesquisa_em_arvore(Problema,juntar_à_cabeça)
```

Pesquisa em largura

Esta pesquisa não informada foca-se em explorar os vários níveis hierárquicos (nós vizinhos) de forma progressiva desde a raiz.

```
Implementado através de método que adiciona nós ao final da fila (FIFO)
```

```
pesquisa_em_largura(Problema) retorna a Solução, ou 'falhou'
retornar pesquisa_em_arvore(Problema,juntar_no_fim)
```

Pesquisa A*

Esta é uma pesquisa informada com uma função de avaliação bem definida que dá primazia à exploração dos nós com melhor custo da raiz até ao objetivo em detrimento dos restantes.

```
f(n) = g(n) + h(n)

Com g(n) a dar o custo desde o nó inicial até ao atual e h(n) o estimado até ao objetivo.
```

Quando h(n) não é sobrestimada[8] garantidamente, pode dizer-se que é admissível e que a pesquisa encontra sempre uma solução ótima.

8 Quando se atribuir um valor superior ao real.

Pesquisa de custo uniforme

Esta é uma variante da pesquisa A*, também conhecida por algoritmo de Dijkstra, onde a função de avaliação dá primazia aos nós com base no custo desde o nó inicial.

```
f(n) = g(n) + 0, com g(n) a dar o custo desde o nó inicial até ao atual
```

Tem um comportamento semelhante à pesquisa em largura.

Caso exista solução, a primeira encontrada é a solução ótima.

Pesquisa gulosa

Este método foca-se na proximidade ao objetivo.

```
f(n) = h(n), com h(n) a dar o valor estimado até ao objetivo
```

Tem um comportamento semelhante à pesquisa em profundidade, com a variante de que chega mais rápido à solução.

Ao ignorar o custo acumulado, facilmente deixa escapar a solução ótima.

Avaliação das estratégias de pesquisa

Dependendo das exigências do contexto em que são empregados, os vários métodos de pesquisa podem ser mais adequados para uns do que para outros. Por isso, a classificação destes métodos é dividida em quatro eixos.

Completude é a capacidade de encontrar uma solução quando ela existe

Complexidade temporal é o tempo que demora a encontrar a solução

Complexidade espacial é a memória necessária para encontrar uma solução

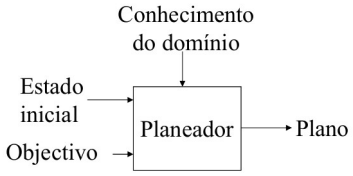
Optimalidade é a capacidade de encontrar de entre as várias, a melhor solução

	Completude	Complexidade		Optimalidade	
		Temporal	Espacial	Ações	Custo
Largura	✓	O(exp(p))		✓	✗
Profundidade	✓	O(exp(p))		✗	✗
Informada	✓	O(exp(p))		Depende da estratégia	

Da resolução de problemas ao planeamento

Fonte adicional: Capítulo 11 do livro "Artifical Intelligence: a modern aproach", de Stuart Russel e Peter Norvig, sobre o planeamento

Nem sempre a resolução de problemas é tão linear como a analisada até aqui. Há cenários em que o **conhecimento do domínio** implica para além das transições entre estados, do conhecimento das condições de aplicabilidade e efeitos das ações possíveis.



Por exemplo, não faz sentido aplicarmos a resolução de problemas à situação de que é necessário ir buscar leite e cereais, sendo o estado inicial estar em casa. O número de estados e sub-estados consequentes é brutalmente grande, tornando a busca muito pouco eficiente.

Mesmo com base em heurísticas, estas dizem se a resolução em estudo está ou não próxima do objetivo, sem permitir a eliminação de ações (por exemplo, neste caso não faria sentido ir à padaria ou à oficina).

O **planeamento** caracteriza-se então por três ideias chave:

1. Utilização de descrições em linguagem formal (p.e. lógica de primeira ordem);

Os estados e os objetivos são descritos em linguagem formal, assim como as ações, numa lógica com pré-condições e efeitos.

2. O planeador é livre de adicionar ações ao plano sempre que necessário, sem necessidade de estabelecer uma sequência de ações desde o estado inicial;

Não há uma conexão necessária entre a ordem do planeamento e a ordem da execução.

3. A maior parte do mundo é desconexo (partes independentes);

Por isso, em algumas situações faz sentido aplicar o *divide-and-conquer* e juntar as várias soluções.

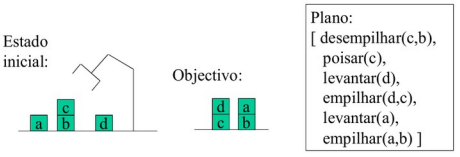
STRIPS

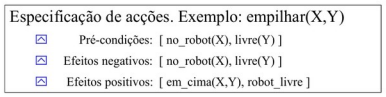
Uma abordagem bastante conhecida ao planeamento de soluções é a **linguagem STRIPS** (Stanford Institute Problem Solver), desenvolvido nos anos 70.

A funcionalidade de um dado tipo de operação é definida pelos seus formalismos, através de uma estrutura denominada **operador**, que é composta da seguinte forma:

Pré-condições. Um conjunto de fórmulas atómicas que representam as condições de aplicabilidade deste tipo de operação.

Efeitos negativos. Um conjunto de fórmulas atómicas que representam propriedades do mundo e que deixam de ser verdade ao executar-se a operação.





Efeitos positivos. Um conjunto de fórmulas atómicas que representam propriedades do mundo e que passam a ser verdade ao executar-se uma operação.

Pesquisa num grafo de estados

Numa pesquisa em árvore muito simplificada e em profundidade, facilmente se criam ciclos infinitos por repetição de estados em cada caminho estudado. Este problema pode ser resolvido com o registo dos estados estudados em cada caminho.

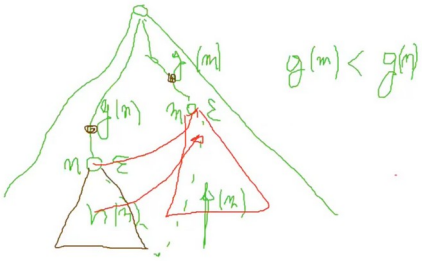
No entanto, não previne que transições a partir de diferentes estados levem ao mesmo estado. A **pesquisa num grafo de estados** tem como objetivo evitar este acontecimento de forma a tornar a pesquisa mais eficiente.

Assim, para além de implementar uma fila de nós abertos (não expandidos), guarda também uma lista de **nós fechados** (já expandidos), para evitar a repetição de estados.

A sua implementação é semelhante ao da pesquisa em árvore, distinguindo-se apenas em algumas etapas.

```
Quando um nó é aberto, é retirado da lista dos abertos e adicionado à lista dos fechados

De seguida, se nó não satisfizer o objetivo, é expandido e para cada expansão X:

- Se X não está numa das listas (abertos ou fechados), ligá-lo a N

- Se X está numa das listas, ligá-lo a N SSE o melhor caminho passar por N

- Adicionar os novos nós a abertos e reordenar
```

Na prática, se um nó E voltar a ser encontrado e já tiver sido visitado anteriormente, só se o custo até E neste novo caminho for menor que o custo no caminho anteriormente estudado é que o nó E vai ser considerado.



Não se aplica ao A* porque ele já chega a cada estado pelo caminho ótimo.

Avaliação da pesquisa em árvore

Existem vários critérios que permitem avaliar a performance dos vários métodos de pesquisa em árvore que podemos utilizar. Os **fatores de ramificação** dão-nos uma métrica para esta avaliação.

A **ramificação média** consiste no quociente entre o número de nós resultantes da expansão (nós filho) pelo número de nós expandidos (mesmo que não tenham gerado filhos). Indica-nos a dificuldade do problema.

$$RM = \frac{N-1}{X}$$

N – número de nós da árvore de pesquisa no momento em que se encontra a solução
X – Número de nós expandidos (não terminais)

O **fator de ramificação efetivo** indica a eficiência da técnica de pesquisa utilizada e tenta normalizar a ramificação, determinando o número de nós filho por nó (B), numa árvore com ramificação constante (progressão geométrica).

$$1 + B + B^2 + \ldots + B^d = N \quad \text{ou seja:} \quad \frac{B^{d+1} - 1}{B - 1} = N$$

d – comprimento do caminho na árvore correspondente à solução

Resolve-se por métodos numéricos, ou seja, tentativa e erro.

Os fatores de ramificação costumam sair em exame!

Geração automática de heurísticas para pesquisa em árvore

Recordando, uma heurística diz-se admissível quando não sobreestima o valor real.

Nem sempre o problema a ser resolvido tem a si inerentes heurísticas claras e facilmente determináveis. Na resolução de problemas complexos podemos então optar por definir uma versão simplificada do problema, relaxando algumas restrições que não são críticas.

Uma vez determinada uma **heurística real** que permita obter uma solução ótima no problema simplificado, podemos aplicá-la ao problema original.

Por exemplo em STRIPS, no mundo dos blocos (aula prática), a condição crítica serão as pilhas de blocos, pelo que podemos criar uma heurística baseada na correção do empilhamento dos blocos.

Caso hajam várias heurísticas admissíveis, estas podem ser combinadas numa nova heurística, dada pelo máximo das heurísticas.

$$h(n) = max(h_1(n), h_2(n), \ldots, h_m(n))$$

Pesquisa em árvore avançada

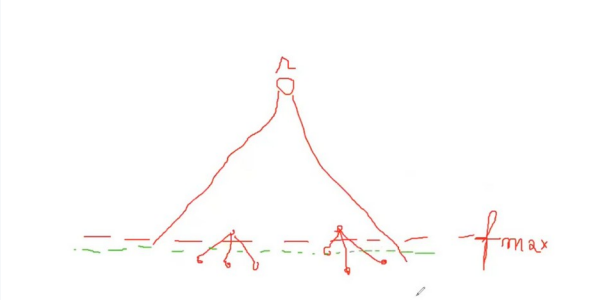
Apesar de permitir obter a **solução ótima**, a pesquisa **A*** tem um comportamento exponencial (tanto em tempo como em memória), pelo que para problemas mais complexos poderá ser preciso utilizar algoritmos mais eficientes (ainda que menos ótimos), ou heurísticas mais aproximadas, sacrificando novamente a optimalidade.

Todas as estratégias apresentadas neste tópico são baseadas no A*, pelo que utilizam a sua função de cálculo dos custos.

f(n) = g(n) + h(n), g(n) custo desde raiz e h(n) custo até objetivo estimado

IDA*

Resulta de uma junção da pesquisa em profundidade com limite e da A*, dando origem a uma **pesquisa com profundidade crescente**.



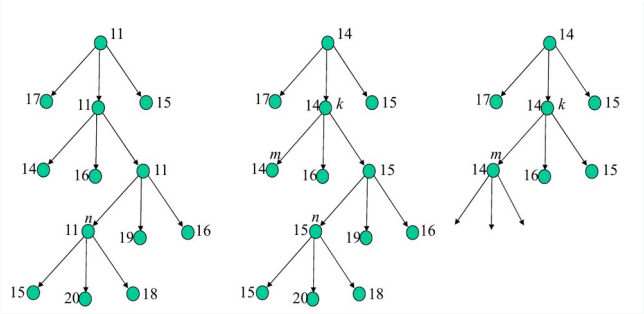
Começa por definir **fmáx = f(raíz).**

Executa pesquisa em profundidade com limite fmáx.

Atualiza **fmáx = menor{ f(n), tal que f(n)>fmáx na última execução }**

RBFS

Esta é uma estratégia que implementa **pesquisa recursiva melhor-primeiro**.

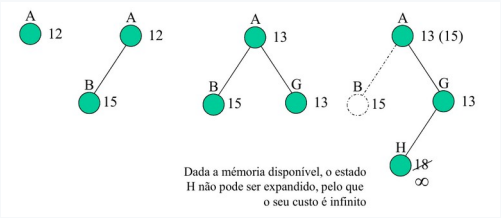


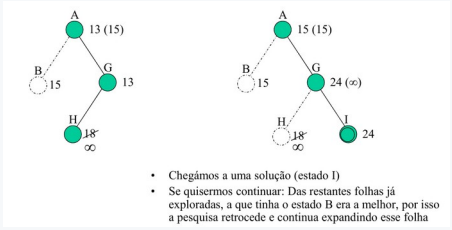
Para cada nó n, o algoritmo guarda o menor valor f(x), sendo x uma das folhas descendentes de n. Assim, sempre que um nó é expandido, os custos dos nós ascendentes são atualizados.

Quando a folha com menor custo (m) não é filha do último nó expandido (n), então o algoritmo retrocede até ao ascendente comum de m e n e avança para m (como na imagem).

SMA*

O SMA* implementa a mesma estatégia do RBFS, com a diferença de que **mantém o registo dos nós com custo superior ao ascendente em memória** (até um determinado limite de memória).





Quando a memória chega ao limite (no exemplo acima de 3), esquece o nó com maior custo f(n), atualizando nos ascendentes o custo do menor esquecido.

Em cada iteração é gerado apenas um sucessor, sendo gerados mais apenas caso o custo dos nós filho gerados seja superior ao do pai.

Quando se geraram todos os nós filho, o custo do pai é atualizado, tal como no RBFS.

Pesquisa com propagação de restrições

Este tipo de pesquisa é feito no espaço das soluções, tendo sido desenvolvido para atribuir valores a um conjunto de variáveis, respeitando um conjunto de restrições.

Responde a problemas como a distribuição de 8 rainhas num tabuleiro de xadrez de forma a que haja só uma em cada linha e em cada coluna e que não haja mais do que uma em cada diagonal.

A sua implementação está assente na construção de um **grafo de restrições**, cujos nós representam variáveis e cada arco (i,j) uma restrição de i imposta pelo valor de j, que se diz consistente se para cada valor de i, existe um valor de j que não viola as restrições.

Tipicamente utiliza-se uma estratégia de **pesquisa em profundidade** com iterações:

1. Seleciona-se um valor arbitrário possível para uma das variáveis;
2. Restrigem-se o conjunto dos valores possíveis para as restantes, por forma a que os arcos do grafo continuem consistentes.

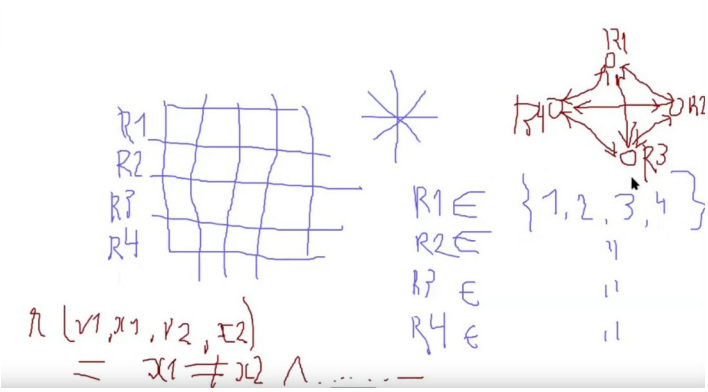
Contrariamente ao problema dos blocos, cada estado de pesquisa não representa uma situação ou configuração possível do mundo. Cada estado é constituído pelos conjuntos de valores possíveis para as variáveis.

No problema das rainhas simplificado à distribuição de 4 delas num tabuleiro 4x4, podemos definir um domínio em que cada rainha podia ficar em qualquer posição do tabuleiro, ou seja, teriam todas como domínio 16 posições entre (1,1) e (4,4).

Esta abordagem pode ser simplificada ao fixarmos cada rainha numa linha diferente, uma vez que já sabemos pelo problema que estas não podem ficar na mesma linha. Passamos a ter um domínio para cada variável { 1, 2, 3, 4 } com 4 posições apenas.

Em ambos os casos, o grafo de restrições tem arcos a ligar todas as variáveis, uma vez que a posição de cada peça cria restrições às posições das restantes.

A restrição é defina para, dadas duas variáveis e a sua posição, esta só é válida se as posições forem distintas. Esta é uma abordagem inicial e incompleta, uma vez que apenas considera a restrição de não estarem na mesma coluna. Falta a restrição das diagonais.



O **algoritmo de pesquisa** é semelhante à pesquisa em árvore, distinguindo-se por não haver um estado objetivo e por ocorrer num espaço de soluções, em oposição a um espaço de estado, começando com uma solução vazia, sobre a qual são fixados valores para as diferentes variáveis até ter uma solução completa.

1. Inicialização: o nó inicial da árvore de pesquisa é composto por todas as variáveis e todos os valores possíveis para cada uma delas
2. Se pelo menos uma variável tem um conjunto de valores vazio, falha e retrocede; se não puder retroceder, a pesquisa falha
3. Se todas as variáveis têm exactamente um valor possível, tem-se uma solução; retornar com sucesso
4. Expansão: Escolher arbitrariamente uma variável V_k e, de entre os valores possíveis, um dado valor X_{kl} – descartar os restantes valores possíveis dessa variável
5. Propagação de restrições: para cada arco (i,j) no grafo de restrições, remover os valores na variável V_i por forma a que o arco fique consistente
6. Caso tenha sido preciso remover valores na origem de algum arco, voltar a repetir o passo 5.
7. Voltar ao passo 2.

A propagação de restrições (passos 5 e 6) é feita com base no algoritmo abaixo.

```
propagarRestricoes(grafoRestricoes,FilaArestas) retorna o grafo de restrições com
    domínios possivelmente mais limitados
enquanto FilaArestas não vazia fazer {
    (Xj,Xi) ← remover cabeça de FilaArestas
    remover valores inconsistentes em Xj
    se removeu valores, então
        para cada vizinho Xk, acrescentar (Xk,Xj) a FilaArestas
}
```

Voltando a pegar no problema das 4 rainhas.

O nó inicial vai ser composto por todas as variáveis e todos os seus valores possíveis.

Este nó é expandido de forma a que seja considerada a fixação de todas as variáveis para todos os seus valores possíveis, que resulta em 16 nós. Destes, é escolhido um arbitrariamente, sobre o qual é verificada a propagação de restrições.

No exemplo ilustrado na imagem ao lado, começar a ficar a variável R1 na coluna 1 vai falhar, porque vamos obter uma variável com um conjunto de valores vazios, a R4.

Esta é uma solução muito mais eficiente do que a pesquisa por força bruta.
Sem restrições, o número de nós gerado seria…
0 – 1 (estado inicial)
1 – 4 * 4 = 16 (4 estados para as 4 rainhas)
2 – 16 * 3 * 4 = 192 (4 estados para as 3 rainhas não definidas em cada estado)
3 – 192 * 2 * 4 = 1536
4 – 1536 * 1 * 4 = 6144
PIOR CASO seria 7883 nós

Pesquisa com ordem de fixação pré-definida

Com ou sem propagação de restrições, há ainda uma forma de tornar o processo de pesquisa mais eficiente, que é a **pesquisa com ordem de fixação pré-definida**, que consiste na fixação de uma variável apenas a cada iteração.

Sem propagação de restrições e apenas com **ordem de fixação pré-definida**, o número de nós gerado para o problema das rainhas, seria:
0 – 1 (estado inicial)
1 – 4 (valores possíveis para R1)
2 – 4 * 4 = 16 (para cada um dos valores anteriores, é fixada R2, que tem também 4 valores possíveis)
3 – 16 * 4 = 64
4 – 64 * 4 = 256
PIOR CASO seria 341 nós

Com esta solução não se repetem estados, que de outra forma, com a fixação de todas as variáveis no primeiro nível da pesquisa iriam ser geradas em algum nível da árvore de pesquisa.

Tipos de restrições

As **restrições** podem ser de três tipos.

Unárias envolvem apenas uma variável. São aplicadas no pré-processamento (antes da pesquisa).

Por exemplo uma restrição unária pode definir que "os valores de determinada variável têm de ser pares". Quando aplicada em pré-processamento, vai remover todos os valores ímpares atribuídos à variável.

Binárias envolvem duas variáveis (aresta do grafo)

De ordem superior envolvem três ou mais variáveis (podem no entanto ser decompostas num conjunto das anteriores)

Temos como exemplo o quebra-cabeças criptoaritmético.

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

Variáveis principais: F, O, R, T, U, W $(\in \{ 0 .. 9 \})$
Variáveis internas: X_1 (transporte das unidades para as dezenas) e X_2 (transporte das dezenas para as centenas) $(\in \{ 0, 1 \})$
Restrições:

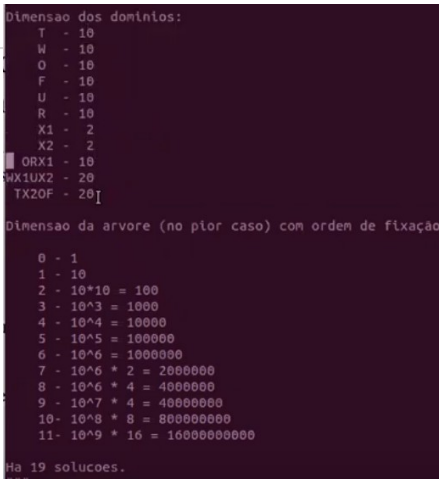
- Todas as variáveis são diferentes [restrição sobre 6 variáveis]
- $2 \cdot O = R + 10 \cdot X_1$ [restrição sobre 3 variáveis]
- $2 \cdot W + X_1 = U + 10 \cdot X_2$ [restrição sobre 4 variáveis]
- $2 \cdot T + X_2 = O + 10 \cdot F$ [restrição sobre 4 variáveis]

No exemplo anterior, a restrição ternária $2 \cdot O = R + 10 \cdot X_1$ pode ser transformada no seguinte conjunto de restrições:

- Restrições binárias:
  - O = primeiro(Aux)
  - R = segundo(Aux)
  - X_1 = terceiro(Aux)
- Restrição unária:
  - $2 \cdot$ primeiro(Aux) = segundo(Aux) + $10 \cdot$ terceiro(Aux)

Aux é uma variável auxiliar cujo domínio é o produto cartesiano dos domínios de O, R e X_1.

- Ou seja: $Aux \in \{ 0 .. 9 \} \times \{ 0 .. 9 \} \times \{ 0, 1 \}$
- A restrição unária sobre Aux pode ser satisfeita através de pré-processamento



Pesquisa por melhorias sucessivas

Também conhecida por **pesquisa local**, esta é uma técnica que parte de uma solução inicial fraca (que pode ser aleatória) e que procura otimizá-la de forma iterativa.

Há várias técnicas para esta otimização, mas partem todas de uma **heurística**, que avalia a proximidade da solução atual à ideal.

Montanhismo

Esta técnica procura otimizar a função de avaliação, analisando o seu espaço de valores como uma paisagem de vales (zonas de soluções menos satisfatórias) e colinas (zonas de soluções melhores) com o objetivo de atingir o pico de uma colina.

Para isto escolhe sempre o sucessor com melhor valor da função, sem retrocessos, até que o valor da função seja superior ao do seu(s) sucessor(es), quando pára e retorna a solução atual.

Na realidade esta não será a solução ideal, porque pode ser apenas um **máximo local** e não um global. Os **planaltos** são também um problema, uma vez que as alterações à solução não têm impacto na função de avaliação e podem ser em vão, caso se siga uma **ravina**.

O montanhismo "puro" apresenta assim algumas desvantagens, pelo que foram desenvolvidas algumas variantes com vista a colmatá-los, que deixam de querer sempre maximizar a função de avaliação, deixando que ela piore momentaneamente para atingir um fim maior.

Montanhismo estocástico[9] Escolhe aleatóriamente entre os sucessores que melhoram a função de avaliação.

Para quatro ações possíveis em que duas melhoram e duas pioram, escolhe aleatóriamente entre as duas primeiras.

Montanhismo de primeira escolha Escolhe sucessores aleatóriamente até encontrar um com melhor função de avaliação que o estado atual.

Montanhismo com reinício aleatório Executa o montanhismo várias vezes, a partir de estados iniciais aleatórios e escolhe a melhor solução.

Recozimento simulado

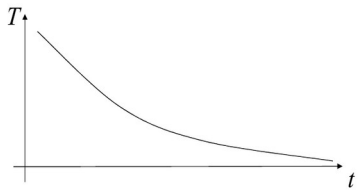
Recozimento[10] simulado

Consiste numa variante do montanhismo, em que o sucessor é selecionado aleatoriamente, sendo aceite quando a sua função de avaliação é superior à do nó atual **ou** com uma probabilidade que diminui exponencialmente em função da perda na função de avaliação. A pesquisa termina quando o indicador "temperatura" chega a zero.

9 Diz-se dos processos que não estão submetidos senão a leis do acaso.

10 Na indústria da cerâmica e do vidro este termo significa cozer uma peça e deixá-la arrefecer muito lentamente para não partir.

```
recozimento_simulado(Problema,Regime_termico,Aval)
(* A função Regime_termico dá a temperatura em função do tempo. *)
Nó ← fazer_nó(estado inicial do Problema)
repetir para t=0 .. ∞: {
    T ← Regime_termico(t)
    se T=0, retornar a solução de Nó
    Prox ← um sucessor de Nó gerado aleatoriamente
    Ganho ← Aval(Prox)-Aval(Nó)
    se Ganho>0, Nó ← Prox
    senão, com probabilidade exp(Ganho/T), fazer: Nó ← Prox
}
```



- $t \rightarrow \infty$
- $T \rightarrow 0$
- $Ganho/T \rightarrow -\infty$ (dado que o Ganho é negativo)
- Probabilidade: $\exp(Ganho/T) \rightarrow 0$
- Ou seja: À medida que o tempo passa, a pesquisa arrisca cada vez menos quanto a aceitar alterações com ganho negativo

Se a temperatura T diminuir de forma suficientemente lenta, o recozimento simulado encontra um máximo global.

Pesquisa local alargada

Esta é mais uma variante do montanhismo, mas em cada iteração são mantidos k estados e os melhores k sucessores são passados para a iteração seguinte.

Na realidade esta abordagem não funciona muito bem, sendo aplicadas melhorias a este algoritmo.

Pesquisa alargada estocástica Semelhante à anterior, mas os sucessores são selecionados aleatoriamente.

A pesquisa local alargada é como uma pesquisa greedy, procurando sempre maximizar o ganho, mas esquece-se do que já andou. A pesquisa alargarda estocástica por outro lado demora mais tempo a chegar ao melhor resultado.

Algoritmos genéticos Variante da anterior, mas com os sucessores gerados por combinação de dois estados, em vez de modificar um único estado.