

INE5371

Inteligência Artificial

Ementa

- ♦ Técnicas de IA Aplicadas à Resolução de Problemas.
- ♦ IA Simbólica e Não Simbólica

Professor Mauro Roisenberg

mauro@inf.ufsc.br

<http://www.inf.ufsc.br/~mauro>

Tópicos

- ♦ Histórico, Conceitos, Teoria de Problemas
- ♦ Modelagem de Agentes Inteligentes
- ♦ Revisão de Lógica
- ♦ Representação de Conhecimento
- ♦ Mecanismos de Raciocínio
- ♦ Sistemas Especialistas
- ♦ Redes Neurais Artificiais
- ♦ Conexionismo e Resolução de Problemas
- ♦ Outras Técnicas de IA (Lógica Nebulosa, Computação Evolutiva, etc...)

Avaliação

- ♦ 2 Provas + n Trabalhos

$$MF = [(P1+P2)/2] \times 0,6 + [(T1+T2+\dots+Tn)/n] \times 0,4$$

MF ≥ 6 : Aprovado

3 ≤ MF < 6 : Recuperação

MF < 3 : Reprovado

Índices Variáveis



Bibliografia

- Russel & Norvig - Artificial Intelligence: A modern approach
- E. Rich & K. Knight - Inteligência Artificial
- J.M. Barreto - Inteligência Artificial: Uma abordagem híbrida
- G. Bittencourt - Inteligência Artificial: Ferramentas e teorias
- R. A. Rabuske - Inteligência Artificial

Introdução

- ♦ Afinal, pra que estudamos Inteligência Artificial?
 - Existem 3 tipos de problemas
 1. Os que não têm solução.
Não há nada a fazer...
 2. Os que têm solução algorítmica
Ótimo. Basta codificar os algoritmos...
 3. Os outros....
 - Aqueles em que a solução algorítmica têm complexidade NP-Completa;
 - Aqueles que o Ser Humano é capaz de resolver;
 - Aqueles que os Seres Vivos são capazes de resolver.
Jogar Xadrez, Jogar Futebol, Reconhecer Faces, Fazer Traduções, Procurar Comida, Reconhecer Letras, etc, etc...

É AQUI QUE ENTRA A I.A.!!!!

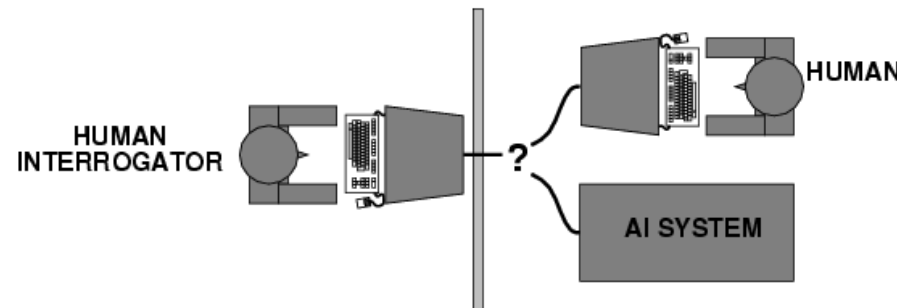
Histórico e Conceitos Básicos

- ♦ 1a. Pergunta: O que é Inteligência Artificial?
- ♦ Algumas Respostas:
 - A automatização das atividades que associamos com o pensamento humano, atividades tais como tomada de decisões, resolução de problemas, aprendizado,... (Bellman, 1978)
 - O estudo de como fazer os computadores realizarem coisas que, hoje em dia são feitas melhores pelas pessoas. (Rich & Knight, 1991)
 - O estudo das faculdades mentais através de modelos computacionais. (Charniak & McDermott, 1985)
 - O ramo da ciência da computação que se ocupa da automatização do comportamento inteligente. (Luger & Stubblefield, 1993)

Sistemas que PENSAM como HUMANOS	Sistemas que PENSAM RACIONALMENTE
Sistemas que ATUAM como HUMANOS	Sistemas que ATUAM RACIONALMENTE

Sistemas que Agem como Humanos

- Turing (1950) "Computing machinery and intelligence":
- "As máquinas podem pensar?"
- Teste Operacional para a Inteligência: O Jogo da Imitação



- Era previsto que por volta do ano 2000 uma máquina teria uma chance de 30% de enganar um leigo por 5 minutos.
- Teste ainda relevante nos dias atuais, apesar de se preocupar com a questão errada.
- Requer o desenvolvimento de várias áreas da IA: conhecimento, raciocínio, compreensão da linguagem natural, aprendizado, etc

Sistemas que Pensam como Humanos

- ♦ Como os seres humanos pensam?
- ♦ Necessita teorias científicas sobre as atividades internas do cérebro (modelo cognitivo):
 - Nível de abstração? (conhecimento ou circuito?)
 - Como validar?
 - Predizendo e testando o comportamento humano
 - Identificação a partir de dados neurológicos
 - Ciência Cognitiva vs. Neurociência Cognitiva.
- ♦ Ambas as abordagens são hoje em dia separadas das IA

Sistemas que Pensam Racionalmente

- ♦ Capturam as leis do raciocínio
- ♦ Aristóteles: O que é Argumentação Correta e processos de raciocínio?
 - A correção depende da irrefutabilidade dos processos de raciocínio lógico.
- ♦ Estes estudos iniciaram o campo da LÓGICA.
- ♦ A tradição logicista da IA espera criar sistemas inteligentes usando programação lógica.
- ♦ Problemas:
 - Nem todo comportamento inteligente emerge de um comportamento lógico

Histórico e Conceitos Básicos

- ♦ UMA BOA DEFINIÇÃO
A grande atividade da IA é a solução de problemas usando e manipulando conhecimento.
- ♦ Formalmente a área foi criada em 1956 quando o nome foi cunhado por John McCarthy no encontro do Dartmouth College, onde se reuniram os primeiros pesquisadores da área.
- ♦ Entretanto, há mais de 2000 anos, filósofos, psicólogos e cientistas estudam como o ver, aprender, recordar e raciocinar pode ser realizado.
- ♦ UM POUCO DE FILOSOFIA
 - Um dia será possível entender completamente a inteligência humana?
 - Cérebro e mente são a mesma coisa?
 - Existe a alma e o livre arbítrio?

Histórico e Conceitos Básicos

É possível dividir as fases da história da Inteligência Artificial com os seguintes períodos:

1. ÉPOCA PRÉ-HISTÓRICA

(Nesta época nada se conhecia sobre os mecanismos da mente, nem sob o prisma fisiológico nem psicológico e por esta razão vai até 1875 quando Camilo Golgi visualizou o neurônio)

- **Objetivo:** Criar seres e mecanismos apresentando comportamento inteligente.
- **Metodologia e conquistas:** Mecanismos usando mecânica de precisão desenvolvida nos autômatos, mecanismos baseados e teares, etc. Apelo ao sobrenatural.
- **Limitações:** Complexidade dos mecanismos, dificuldades de construção. Insucesso dos apelos ao sobrenatural.

Histórico e Conceitos Básicos

2. ÉPOCA ANTIGA (1875-1943)

(Modelo de Neurônio de McCulloch & Pitts)

- Época em que a lógica formal apareceu (Russel, Gödel, etc) bem como se passou a reconhecer o cérebro como órgão responsável pela inteligência. Hilbert imaginava um mundo paradisíaco, em que tudo poderia ser axiomatizado e reduzido à Lógica. Entretanto assim como o final do século XIX viu o desmoronamento do mundo Euclidiano, Gödel abalou o mundo de Hilbert com seu teorema de incompletude da aritmética. Foi a época em que, tal como os filósofos gregos fizeram, são colocadas as bases da IAS e IAC, terminando com a publicação do trabalho de McCulloch e Pitts modelando o neurônio.
- **Objetivo:** Entender a inteligência humana
- **Metodologia e conquistas:** Estudos da psicologia e de neurofisiologia. Nascimento da psicanálise.
- **Limitações:** Grande distância entre as conquistas da psicologia e da neurofisiologia.

Histórico e Conceitos Básicos

3. ÉPOCA ROMÂNTICA (1943-1956)

(É o otimismo desordenado, que tem um jovem rapaz romântico crê que tudo é possível. Acaba com a reunião no Dartmouth College)

- **Objetivo:** Simular a inteligência humana em situações pré-determinadas.
- **Metodologia e conquistas:** Inspiração na natureza. Nascimento Cibernético. Primeiros mecanismos imitando funcionamento de redes de neurônios. Primeiros programas imitando comportamento inteligente.
- **Limitações:** Limitação das capacidades computacionais.

Histórico e Conceitos Básicos

4. ÉPOCA BARROCA (1956-1969)

(livro Perceptrons)

- Tudo é fácil e será conseguido.
- Provadores Automáticos de Teoremas.
- Acreditava-se na tradução automática entre linguagens.
- Acreditava-se ser possível construir um programa para resolver qualquer problema.
- Em alguns anos um computador ganharia o campeonato mundial de xadrez.
- **Objetivo:** Expandir ao máximo as aplicações da IA tanto usando a abordagem simbólica quanto a conexionista.
- **Metodologia e conquistas:** Perceptron. Primeiros sistemas especialistas usando a abordagem simbólica. Grandes esperanças da IAS.
- **Limitações:** Dificuldades em técnicas de aprendizado de redes complexas; Subestimação da complexidade computacional dos problemas.

Histórico e Conceitos Básicos

4. ÉPOCA BARROCA (1956-1969)

(livro Perceptrons)

- ♦ Princípio da IA Simbólica (IAS)
“Hipótese do Sistema de Símbolos Físicos” (Newell & Simon):
 - A inteligência é o resultado da manipulação de símbolos que representam o mundo.
- ♦ Princípio da IA Conexionista (IAC)
“Metáfora Biológica”:
 - Se for construído um modelo do cérebro, este modelo apresentará um comportamento inteligente.
- ♦ Subestimação da Complexidade Computacional:
 - Os problemas de IA são comumente de complexidade NP-Completa.

Histórico e Conceitos Básicos

5. ÉPOCA DA TREVAS (1969-1981) (5a. Geração)

- ♦ Paralisação de quase todas as pesquisas em IA por falta de verbas. Acabou quando os japoneses anunciaram seus planos para a Quinta Geração de Computadores e em outro ambiente Hopfield publica célebre artigo sobre redes neurais.
- ♦ Assim como a Idade Média da História da humanidade viu florescer idéias novas, nesta época não foi de total trevas. Nasceram as primeiras aplicações dos conjuntos nebulosos de Zadeh, nascendo o controle inteligente com Mamdani. Além disto os sistemas especialistas se firmaram com Shortliffe.
 - **Objetivo:** Encontrar para a IA aplicações práticas.
 - **Metodologia e conquistas:** Sistemas especialistas. Aplicações principalmente em laboratórios. Os computadores usados principalmente para aplicações administrativas e numéricas. Interesse dos fabricantes de computadores de desmistificar a máquina levando a pouco interesse em IA.
 - **Limitações:** Era necessário muito conhecimento para tratar mesmo o mais banal problema de senso-comum; Interesses econômicos.

Histórico e Conceitos Básicos

6. RENASCIMENTO (1981-1987)

(Começou a corrida para IA. Os resultados obtidos nas épocas anteriores atingiram o público em geral. Sistemas especialistas se popularizaram. Primeira conferência internacional de Redes Neurais marca final do período).

- **Objetivo:** Renascimento da IA, simbólica e conexionista
- **Metodologia e conquistas:** Popularidade da linguagem Prolog, adotada pelos japoneses. Crescimento da importância da Lógica. Proliferação de máquinas suportando ferramentas para IA. Sistemas Especialistas capazes de simular o comportamento de um especialista humano ao resolver problemas em um domínio específico.
- Alguns poucos pesquisadores continuaram seus trabalhos em RNAs, Grossberg, Kohonen, Widrow, Hinton, etc. No final do período, trabalhos de Hopfield, do grupo PDP, etc., criaram condições para a fase seguinte no que diz respeito às RNAs.
- **Limitações:** a IAS e a IAC evoluindo separadamente.

Histórico e Conceitos Básicos

7. ÉPOCA CONTEMPORÂNEA (1987- atual)

(Logo no início do período Gallant publica seu cérebre artigo sobre sistemas especialistas connexionistas. Foi o ponto de partida para a união das duas abordagens de IA, tornando a abordagem dirigida problemas a abordagem atual.)

- **Objetivo:** Alargamento das aplicações das IAs. Uso em tomografia, pesquisas em campos de petróleo, e bases de dados inteligentes.
- **Metodologia e conquistas:** Redes diretas como aproximador universal. Lógica nebulosa usada largamente em indústrias para controle inteligente. Sistemas especialistas se torna tecnologia dominada. Bons resultados em problemas mal definidos com sistemas usando hibridismo neural-nebuloso. Novo paradigma de programação: programação connexionista.
- **Limitações:** Quem sabe??? Uma possibilidade é uma grande expansão das bases de dados inteligentes.

Teste de Turing para a Inteligência (1950)

- ♦ Visão de que computadores atuam como humanos.
 - Computador com as seguintes capacidades:
 - Processamento de linguagem natural;
 - Representação de conhecimento;
 - Raciocínio automático;
 - Aprendizado de máquina.

Domínios de Aplicação

- ♦ Resolução de problemas (planejamento)
 - Quebra-cabeça, jogos
 - Problemas que requerem conhecimento especialista (diagnóstico médico, localização de recursos minerais, configuração de computadores)
- ♦ Raciocínio por senso-comum
 - Simulação qualitativa ou intuitiva
 - Mecanismos de inferência
- ♦ Percepção (visão e fala)
 - Reconhecimento de objetos através de imagens
 - Reconhecimento de voz ou identificação de imagens

Domínios de Aplicação

- ♦ Processamento de linguagem natural
 - O que significa um conjunto de palavras
 - Tradução de idiomas
 - Acesso a dados em base de dados
- ♦ Extração de conhecimento
 - Knowledge Data Discovery
- ♦ Aprendizado
 - Desenvolver sistema que melhorem seu desempenho através de experiências
 - Desenvolver sistemas que auxiliem no aprendizado de alunos
- ♦ Programação
 - Desenvolvimento de "shells" para sistemas especialistas
 - Paralelização de linguagens de IA
 - Distribuição da resolução de problemas
 - Sistemas Multi-agentes

Teoria de Problemas

A IA se ocupa da resolução de problemas, para tal é necessário conhecimento sobre o problema e técnicas de manipular este conhecimento para obter a solução.

- ♦ **O que é um PROBLEMA?**

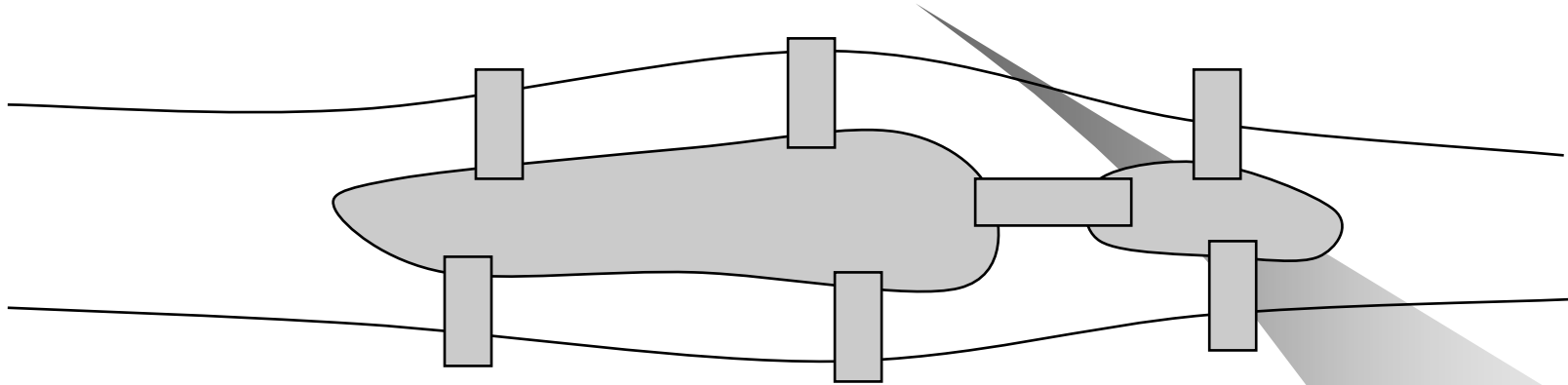
- Resolver um problema é diferente de ter um método para resolvê-lo.
- Antes de tentar buscar a solução de um problema, deve-se responder as seguintes perguntas:
 - Quais são os dados?
 - Quais são as soluções possíveis?
 - O que caracteriza uma solução satisfatória?

- ♦ **Exemplos:**

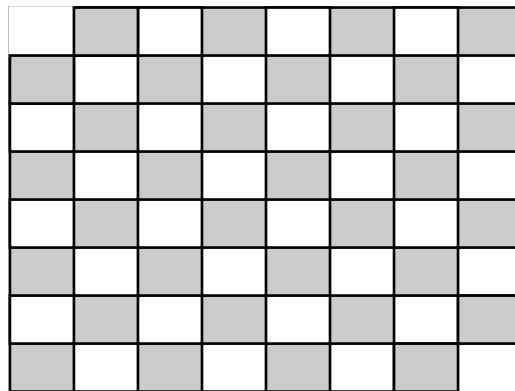
- As pontes de Königsberg
- O tabuleiro de xadrez mutilado

Teoria de Problemas

- As pontes de Königsberg

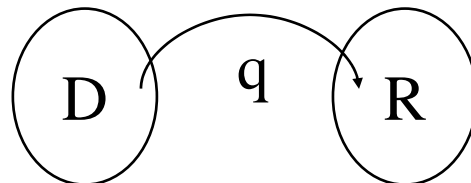


- O tabuleiro de xadrez mutilado



Teoria de Problemas

- ♦ **Definição:** Um problema é um objeto matemático $P=\{D,R,q\}$, consistindo de dois conjuntos não vazios, D os dados e R os resultados possíveis, e de uma relação binária $q \subset D \times R$, a condição que caracteriza uma solução satisfatória, associando a cada elemento do conjunto de dados a solução desejada.
- ♦ **Exemplo:** Um problema de diagnóstico médico
 - O conjunto de dados disponíveis $d \in D$ (observação da anamnese, sintomas, exames, etc.)
 - R é o conjunto de doenças possíveis
 - Solução satisfatória: encontrar o par (d,r) onde $r \in R$ é o diagnóstico correto.
- ♦ A definição de um problema permite testar se um certo elemento é ou não solução, mas não guia na busca deste elemento.



Teoria de Problemas

Modos de definir uma FUNÇÃO PROBLEMA

1. Por ENUMERAÇÃO EXAUSTIVA

- Fornece-se todos os conjuntos de pares (dado, resultado).
 - Ex.: Agenda de telefone.

2. DECLARATIVAMENTE

- Definir declarativamente um problema é dar propriedades que devem ser satisfeitas pela solução do problema.
 - Ex.: O avô de alguém é aquela pessoa que é pai do pai da pessoa.

3. Por um PROGRAMA (um algoritmo)

- Um programa de computador define a correspondência entre dados e resultados sempre que ele pára, conseguindo chegar a uma solução.
 - Ex.: Programa para declaração do imposto de renda.

4. Por EXEMPLOS

- O problema não completamente definido para todo valor de seus dados. Conhece-se para um subconjunto.
 - Ex.: Ensinar a pegar uma bola atirada no ar.

Teoria de Problemas

Os modos de definir uma função levam ao conceito de

♦ COMPUTABILIDADE

Definição 1: Uma função é dita computável se é possível calcular seu valor para todos os elementos de seu domínio de definição.

- Ex1.: Equações Diofantinas
 - $a^n + b^n = c^n$ $n \geq 3$ a, b, c Inteiros
- Ex2.: Problema da parada de um programa
 - Dado um programa e um conjunto de dados infinito, é impossível ter um outro programa que decida se o primeiro programa vai conseguir parar para todos os dados.

♦ COMPLEXIDADE

Definição 2: A complexidade de um problema, com relação a um conjunto bem definido de recursos, é definida como aquela que considera o modo mais parcimonioso de uso de recursos conhecidos para a solução do problema.

Teoria de Problemas

- ♦ Se a computabilidade diz respeito à existência de solução para um problema, a complexidade se refere a quantidade de recursos necessários para resolvê-los.
- ♦ Um mesmo problema pode ter complexidade diferente, dependendo da técnica que se utiliza para resolvê-lo.
- ♦ Definição 3: Um problema é dito NP-Completo quando não se conhece algoritmo de ordem polinomial capaz de resolvê-lo.
 - Ex.: Problema do caixeiro-viajante resolvido de maneira algorítmica.
- ♦ **HEURÍSTICAS**
Definição 4: Conjunto de regras e métodos que conduzem à descoberta, à invenção e à resolução de problemas.
- ♦ O papel das heurísticas - "boa solução"
 - Na IA as heurísticas são as "técnicas" que possibilitam tratar problemas NP-Completo e buscar algoritmos de ordem mínima para problemas polinomiais.

Teoria de Problemas

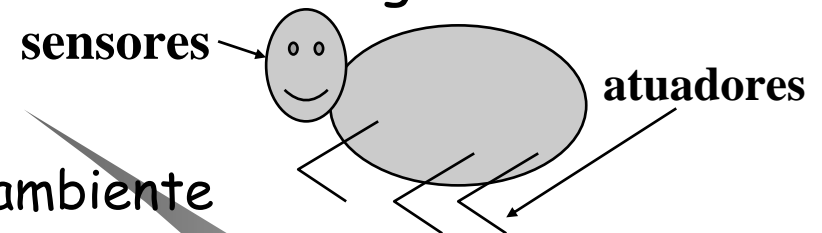
- ♦ Estratégias Básicas para Resolver Problemas (Estratégias constituem os modos básicos de raciocínio para resolver problemas)
Pela definição do problema, o qual se apresenta como uma função, estes modos de raciocínio devem se adaptar ao modo que a função foi definida.
 1. *Por enumeração exaustiva*: o conhecimento necessário para resolver o problema está na enumeração.
 2. *Declarativamente*: leva frequentemente a problemas de busca. "Utilizar um método de busca em que, por passos sucessivos se aproxima da solução, usando algumas vezes técnicas sem grande justificativa teórica". ESTA É A ABORDAGEM DA IA SIMBÓLICA!
 3. *Por exemplos*: Se o problema foi definido por exemplos, se deverá usar um método para aproximar a função. ESTA É A ABORDAGEM DA IA CONEXIONISTA!
- **ALGUNS PROBLEMAS CLÁSSICOS:**
 - Missionários e canibais; Torres de Hanói; Baldes de Água; Jogo do Oito; Reconhecimento de Caracteres, Previsão, etc.

Agentes Inteligentes

- Uma Ferramenta para Análise de Sistemas Inteligentes

- **O que é um agente???**

- Um agente é algo que percebe seu ambiente através de sensores e atua no ambiente através de atuadores.
 - Agente Humano, Agente Animal, Agente Robótico, Agente em Software, Termostatos, etc...
 - A Função Agente mapeia dados da percepção para ações.



- **O que é um agente racional???**

- O objetivo da IA, segundo Russel & Norvig é projetar agentes que façam um bom trabalho agindo no seu ambiente. O princípio básico da utilização de agentes é que eles devem "saber das coisas" (know things).
- Um agente racional ideal é aquele que, para cada possível seqüência de percepção, realiza uma ação que maximiza seu desempenho (mapeamento ideal), tendo como base as evidências fornecidas pela seqüência de percepções e pelos conhecimentos previamente existentes no agente.

Agentes Inteligentes

- ♦ Especificar que ações um agente deve tomar em resposta a qualquer sequência de percepções, leva ao projeto de um agente ideal.
 - Medida de Desempenho
 - Segurança, velocidade, destino, conforto, etc...
 - Ambiente
 - Observável, determinístico, episódico, estático,...
 - Atuadores
 - Rodas, pés, vídeos, mensagens, etc...
 - Sensores
 - Imagens, gps, teclados, mensagens, encoders, ultra-som,...
- ♦ A noção de agente pretende ser uma ferramenta para análise de sistemas inteligentes, não uma caracterização absoluta que divide o mundo em agentes e não-agentes.
 - Exemplos: Aspirador de pó, Agente de busca na Internet, Agente de auxílio a aprendizagem, Agente de auxílio ao diagnóstico médico, etc...

Agentes Inteligentes

- ♦ Tipos de Agentes

- Software Agents

- Agentes são considerados entidades computacionais baseadas na idéia de que os usuários necessitam apenas especificar um objetivo em alto nível ao invés de utilizar instruções explícitas, deixando as questões de como e quando agir a cargo do agente.
- Aplicações: Interfaces Amigáveis, Cartografia, Auxílio ao Ensino, Auxílio ao Diagnóstico Médico.

- Hardware Agents

- Agentes que operam em ambientes físicos (AGVs, Robôs, Embedded Systems, etc.)
- Agentes Físicos capazes de detectar mudanças ambientais e, através da reavaliação de seus objetivos encontrar uma nova seqüência de ações capazes de persegui-los, sem que esta seqüência tivesse sido prevista.

Agentes Inteligentes

- ♦ O que é um Agente Autônomo?
 - Agentes Autônomos são sistemas computacionais que operam em ambientes dinâmicos e imprevisíveis. Eles interpretam dados obtidos pelos sensores que refletem eventos ocorridos no ambiente e executam comandos em atuadores que produzem efeitos no ambiente.
 - O grau de "autonomia" de um agente está relacionado à capacidade de decidir por si só como relacionar os dados dos sensores com os comandos aos atuadores em seus esforços para atingir seus objetivos, satisfazer motivações, etc...

Agentes Reflexivos

- ♦ Não tem memória.
- ♦ Quando cessa a percepção, cessa a ação.
 - If car-in-front-is-braking (brake-light on)
 - then initiate-braking

function SIMPLE-REFLEX-AGENT (*percept*) **returns** *action*
static: *rules*, a set of condition-action rules

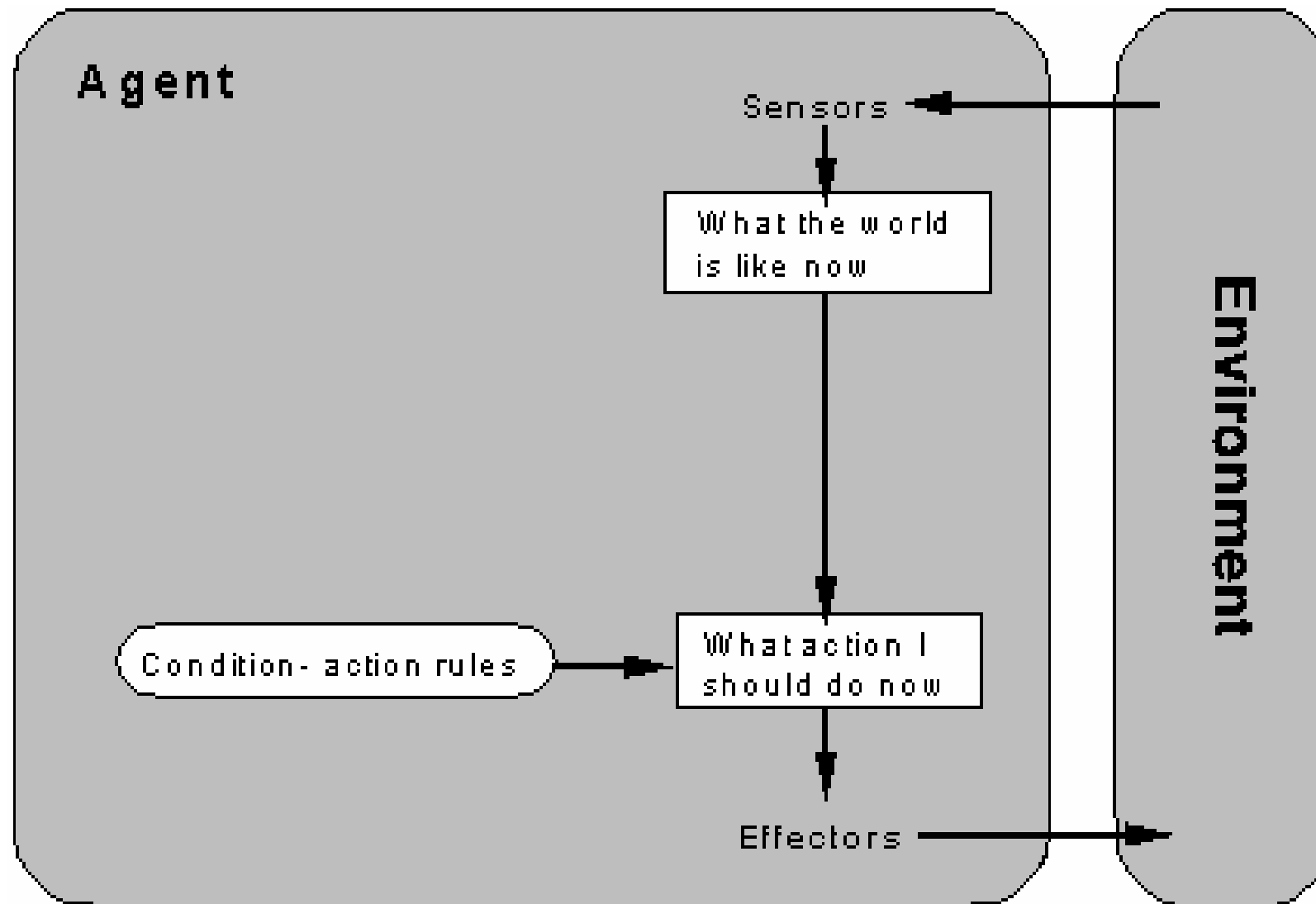
state := INTERPRET-INPUT(*percept*)

rule := RULE-MATCH(*state*, *rules*)

action := RULE-ACTION[*rule*]

return *action*

Agentes Reflexivos



Agentes com Estados Internos

- ♦ Guarda informações que não são percebidas no momento
 - Como o mundo evolui (modelo do mundo)
 - O que as ações provocam no mundo

function REFLEX-AGENT-WITH-STATE (*percept*) **returns** *action*

static: *state*, uma descrição do estado corrente do mundo

rules, a set of condition-action rules

state := UPDATE-STATE(*state*, *percept*)

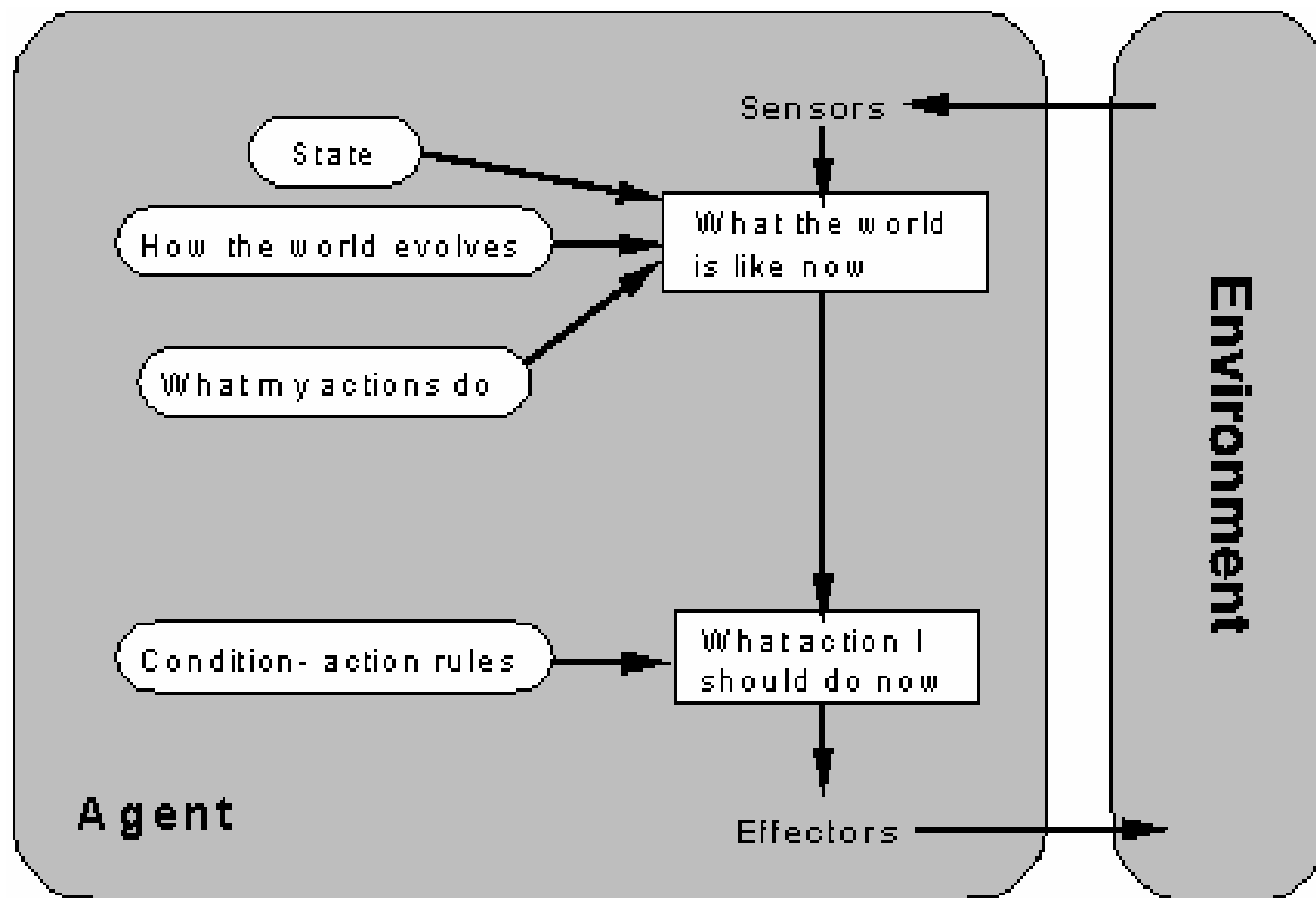
rule := RULE-MATCH(*state*, *rules*)

action := RULE-ACTION[*rule*]

state := UPDATE-STATE(*state*, *action*)

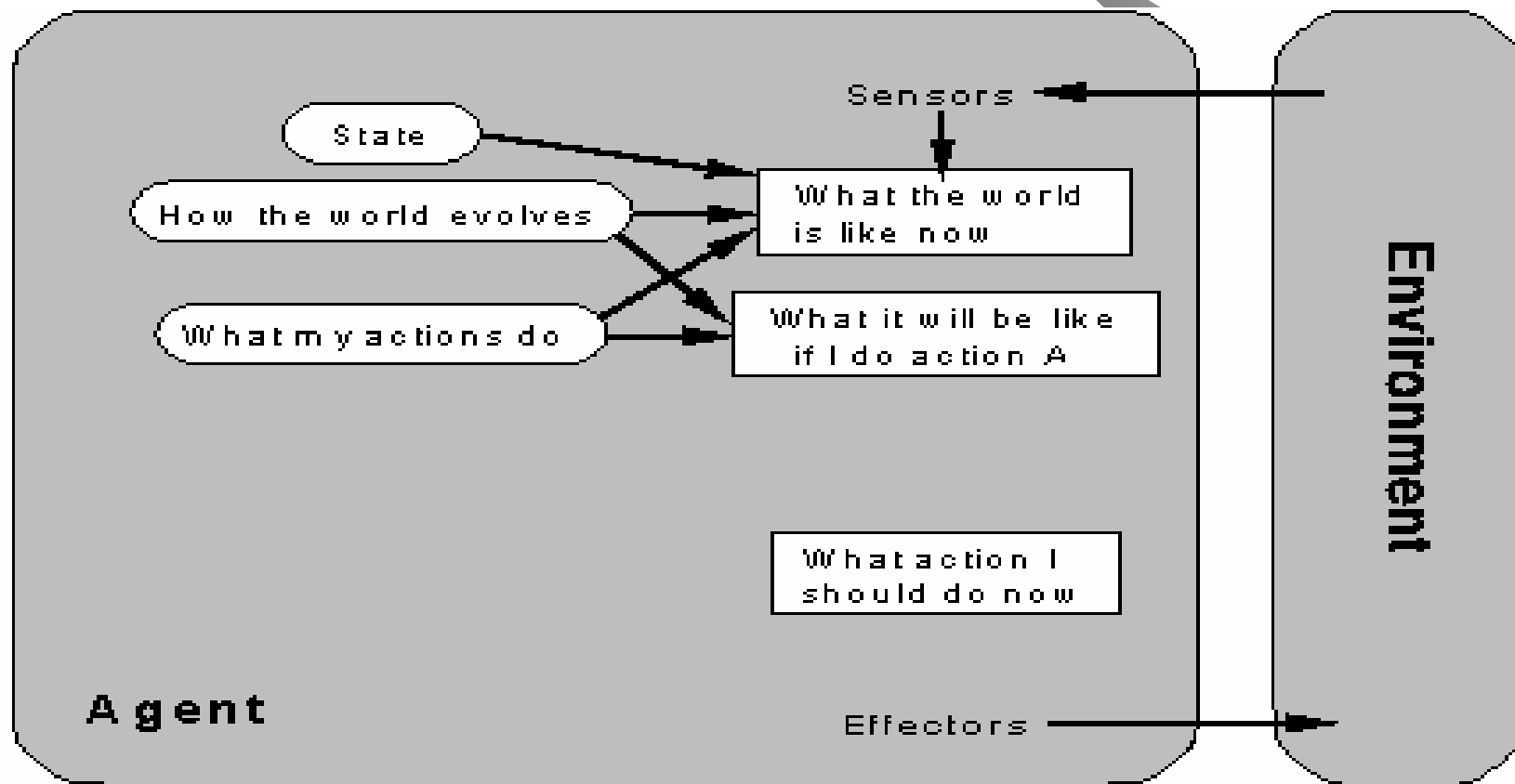
return *action*

Agentes com Estados Internos



Agentes com Metas

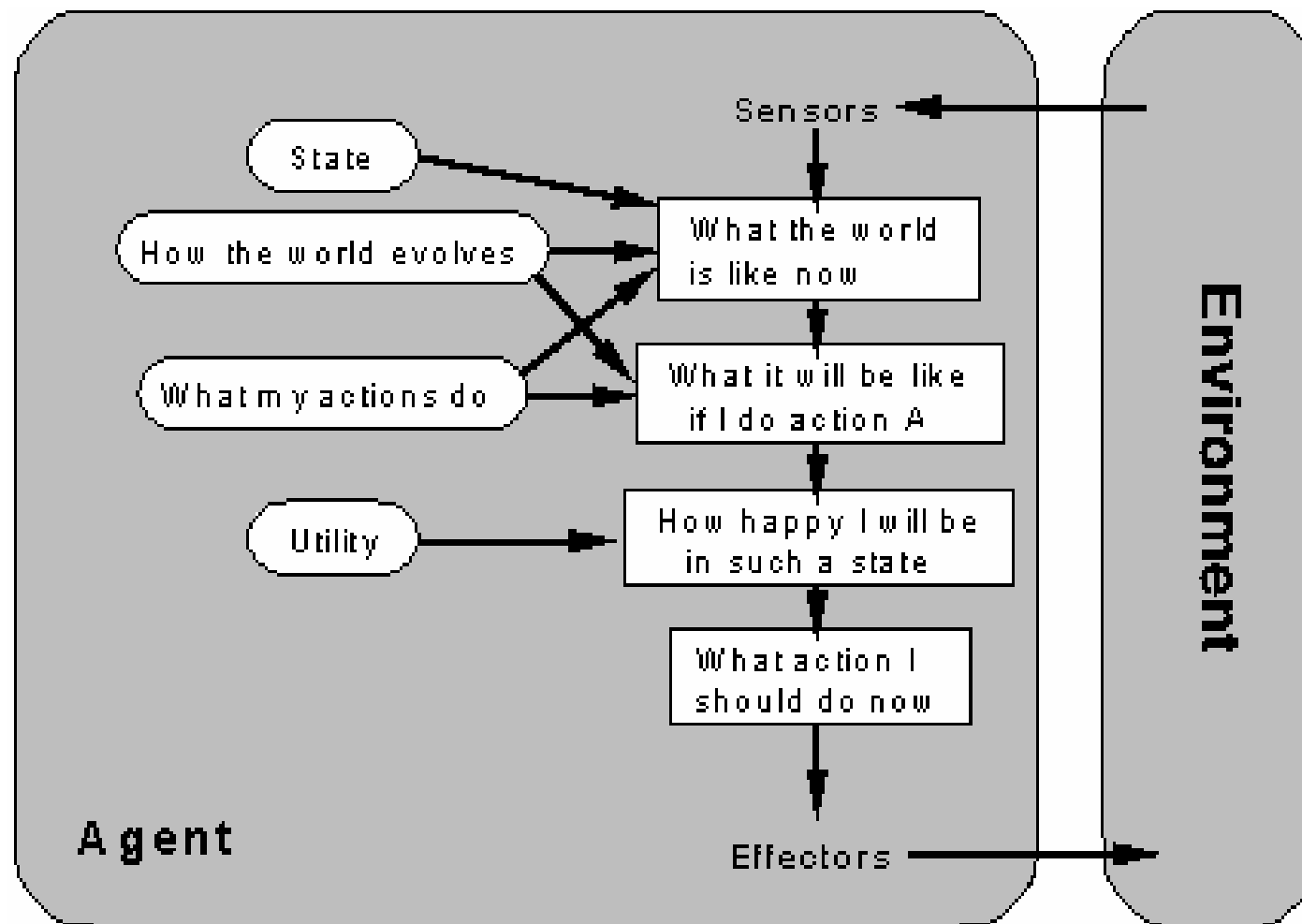
- ♦ Metas
- ♦ Busca e Planejamento são subcampos da IA cujo objetivo é achar seqüências de ações que conduzam ao objetivo do agente.



Agentes baseados em Utilidade

- ♦ Utilidade é uma função que mapeia um estado em um número real que descreve o grau de "felicidade" associado ao estado.
- ♦ Permite decisões racionais em casos em que o objetivo tem algum "problema";
- ♦ Quando existem objetivos conflitantes (p.ex.: velocidade e segurança);
- ♦ Quando existem vários objetivos, a utilidade "diz" qual tentar alcançar primeiro.

Agentes baseados em Utilidade



Propriedades dos Ambientes

- ♦ **Completamente Observável x Parcialmente Observável**
 - se o aparato sensor fornece acesso a uma descrição completa do ambiente.
- ♦ **Determinístico x Estocástico**
 - se o próximo estado do ambiente pode ser completamente determinado pelo estado atual do ambiente e pelas ações selecionadas pelo agente.
- ♦ **Episódico x Seqüencial**
 - a experiência do agente é dividida em episódios. Cada episódio consiste na percepção do agente e na sua ação. Não existe passado nem futuro.
- ♦ **Estático x Dinâmico**
 - se o ambiente se altera enquanto o agente está pensando, então o ambiente é dinâmico para o agente.
- ♦ **Discreto x Contínuo**
 - se existe um número finito de diferentes percepções e ações possíveis, então o ambiente é discreto.

Propriedades dos Ambientes

Ambiente	Observável	Determinístico	Episódico	Estático	Discreto
Xadrez com Relógio	SIM	SIM	NÃO	SEMI*	SIM
Xadrez sem Relógio	SIM	SIM	NÃO	SIM	SIM
Poker	NÃO	NÃO	NÃO	SIM	SIM
Gamão	SIM	NÃO	NÃO	SIM	SIM
Dirigir Taxi	NÃO	NÃO	NÃO	NÃO	NÃO
Diagnóstico Médico	NÃO	NÃO	NÃO	NÃO	NÃO
Análise de Imagens	SIM	SIM	SIM	SEMI*	NÃO
Robô Manipulador	NÃO	NÃO	SIM	NÃO	NÃO
Controlador de Refin.	NÃO	NÃO	NÃO	NÃO	NÃO
Tutor Interativo Ling.	NÃO	NÃO	NÃO	NÃO	SIM

* Semi: quando o próprio ambiente não mudar com a passagem do tempo, mas o nível de desempenho do agente se altera.

Resolução de Problemas

- ♦ Já vimos o que é um problema. Vamos agora buscar mecanismos para representá-lo e resolvê-lo, utilizando as técnicas da IA, ou seja, usando e manipulando CONHECIMENTO
- ♦ O Estudo do Conhecimento
- ♦ Resolução de Problemas por Busca
- ♦ Representação de Conhecimento

Resolução de Problemas

- ♦ O Estudo do Conhecimento
 - Uma teoria em IA consiste na especificação do conhecimento necessário a uma entidade cognitiva.
 - O que é uma ENTIDADE COGNITIVA?
 - É o "mecanismo" inteligente que permite entre outras atividades: solução de problemas, uso de linguagem, tomada de decisões, percepção, etc...
 - Na abordagem da IA Simbólica, a simulação da capacidade cognitiva requer conhecimento declarativo (definição declarativa da função) e algum tipo de raciocínio. Além disso, a evolução dos estados de conhecimento de um agente pode ser descrita em forma de linguagem (lógica ou natural).

Resolução de Problemas

- ♦ O conhecimento é central para a tarefa inteligente e na IAS, para que esta tarefa ocorra são necessários:
 - Uma BASE DE CONHECIMENTOS
 - Um MOTOR DE INFERÊNCIA
- ♦ **Base de Conhecimentos:**
 - Contém a informação específica sobre o domínio e será tão complexa quanto for o domínio e a capacidade cognitiva a ser simulada.
- ♦ **Motor de Inferência:**
 - Mecanismo que manipula a Base de Conhecimentos e gera novas conhecimentos.

Métodos de Busca

- ♦ A maioria dos problemas interessantes de IA não dispõe de soluções algorítmicas. Porém:
 - São solucionáveis por seres humanos e, neste caso, sua solução está associada à "inteligência";
 - Formam classes de complexidade variável existindo desde pequenos problemas triviais (jogo da velha) até instâncias extremamente complexas (xadrez);
 - São problemas de conhecimento total, isto é, tudo o que é necessário para solucioná-los é conhecido, o que facilita sua formalização.
 - Suas soluções têm a forma de uma sequência de situações legais e as maneiras de passar de uma situação para outra são em número finito e conhecidas.
- ♦ Diante da falta de solução algorítmica viável, o único método de solução possível é a BUSCA.

Métodos de Busca

- ♦ Agentes de Resolução de Problemas
- ♦ Decidem o que fazer pela busca de ações que levem a estados desejáveis
 - estado inicial
 - operadores
 - teste de meta
 - função de custo de caminho
- ♦ Desempenho da Busca
 - Encontra uma solução?
 - É uma boa solução?
 - Custo do caminho
 - Qual o custo da busca?
 - Tempo e memória
 - $\text{Custo total} = \text{Custo da busca} + \text{custo do caminho}$

Métodos de Busca

- ♦ Exemplo: Jogo do Oito

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

- **Estados:** local de cada uma das peças e do espaço
- **Operadores:** mover o espaço para cima, para baixo, esquerda ou direita.
- **Teste de meta:** dado na figura
- **Custo do caminho:** 1 para cada movimento

Métodos de Busca

♦ Estratégias de Busca

- Critérios

- Completude
- Complexidade de Tempo
- Complexidade de Espaço
- Otimização

- Métodos

- Busca Cega - Não existe informação
- Busca Heurística - Faz uso de informação

Busca Cega

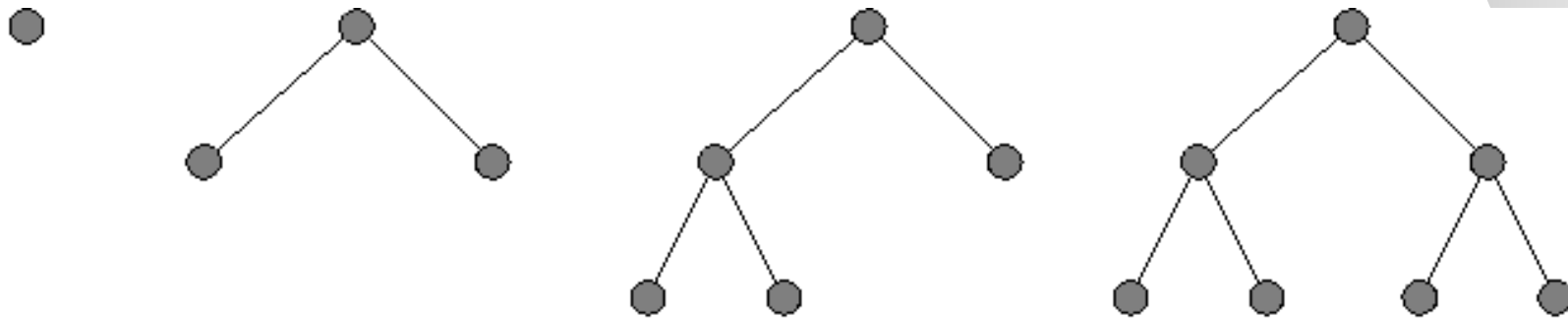
(Blind Search ou Uninformed Search)

- ♦ Uma estratégia de busca é dita cega se ela não leva em conta informações específicas sobre o problema a ser resolvido.
- ♦ Tipos de Busca Cega
 - Busca em largura
 - Busca pelo custo uniforme
 - Busca em profundidade
 - Busca em profundidade limitada
 - Busca por aprofundamento iterativo
 - Busca bidirecional

Busca Cega

Busca em Largura (Amplitude)

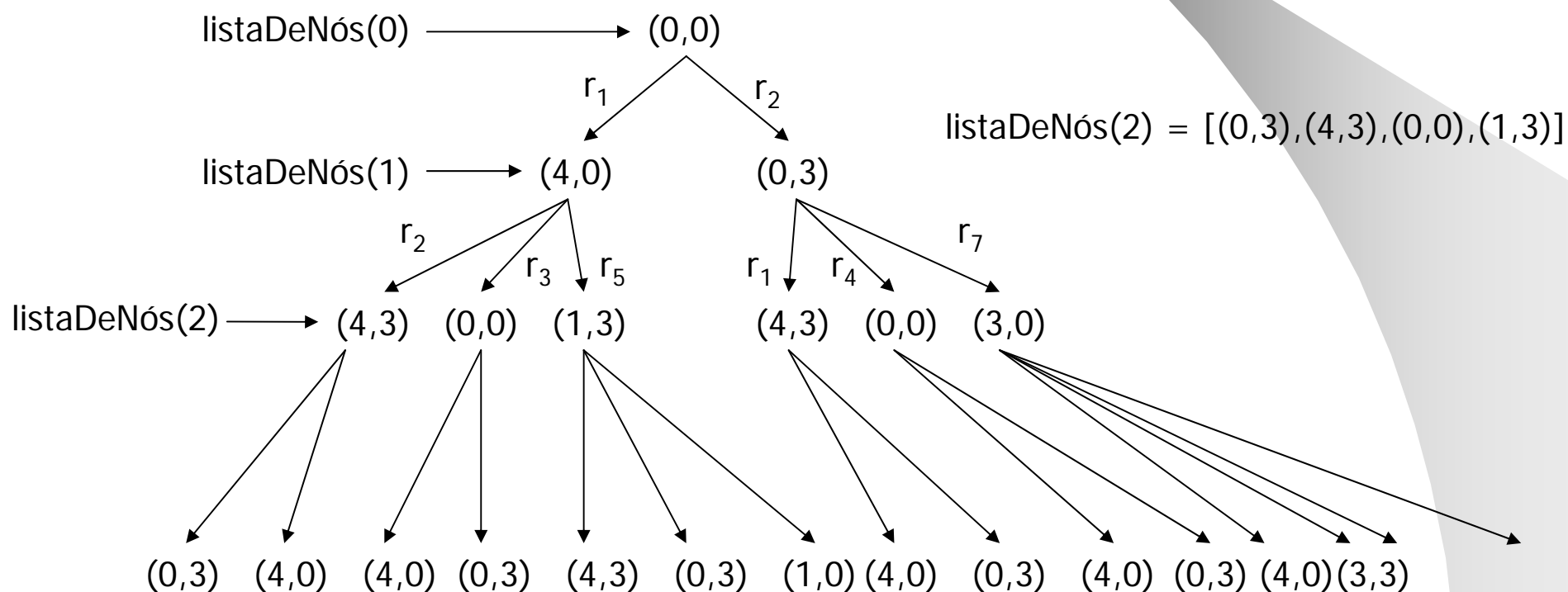
- Consiste em construir uma árvore de estados a partir do estado inicial, aplicando a cada momento, todas as regras possíveis aos estados do nível mais baixo, gerando todos os estados sucessores de cada um destes estados. Assim, cada nível da árvore é completamente construído antes de qualquer nodo do próximo nível seja adicionado à árvore



Busca Cega

Busca em Largura (Amplitude)

- ♦ Exemplo: Um balde de 4 litros e um balde de 3 litros. Inicialmente vazios.
 - Estado Final: um dos baldes com 2 litros de água.



Busca Cega

Busca em Largura (Amplitude)

- ♦ Características: Completa e Ótima
 - Se existe solução, esta será encontrada;
 - A solução encontrada primeiro será a de menor profundidade.
- ♦ Análise de Complexidade - Tempo e Memória
 - Seja um fator de ramificação b .
 - Nível 0: 1 nó
 - Nível 1: b nós
 - Nível 2: b^2 nós
 - Nível 3: b^3 nós
 - Nível d (solução) b^d nós

Busca Cega

Busca em Largura (Amplitude)

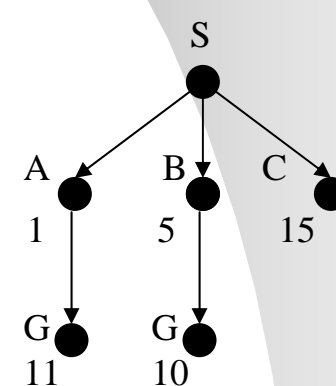
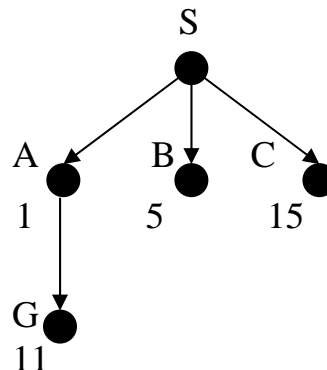
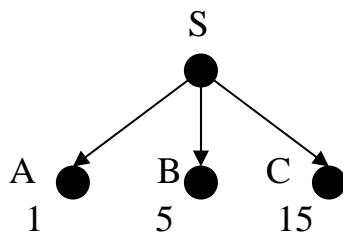
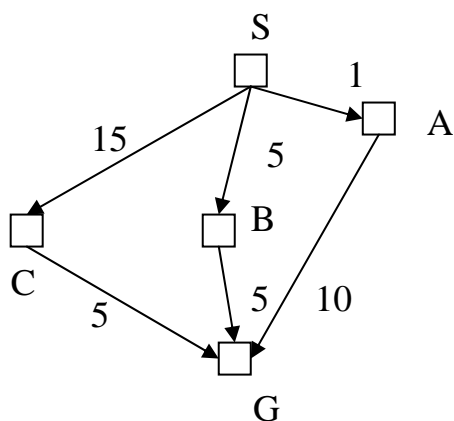
- ♦ Análise de Complexidade - Tempo e Memória

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Busca Cega

Método do Custo Uniforme

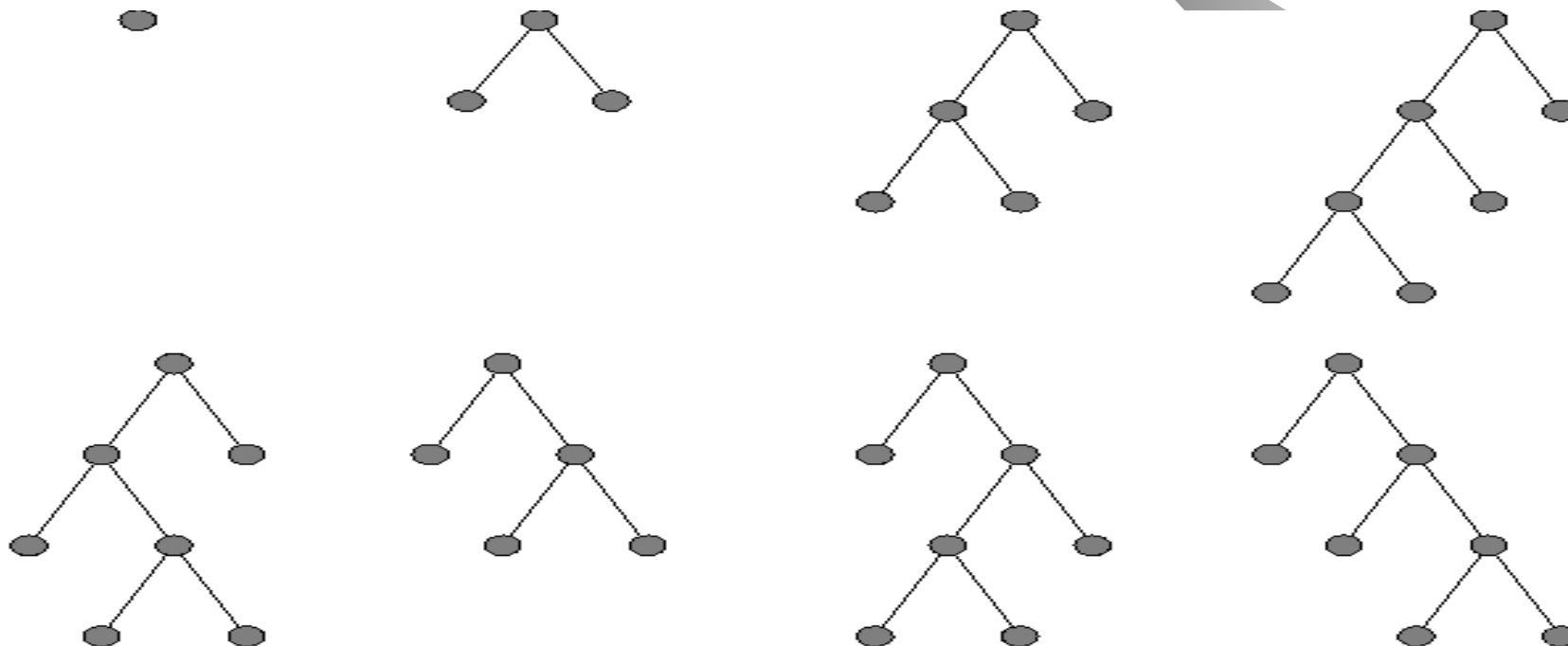
- ♦ Supondo que exista um "custo do caminho" associado a cada nó percorrido e que se deseje achar o caminho de custo mínimo.
- ♦ Neste caso, o algoritmo anterior é modificado para expandir primeiro o nó de menor custo.
- ♦ Exemplo: Problema de Rota entre S e G



Busca Cega

Busca em Profundidade

- ♦ Procurar explorar completamente cada ramo da árvore antes de tentar o ramo vizinho.



Busca Cega

Busca em Profundidade

- Exemplo: Um balde de 4 litros e um balde de 3 litros. Inicialmente vazios. Estado Final: um dos baldes com 2 litros de água.
- O que acontece quando nenhuma regra pode ser aplicada, ou a árvore atinge uma profundidade muito grande sem que tenha encontrado uma solução?
 - Neste caso ocorre o BACKTRACKING, ou seja, o algoritmo volta atrás e tenta outro caminho.
 - Considere o seguinte sistema de produção:
 $E = \{0, 1, 2, 3, 4, 5\}$
 $e_0 = 0$
 $F = \{3\}$
 $R = \{ r_1 = (x \mid x \geq 1 \text{ e } x \leq 2) \rightarrow (2 * x)$
 $r_2 = (x \mid \text{é Par}(x)) \rightarrow (x + 1) \}$

Busca Cega

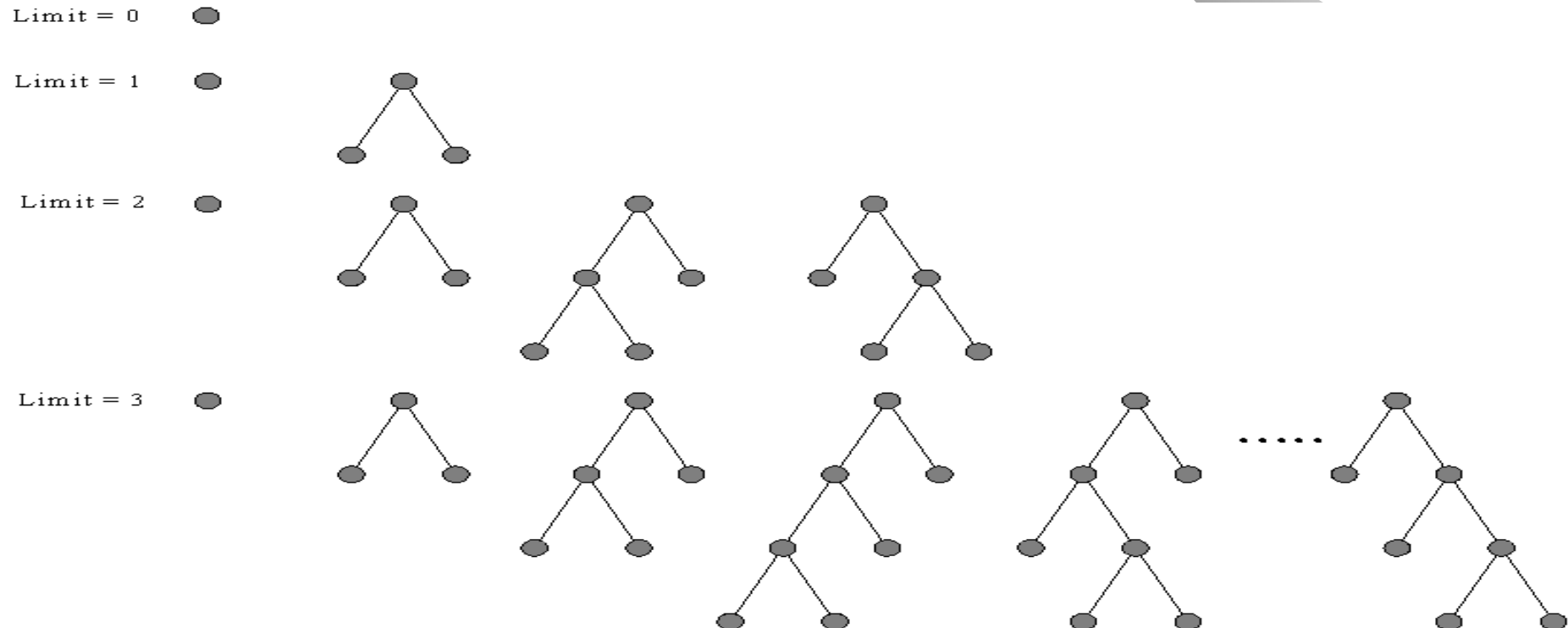
Busca em Profundidade

- ♦ Características: Não é Completa e Não é Ótima
 - Se admitir estados repetidos ou um nível máximo de profundidade, pode nunca encontrar a solução.
 - A solução encontrada primeiro poderá não ser a de menor profundidade.
 - O algoritmo não encontra necessariamente a solução mais próxima, mas pode ser **MAIS EFICIENTE** se o problema possui um grande número de soluções ou se a maioria dos caminhos pode levar a uma solução.
- ♦ Análise de Complexidade - Tempo e Memória
 - Seja m a profundidade máxima e um fator de ramificação b .
 - Tempo: b^m
 - Memória: $b.m$

Busca Cega

Busca por Aprofundamento Iterativo

- ♦ Teste de todos os possíveis limites com busca por profundidade limitada.
- ♦ Em geral é o melhor método quando o espaço de busca é grande e a profundidade é desconhecida.



Busca Cega

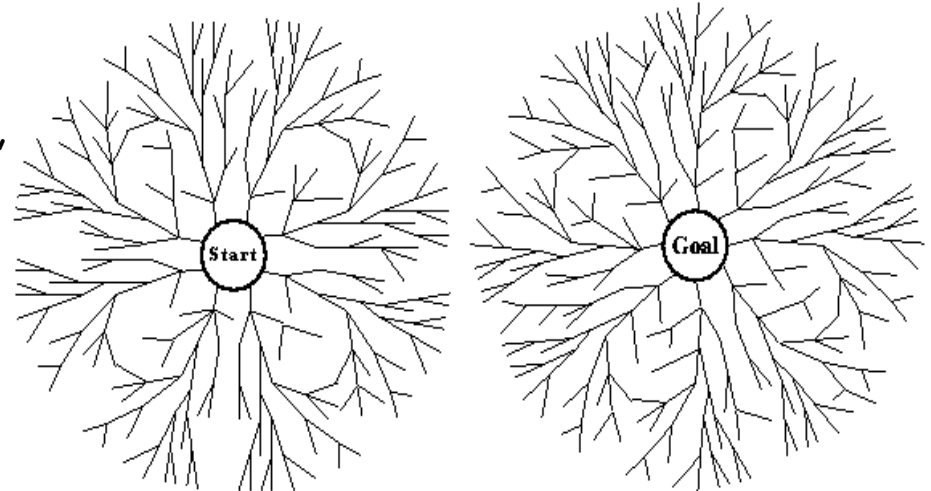
Busca Bidirecional

- A idéia deste método de busca é procurar simultaneamente “para a frente” a partir do estado inicial e “para trás” a partir do estado final, e parar quando as duas buscas se encontrarem no meio.
- Nem sempre isto é possível, para alguns problemas os operadores não são reversíveis, isto é, não existe a função predecessora e portanto não é possível fazer a busca “para trás”.

- **Análise de Complexidade**

- Comparando com a busca em largura, o tempo e o espaço para a busca é proporcional a $2b^{d/2}$, onde d é o nível onde está a solução e b é o fator de ramificação da árvore.

- Exemplo: Para $b=10$ e $d=6$, na busca em largura seriam gerados 1.111.111 nós, enquanto que na busca bidirecional seriam gerados 2.222 nós.



Comparação entre Métodos de Busca

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes*	No	No	Yes

b: fator de ramificação

d: profundidade da solução mais rasa

m: profundidade máxima da árvore de busca

l: limite de profundidade

Busca Heurística (Informed Search)

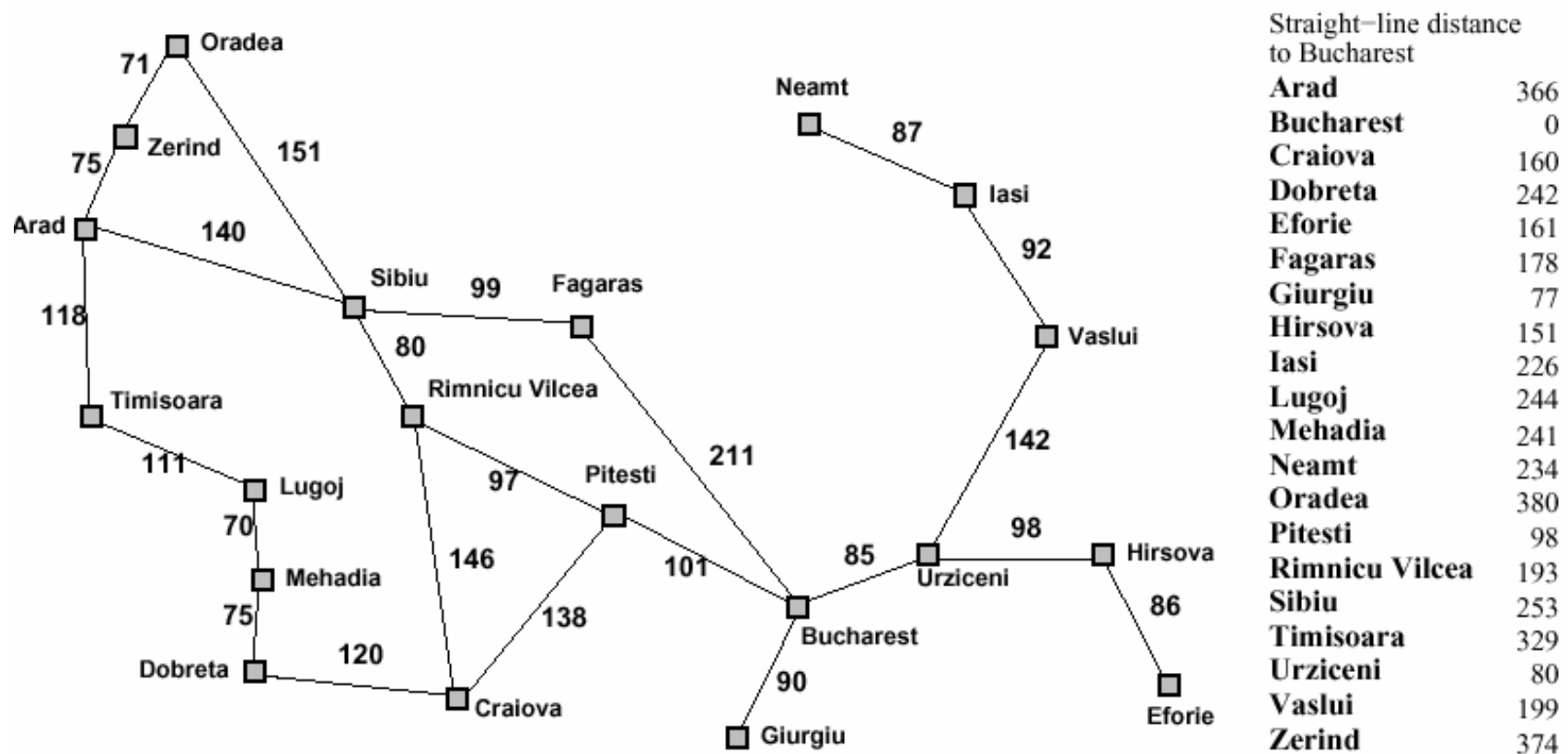
- ♦ Os métodos de busca vistos anteriormente fornecem uma solução para o problema de achar um caminho até um nó meta. Entretanto, em muitos casos, a utilização destes métodos é impraticável devido ao número muito elevado de nós a expandir antes de achar uma solução.
- ♦ Para muitos problemas, é possível estabelecer princípios ou regras práticas para ajudar a reduzir a busca.
- ♦ A técnica usada para melhorar a busca depende de informações especiais acerca do problema em questão.
- ♦ Chamamos a este tipo de informação de **INFORMAÇÃO HEURISTICA** e os procedimentos de busca que a utilizam de **MÉTODOS DE BUSCA HEURISTICA**.

Busca Heurística (Informed Search)

- ♦ A informação que pode compor uma informação heurística é o Custo do Caminho.
- ♦ O CUSTO DO CAMINHO pode ser composto pelo somatório de dois outros custos:
 1. O custo do caminho do estado inicial até o estado atual que está sendo expandido (função g); e
 2. Uma estimativa do custo do caminho do estado atual até o estado meta (função heurística h).
- ♦ A filosofia geral que move a busca heurística é: O MELHOR PRIMEIRO. Isto é, no processo de busca deve-se primeiro expandir o nó "mais desejável" segundo uma função de avaliação.

Busca Heurística (Informed Search)

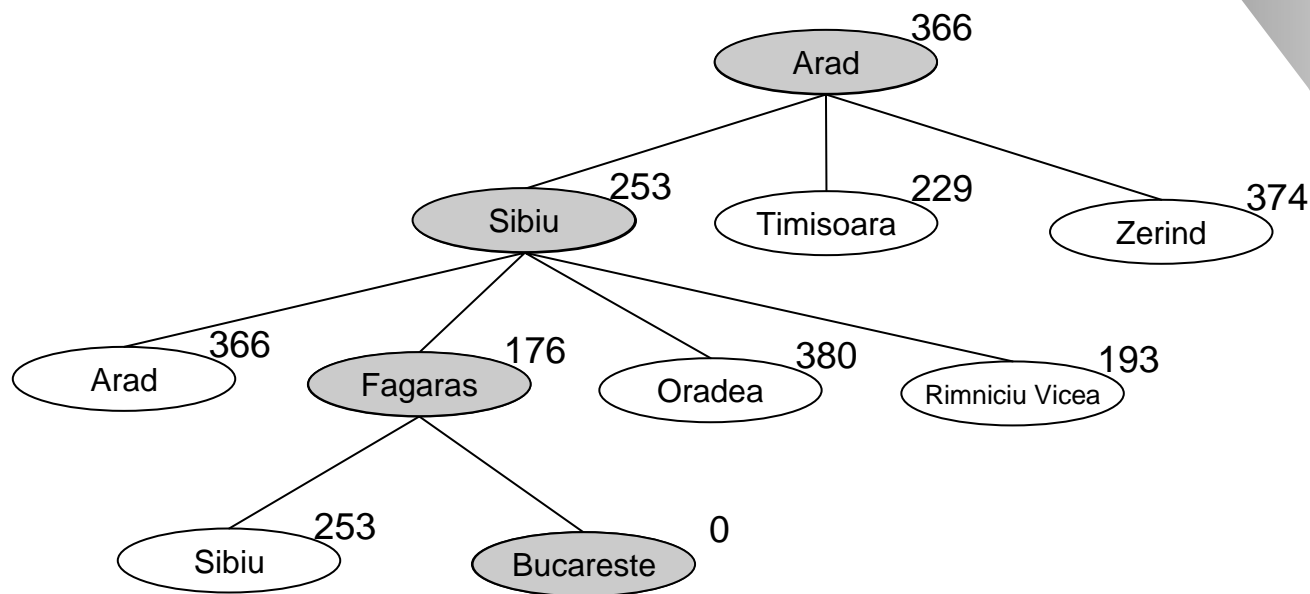
Romania with step costs in km



Busca Heurística

Busca Gulosa (Greedy Search)

- Semelhante à busca em profundidade com backtracking.
- Tenta expandir o nó que parece mais próximo ao nó meta com base na estimativa feita pela função heurística h .
- No caso do mapa da Romênia, $h(n)$ é a distância em linha reta de n até Bucareste.



Busca Gulosa (Greedy Search)

- ♦ **Análise de Complexidade**
 - É completa se não admitir estados repetidos;
 - Tempo: $O(b^m)$, mas uma boa heurística pode reduzir drasticamente o tempo;
 - Espaço: $O(b^m)$, todos os nós são mantidos na memória;
 - Não garante a solução ótima.

Busca Heurística

Busca A^* (A estrela)

- Filosofia: procurar evitar expandir nós que já são "custosos".
- É um método de busca que procura otimizar a solução, considerando todas as informações disponíveis até aquele instante, não apenas as da última expansão.
- Todos os estados abertos até determinado instante são candidatos à expansão.
- Combina, de certa forma, as vantagens tanto da busca em largura como em profundidade
- Busca onde o nó de menor custo "aparente" na fronteira do espaço de estados é expandido primeiro.
- $f(n) = g(n) + h(n)$ onde
 - $g(n)$ = custo do caminho do nó inicial até o nó n .
 - $h(n)$ = custo do caminho estimado do nó n até o nó final.
 - $f(n)$ = custo do caminho total estimado.

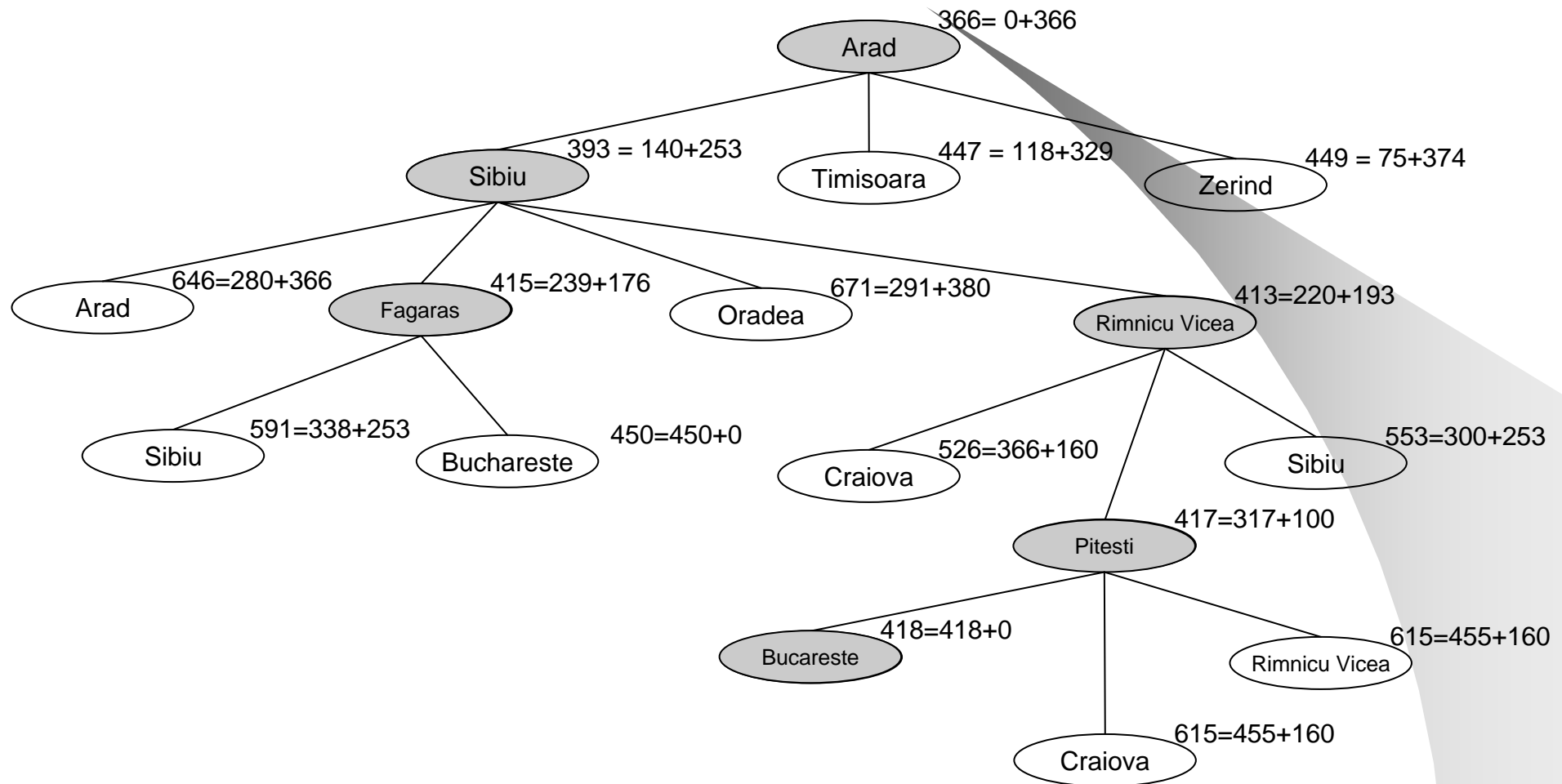
Busca Heurística

Busca A^* (A estrela)

- A^* expande o nó de menor valor de f a cada instante.
- A^* deve usar uma heurística admissível, isto é, $h(n) \leq h^*(n)$ onde $h^*(n)$ é o custo real para ir de n até o nó final.
- **Admissibilidade de A^***
 - Diz-se que um método de busca é **ADMISSÍVEL** se ele sempre encontra uma solução e se esta solução é a de menor custo.
 - A busca em largura é admissível. O mesmo não ocorre com a busca em profundidade.
- **Teorema da Admissibilidade de A^***
 - A busca A^* é ótima, isto é, sempre encontra o caminho de menor custo até a meta.

Busca Heurística

Busca A* (A estrela)



Busca Heurística

Busca A* (A estrela)

- ♦ Quanto mais admissível a heurística, menor o custo da busca.
- ♦ Exemplo: Para o jogo do oito
 - $h_1(n)$: número de peças fora do lugar
 - $h_2(n)$: distância Manhattan (número de casas longe da posição final em cada direção)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$$h_1(S) = ?? \quad 7$$

$$h_2(S) = ?? \quad 4+0+3+3+1+0+2+1 = 14$$

Busca Subida da Encosta (Hill climbing)

- É a estratégia mais simples e popular. Baseada na Busca em Profundidade.
- É um método de busca local que usa a idéia de que o objetivo deve ser atingido com o menor número de passos.
- A idéia heurística que lhe dá suporte é a de que o número de passos para atingir um objetivo é inversamente proporcional ao tamanho destes passos.
- Empregando uma ordenação total ou parcial do conjunto de estados, é possível dizer se um estado sucessor leva para mais perto ou para mais longe da solução. Assim o algoritmo de busca pode preferir explorar em primeiro lugar os estados que levam para mais perto da solução.

Busca Subida da Encosta (Hill climbing)

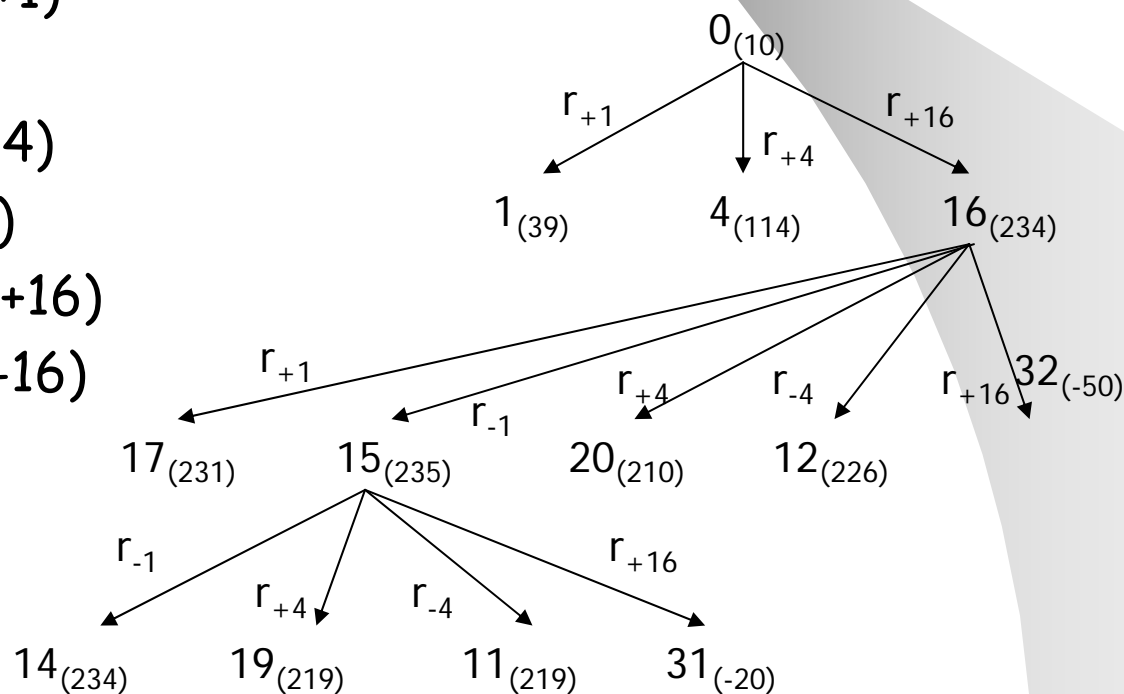
- ♦ Há duas variações do método:
 - **SUBIDA DE ENCOSTA SIMPLES:** Vai examinando os sucessores do estado atual e segue para o primeiro estado que for maior que o atual.
 - **SUBIDA DE ENCOSTA PELA TRILHA MAIS ÍNGREME:** Examina TODOS os sucessores do estado atual e escolhe entre estes sucessores qual é o que está mais próximo da solução.
- ♦ Este método não assegura que se atinja o ponto mais alto da montanha.
- ♦ Ele assegura somente que atingido um ponto mais alto do que seus vizinhos, então encontramos uma boa solução local.

Busca Subida da Encosta (Hill climbing)

♦ Exemplo

- Achar o ponto máximo da função $f(x) = -x^2 + 30x + 10$ no intervalo $[0, 100]$.

- $r+1 = (x | x < 100) \rightarrow (x+1)$
- $r-1 = (x | x > 0) \rightarrow (x-1)$
- $r+4 = (x | x < 97) \rightarrow (x+4)$
- $r-4 = (x | x > 3) \rightarrow (x-4)$
- $r+16 = (x | x < 85) \rightarrow (x+16)$
- $r-16 = (x | x > 15) \rightarrow (x-16)$

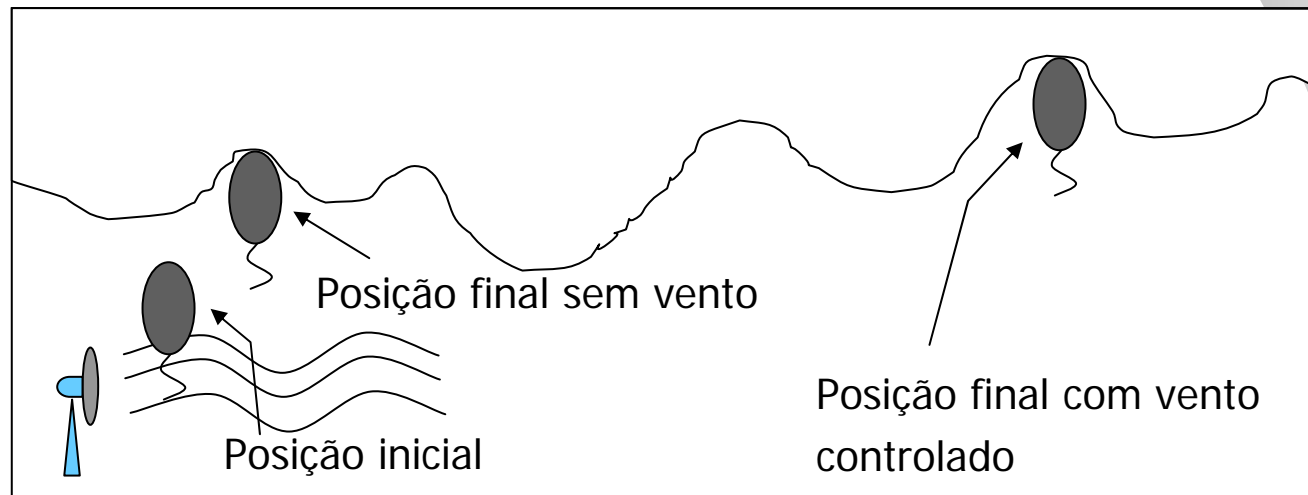


Busca por Têmpera Simulada (Simulated Annealing)

- É adequado a problemas nos quais a subida de encosta encontra muitos platôs e máximos locais.
- Não utiliza backtracking e Não garante que a solução encontrada seja a melhor possível.
- Pode ser utilizado em problemas NP-completos.
- É inspirado no processo de têmpera do aço.
Temperaturas são gradativamente abaixadas, até que a estrutura molecular se torne suficientemente uniforme.

Busca por Têmpera Simulada (Simulated Annealing)

- A idéia é permitir "maus movimentos" que com o tempo vão diminuindo de frequência e intensidade para poder escapar de máximos locais.
- O que o algoritmo de têmpera simulada faz é atribuir uma certa "energia" inicial ao processo de busca, permitindo que, além de subir encostas, o algoritmo seja capaz de descer encostas e percorrer platôs se a energia for suficiente.



Jogos

- ♦ Os jogos tem atraído a atenção da humanidade, às vezes de modo alarmante, desde a antiguidade.
- ♦ O que o torna atraente para a IA é que é uma abstração da competição (guerra), onde se idealizam mundos em que agentes agem para diminuir o ganho de outros agentes. Além disso, os estados de um jogo são facilmente representáveis (acessíveis) e a quantidade de ações dos agentes é normalmente pequena e bem definida.
- ♦ A presença de um oponente torna o problema de decisão mais complicado do que os problemas de busca, pois introduz incertezas, já que não sabemos como o oponente irá agir.
- ♦ Geralmente o oponente tentará, na medida do possível, fazer o movimento menos benéfico para o adversário.

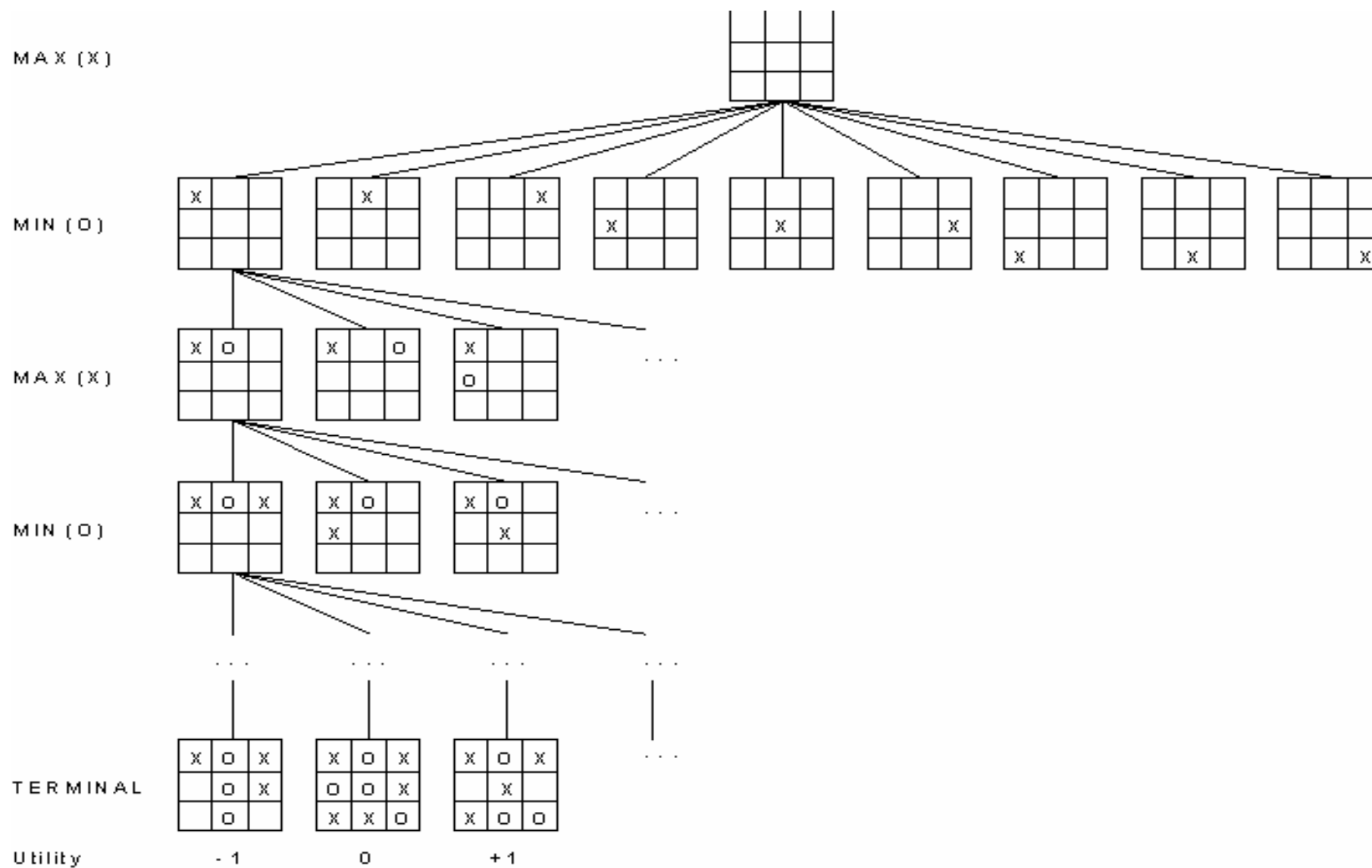
Jogos

- ♦ Jogos são, geralmente, problemas muito difíceis de resolver.
 - Xadrez difícil porque muito estados
 - Fator de ramificação 35
 - Geralmente 50 movimentos para cada jogador
 - 35^{100} estados ou nós
- ♦ Limites de tempo penalizam a ineficiência;
- ♦ Não é possível fazer a busca até o fim, de modo que devemos fazer o melhor possível baseados na experiência passada.
- ♦ Deste modo, jogos são muito mais parecidos com problemas do Mundo Real do que os problemas "Clássicos" vistos até agora.

Jogos

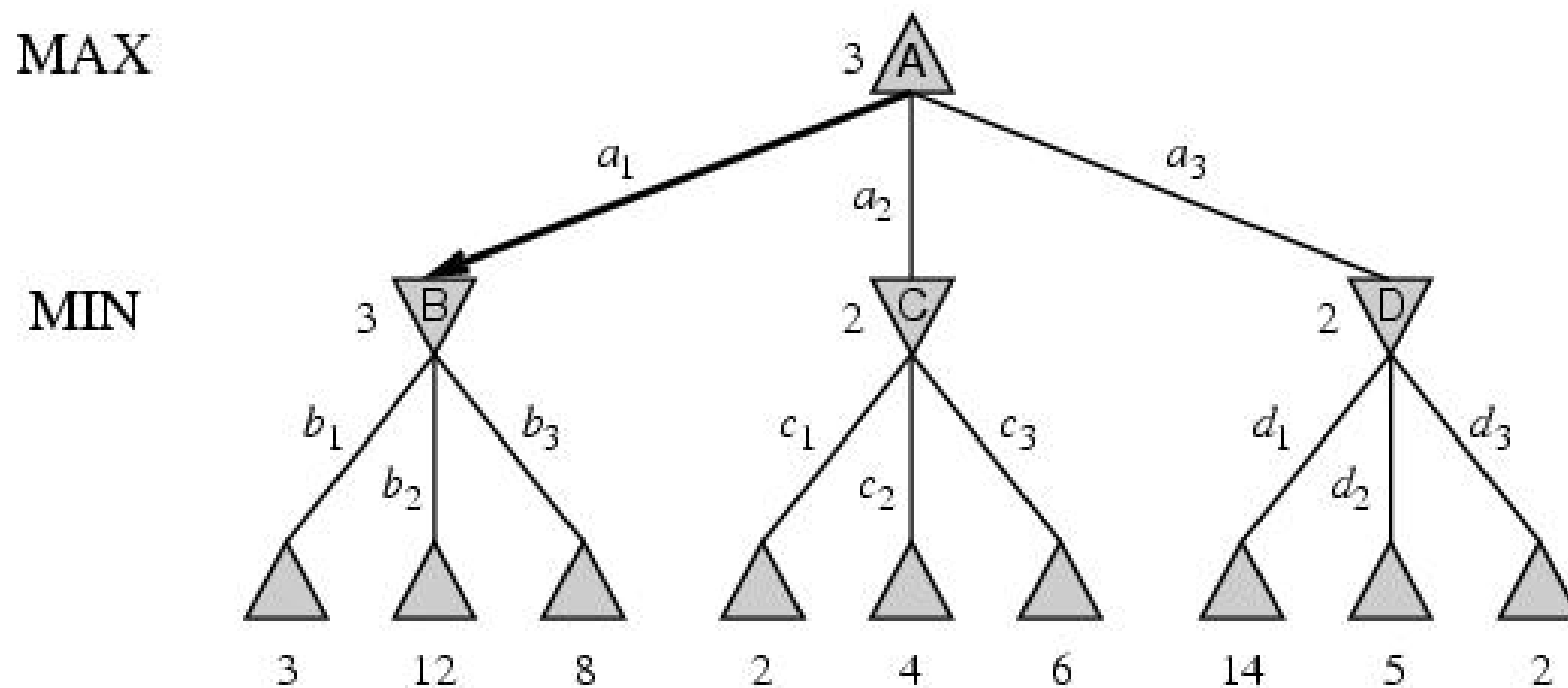
- ♦ Jogos de Duas Pessoas
 - Existem dois jogadores: MAX e MIN (MAX começa jogando).
- ♦ Jogos como um tipo de problema de busca:
 - O estado inicial;
 - Um conjunto de operadores;
 - Teste de Fim de Jogo (estados finais);
 - Função de Utilidade (payoff) - dá um resultado numérico para o resultado ou consequência de um jogo.
- ♦ Estratégia
 - Problemas de busca:
 - sequência de movimentos que levam a um estado meta
 - MIN NÃO DESEJA QUE MAX ganhe;
 - MAX achar estratégia que leve a vitória independentemente dos movimentos de MIN.

Jogos



Jogos

- 1º Exemplo: Algoritmo MINMAX



Jogos

- ♦ **1º. Exemplo: Algoritmo MINMAX**
 1. Gerar toda a árvore do jogo;
 2. Aplicar a função utilidade a cada nó terminal;
 3. Usar a utilidade dos nós terminais para determinar a utilidade dos nós um nível acima:
 - a) Quando acima é a vez de MIN fazer um movimento, escolher o que levaria para o retorno mínimo
 - b) Quando acima é a vez de MAX fazer um movimento, escolher o que levaria para o retorno máximo
 4. Continuar calculando os valores das folhas em direção ao nó raiz;
 5. Eventualmente é alcançado o nó raiz nesse ponto MAX escolhe o movimento que leva ao maior valor.

Jogos

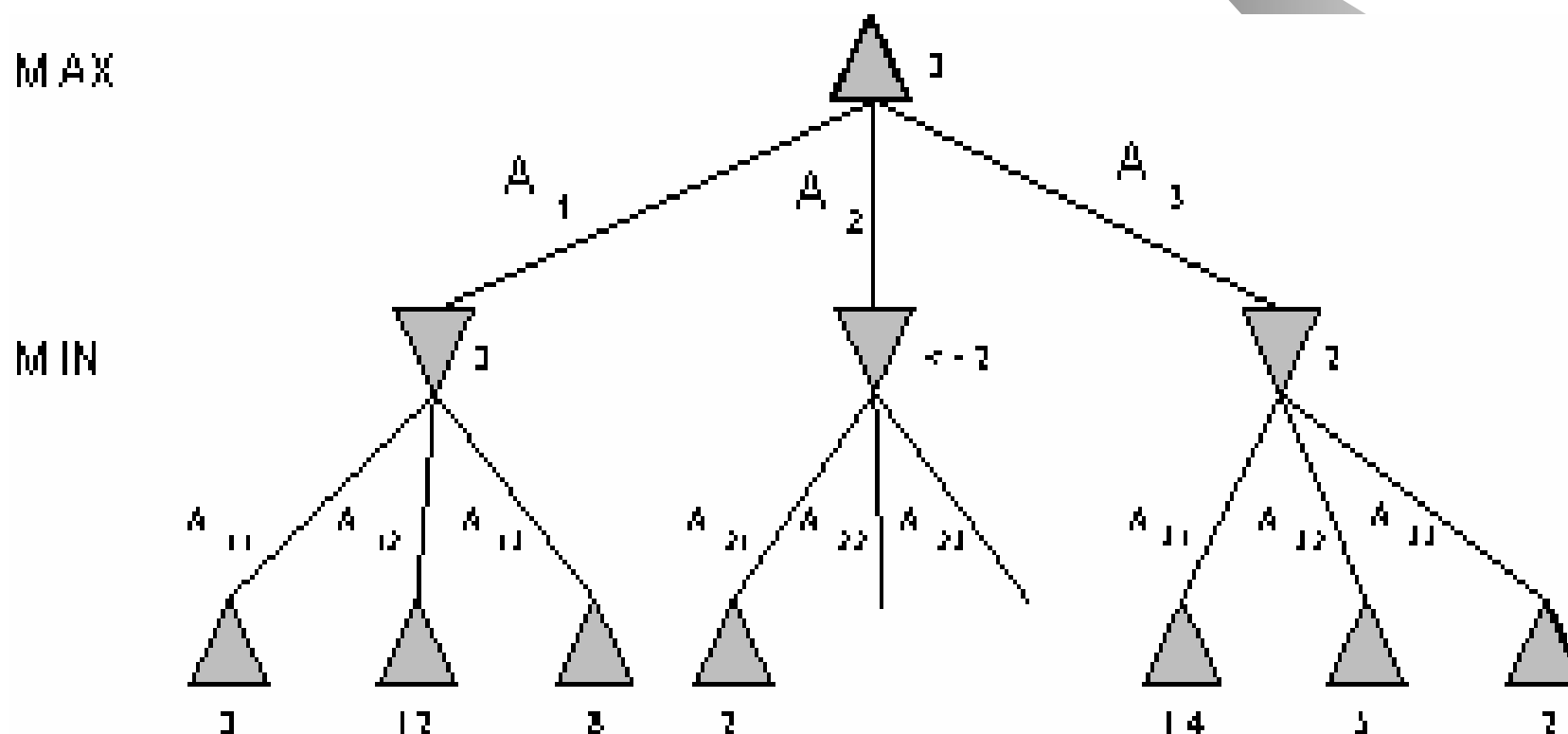
- ♦ Decisões Imperfeitas
 - Algoritmo MINIMAX deve procurar até alcançar um nó terminal.
 - Usualmente isso não é prático (complexidade $O(b^m)$)
 - Sugestão:
 - Parar a busca num tempo aceitável;
 - A função utilidade é substituída por uma função de avaliação heurística (EVAL), que retorna uma estimativa da utilidade esperada do jogo em uma dada posição;
 - Teste terminal por um teste de corte.

Jogos

- ♦ Função de Avaliação
 - Retorna uma estimativa da utilidade do jogo a partir de uma dada posição;
 - EX: xadrez:
<http://caissa.onenet.net/chess/texts/Shortcut>
 - Deve coincidir com a função de utilidade nos nós terminais;
 - Não deve ser difícil de calcular;
 - Deve refletir as chances reais de ganhar.

Jogos

- ♦ Poda Alfa-Beta
 - Processo de eliminar ramos da árvore sem examiná-los.
 - MINIMAX é em profundidade
- ♦ Eficiência depende da ordem em que os sucessores são examinados.



Representação de Conhecimento

- ♦ O Conhecimento e o Raciocínio são importantes para agentes artificiais, pois permitem comportamentos bem-sucedidos que seriam muito difíceis de alcançar de outra forma.
- ♦ Conhecimento pode ser definido como a informação armazenada, ou os modelos usados por pessoas ou máquinas para prever, interpretar e responder apropriadamente ao mundo exterior.
- ♦ A manipulação do conhecimento exige, antes, formas de representação. Esta representação deve ser suficientemente rica e completa para evitar falhas evidentes de entendimento pelo motor de inferência.
 - Exemplo:
 - Gato - é um: ser vivo + mamífero + raças + etc.
 - Logo: leão, rato, cão, etc é gato.

Representação de Conhecimento

- ♦ Para que uma representação lógica seja adequada, duas condições são necessárias:
 1. Existência de uma correspondência um para um entre certas classes de símbolos da representação e conjuntos de objetos de interesse no mundo externo;
 2. Existência, para cada relação simples no mundo externo, de uma relação na representação, de tal maneira que a relação entre dois símbolos da representação seja válida se, e somente se, a relação correspondente for válida entre os objetos correspondentes do mundo externo.
- ♦ Sem essas condições temos o **Conhecimento Incerto**.

Representação de Conhecimento

- ♦ É importante distinguir: FORMA e CONTEÚDO.
 - Exemplo: Um texto que usa a linguagem natural como recurso de representação pode ter, também seu conteúdo sintetizado através de outros recursos, como, por exemplo a lógica de predicados.
 - O conteúdo é o mesmo, as formas de representação diferentes facilitam a manipulação por diferentes agentes (computador, ser humano)

Representação de Conhecimento

Conceitos Básicos

- ♦ O processo de raciocínio é importante, pois permite a explicitação de uma solução adequada, para um problema em particular.
- ♦ Este processo dever ser capaz de gerar novos conhecimentos a partir de conhecimentos previamente armazenados (inferência).
- ♦ Se a informação não estiver explicitamente na base, a inferência é necessária. Existe um compromisso entre a quantidade de conhecimento armazenada explicitamente na base de conhecimento e a atividade de inferência.

$$\begin{array}{c} \text{Capacidade Cognitiva} \\ = \\ \text{base de conhecimento} + \text{motor de inferência} \end{array}$$

Representação de Conhecimento

Conceitos Básicos

- ♦ Inferência: o raciocínio formal é utilizado em geral, nas representações baseadas em lógica. Sendo possível três tipos:
 - **Inferência Dedutiva**: a partir de elementos de conhecimento representados em forma lógica, utiliza-se uma regra de inferência válida para inferir um novo elemento. Algumas regras de inferência utilizadas em lógica são: Modus Ponens, Modus Tollens, Silogismo Hipotético, etc.
 - **Inferência Abdutiva**: a partir de um conhecimento geral da forma $\forall x P(x) \rightarrow Q(x)$, e tendo por objetivo provar $Q(a)$, toma-se por hipótese que a razão pela qual $Q(a)$ se verifica é a validade de $P(a)$.
 - **Inferência Indutiva**: a partir de fatos experimentais que comprovam que a cada vez que a validade de $P(a)$ é verificada, verifica-se a validade de $Q(a)$ (mas não o contrário), para diferentes elementos a , pode-se inferir por indução que é válido.

Representação de Conhecimento

Conceitos Básicos

- ♦ Conhecimento: repositório de procedimentos, heurísticas, dados, etc., que compõe o conhecimento. Assemelha-se a um banco de dados no sentido de que exige manutenção (atualizações, inserções e deleções) mas o acesso a uma informação é mais elaborado.
- ♦ Engenheiro do Conhecimento: o profissional de ciência da computação responsável pela implantação da base de conhecimento. É um profissional com sólidos conhecimentos em técnicas de IA.
- ♦ Especialista do Domínio: é um profissional altamente capacitado no domínio para o qual estamos desenvolvendo a aplicação. Supre o engenheiro do conhecimento com os procedimentos (formais e heurísticas) necessários à construção da base de conhecimento.

Representação de Conhecimento

Características Essenciais

- ♦ Consistência: Não armazena informações conflitantes.
- ♦ Completude: Não apresenta lacunas no conhecimento armazenado. Todo o conhecimento necessário para a resolução do problema está explicitamente armazenado ou pode ser determinado via inferência.
- ♦ Coerência: Não existem ilhas isoladas de conhecimento não se relacionam com o restante do conhecimento armazenado.
- ♦ Redundância: A mesma unidade de conhecimento é armazenada de forma duplicada.

Representação de Conhecimento

Características Desejáveis

- Boas representações explicitam as coisas importantes.
- Revelam restrições naturais, facilitando algumas classes de computações.
- São concisas, necessitando apenas de recursos mínimos e sendo ao mesmo tempo ainda eficientes quando efetuam inferências.
- Podem ser rapidamente recuperadas e armazenadas.
- Informações raramente usadas são abordadas e recuperadas apenas quando necessárias.
- Permitem uma aquisição fácil e são legíveis pelo especialista, quando for o caso.
- Permitem a aplicação dos mecanismos de inferência necessários.

Representação de Conhecimento

- ♦ Principais Formas de Representação de Conhecimento
 - Sistemas de Produção
 - Redes Semânticas
 - Quadros (Frames) e Roteiros (Scripts)
 - Lógica

Sistemas ou Regras de Produção

- ♦ Concebidas por Emil Post (1943) quando demonstrou que um procedimento computável pode ser modelado como um sistema de produção.
- ♦ Muito utilizada nas décadas de 50 e 60. É o formalismo mais difundido de representação de conhecimento.
- ♦ Consiste em transformar o problema em um grafo de estados. Este grafo deve possuir um estado inicial e deve-se ter uma forma de identificar um estado final quando algum for atingido.
- ♦ Ou seja, consiste em:
 - Regras de Produção + Memória de Trabalho + ciclo de controle (tipo reconhece-atua)

Sistemas de Produção

- Um Sistema de Produção é definido como uma tupla $SP = \langle R, E, e_0, F \rangle$, onde R é um conjunto de regras, E é um conjunto de estados, e_0 é o estado inicial e F é o conjunto de estados finais.
- Uma Regra de Produção é constituída por um par $\langle p, f \rangle$, onde $p: E \rightarrow \{V, F\}$ e $f: E \rightarrow E$. O elemento p é o padrão da regra, e f constitui a operação. Gera normalmente estruturas do tipo:
- SE <estado> ENTÃO <ação>
onde:
 - <estado> ou <condição>: estabelece um teste cujo resultado depende do estado atual da base de conhecimento. Tipicamente o teste verifica a presença ou não de certas informações na base.
 - <ação>: altera o estado atual da base de conhecimento, adicionando, modificando ou removendo unidades de conhecimento presentes na base. Pode acarretar também efeitos externos à base, como por exemplo a escrita de uma mensagem no vídeo.

Sistemas de Produção

- ♦ EXEMPLO: Problema dos Dois Baldes de Água
 - Enunciado: Você recebe dois baldes de água, um de quatro litros e outro de três litros. Nenhum deles possui qualquer marcação de medida. Há uma torneira que pode ser utilizada para encher os baldes de água. Como colocar exatamente dois litros d'água dentro do balde de quatro litros?
 - Conjunto de Estados: O espaço de estados para este problema pode ser modelado como o conjunto de pares ordenados de números naturais (x,y) tal que $x = 0, 1, 2, 3$ ou 4 e $y = 0, 1, 2$ ou 3 , onde x representa a quantidade de água no balde de 4 litros, e y representa a quantidade de água no balde de 3 litros.
 - Estado Inicial: Ambos os baldes estão vazios: $(0,0)$.
 - Estado Final: Constituído por todos os estados onde a qtde de água no primeiro balde é 2, ou seja: $(2,n)$, onde $n = 0, 1, 2$ ou 3 .

Sistemas de Produção

- ♦ EXEMPLO: Problema dos Dois Baldes de Água
 - Um possível conjunto de regras para este problema seria:
 - r_1 $(x,y | x < 4) \rightarrow (4,y)$ Encher o balde de 4 litros
 - r_2 $(x,y | y < 3) \rightarrow (x,3)$ Encher o balde de 3 litros
 - r_3 $(x,y | x > 0) \rightarrow (0,y)$ Esvaziar o balde de 4 litros no chão
 - r_4 $(x,y | y > 0) \rightarrow (x,0)$ Esvaziar o balde de 3 litros no chão
 - r_5 $(x,y | x+y > 4) \rightarrow (4, y-(4-x))$ Despejar água do balde de 3 litros dentro do balde de 4 litros até que este esteja cheio
 - r_6 $(x,y | x+y > 3) \rightarrow (x-(3-y), 3)$ Despejar água do balde de 4 litros dentro do balde de 3 litros até que este esteja cheio
 - r_7 $(x,y | x+y \leq 4 \text{ e } y > 0) \rightarrow (x+y, 0)$ Despejar toda a água do balde de 3 litros dentro do balde de 4 litros
 - r_8 $(x,y | x+y \leq 3 \text{ e } x > 0) \rightarrow (0, x+y)$ Despejar toda a água do balde de 4 litros dentro do balde de três litros

Sistemas de Produção

- ♦ EXEMPLO: Problema dos Dois Baldes de Água
 - Uma solução possível para o problema seria aplicar em seqüência as regras r2, r9, r2, r7, r5 e r9 (r2, r5, r2, r5, r3 e r5).
 - Esta solução não é a única POSSÍVEL. Além disso, não foi mostrado como a solução foi encontrada.
 - Este é exatamente o ponto onde entram os ALGORITMOS DE BUSCA no Espaço de Estados.
- ♦ A modelagem de um problema como um sistema de produção consiste apenas em definir o espaço de estados e as regras. Este processo dificilmente poderia ser feito automaticamente. Como na programação tradicional, trata-se de um processo de modelagem de uma realidade perceptível utilizando uma ferramenta definida. Mas, UMA VEZ ESTABELECIDO O MODELO, O PROCESSO PODE SER LIBERADO PARA A MÁQUINA E ESTA ENCONTRAR SOZINHA A SOLUÇÃO.

Sistemas de Produção

- ♦ Outros Exemplos:

1. Problema dos Missionários e Canibais

- O estado inicial é $(3,3,0,0,0)$ e o único estado final é $(0,0,3,3,1)$.
- As regras são todas de movimentação de no máximo 2 pessoas de uma margem a outra do rio. Assim, pode-se movimentar dois canibais, dois missionários, um canibal e um missionário, apenas um canibal ou apenas um missionário, tanto da margem original para a margem oposta quanto vice-versa.
- Tem-se assim 10 regras possíveis, das quais uma é mostrada a seguir:
$$R_{MM} \rightarrow \text{Se}(m_1, c_1, m_2, c_2, 0 | (m_1 \geq 2) \text{ e } [m_1 - 2 \geq c_1] \text{ ou } (m_1 - 2 = 0)) \text{ e } (m_2 + 2 \geq c_2)) \text{ Então } (m_1 - 2, c_1, m_2 + 2, c_2, 1)$$
- Exercício: Ache as outras regras.

Sistemas de Produção

2. O problema das Três Jarras

- Há três jarras de vinho com capacidade para oito, cinco e três litros. A jarra maior maior está cheia de vinho e as outras estão completamente vazias. Queremos dividir o vinho em porções iguais entre as duas primeiras jarras de modo que cada uma fique com 4 litros.
- Exercício: Ache as regras deste sistema de produção.

3. MYCIN

- IF
 1. A infecção é principalmente por bactérias, e
 2. O local da cultura é um dos locais esterelizados, e
 3. O local suspeito de entrada do organismo é p trato gastro-intestinal
- Then
 - Existe uma evidência sugestiva (0.7) de bacteróide.

Sistemas de Produção

4. DENDRAL

- IF

- O espectro para as moléculas apresenta dois picos de massas $X1$ e $X2$, tal que:
 1. $X1+X2=M+28$, e
 2. $X1-28$ é um pico alto, e
 3. $X2-28$ -e um pico alto, e
 4. Pelo menos $X1$ ou $X2$ são altos,

- Then

- A molécula contém um grupo cetona.

Sistemas de Produção

5. JOGO DA PILHA DE PALITOS

```
; RULE good-human-move
; IF
;   There is a pile of sticks, and
;   The human has chosen how many
;     sticks to take, and
;   It is the human's move, and
;   The human's choice is valid
; THEN
;   Remove unneeded information, and
;   Compute the new pile size, and
;   Update the pile size, and
;   Print the number of sticks left in
;     the stack, and
;   Trigger the computer player's turn
```

```
RULE computer-move
; IF
;   It is the computers's move, and
;   The pile size is greater than 1, and
;   The computer's response is available
; THEN
;   Remove unneeded information, and
;   Compute the new pile size, and
;   Print the number of sticks left in the
;     stack, and
;   Update the pile size, and
;   Trigger the human players move
```

Sistemas de Produção

- ♦ VANTAGENS

- Modularidade: podem ser considerados como peças independentes. Novas regras podem ser acrescentadas ao conjunto já existente sem maiores preocupações.
- Naturalidade: pode ser considerada uma forma natural de pensar a descrição de conhecimentos.
- Uniformidade: todas as regras são escritas seguindo o mesmo padrão. Permite que pessoas não familiarizadas com o sistema possam também analisar seu conteúdo.

Sistemas de Produção

♦ DESVANTAGENS

- Opacidade: é difícil verificar a completeza destes sistemas, bem como verificar os possíveis fluxos de processamento.
- Ineficiência: resulta particularmente do número de regras a combinar e também do esforço do matching necessário ao suporte de execução das regras. (matching entende-se como a verificação das regras que se aplicam ao estado do problema)
- Não raciocinam em vários níveis.
- Não sabem como e quando violam suas próprias regras.
- Não tem acesso ao raciocínio que está por trás das regras.

Representação de Conhecimento

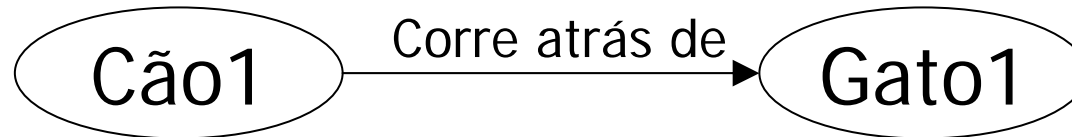
Redes Semânticas

- ♦ Consiste em um conjunto de nodos conectados por um conjunto de arcos.
 - Os nodos, em geral, representam objetos.
 - Os arcos representam relações binárias entre os objetos.
- ♦ Originalmente foram usadas para suporte a linguagem natural. Em 1968 Ross Quillian as usou para representar modelos psicológicos de memória humana chamado memórias semânticas.
- ♦ Quillian desenvolveu um programa que define palavras em inglês de forma similar a dicionários.
 - Em vez de definir palavras formalmente, cada definição simplesmente conduz a outras definições em uma forma desestruturada e, possivelmente circular.
 - Ao procurar uma palavra, percorremos a "rede" até que estejamos satisfeitos com o que compreendemos da palavra original.

Redes Semânticas

♦ Redes Semânticas Elementares

- Usa-se nodos para representar substantivos, adjetivos, pronomes e nomes próprios.
- Os arcos são reservados basicamente para representar verbos transitivos e preposições.
- Exemplo: "O cão corre atrás do gato"

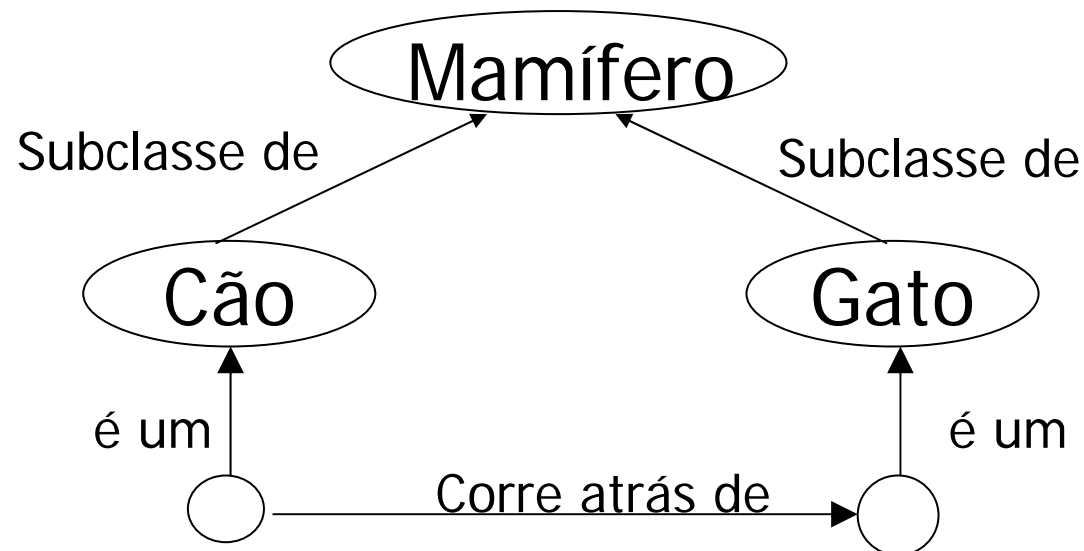


- Pode-se generalizar também a relação entre eles, representando os indivíduos específicos com nodos anônimos e indicando a inclusão destes indivíduos em sua respectiva classe, através da relação "é-um".

Redes Semânticas

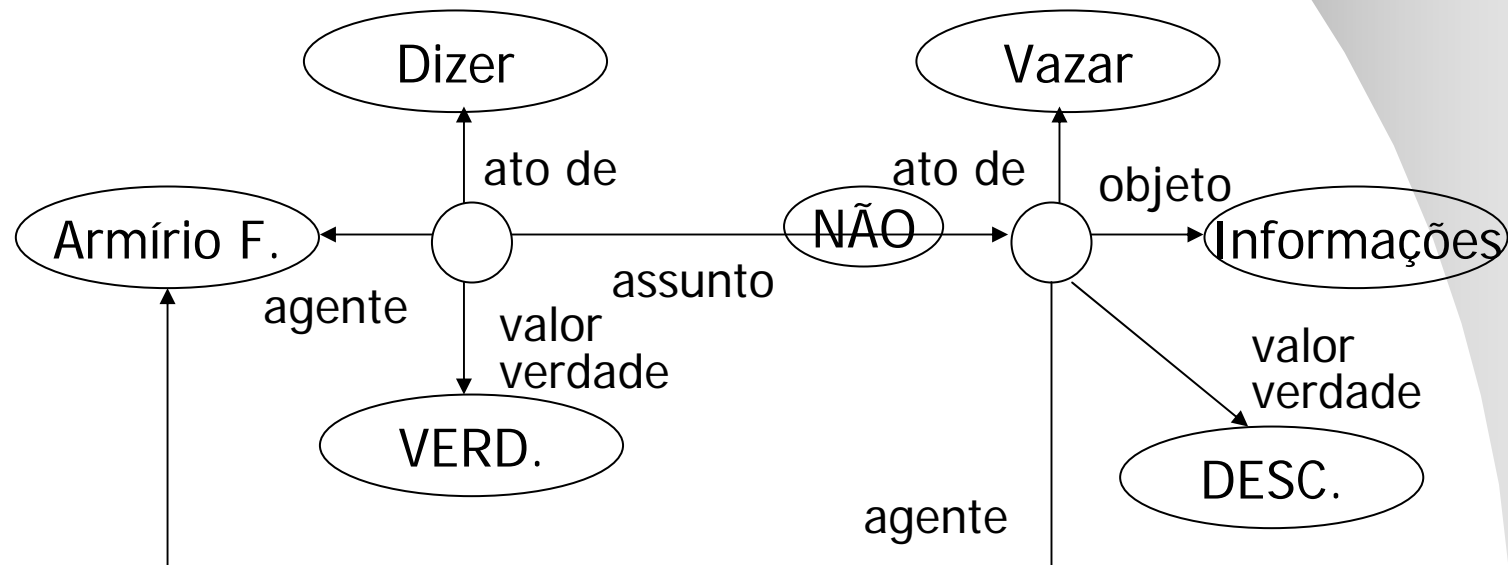
♦ Redes Semânticas Elementares

- Relações de inclusão entre classes são representadas por relações "subclasse-de".
- Os nodos rotulados representam classes genéricas, enquanto que os nodos anônimos representam indivíduos específicos.
- Para saber se um nodo representa uma instância, é só observar se ele está na origem de algum arco do tipo "é-um".



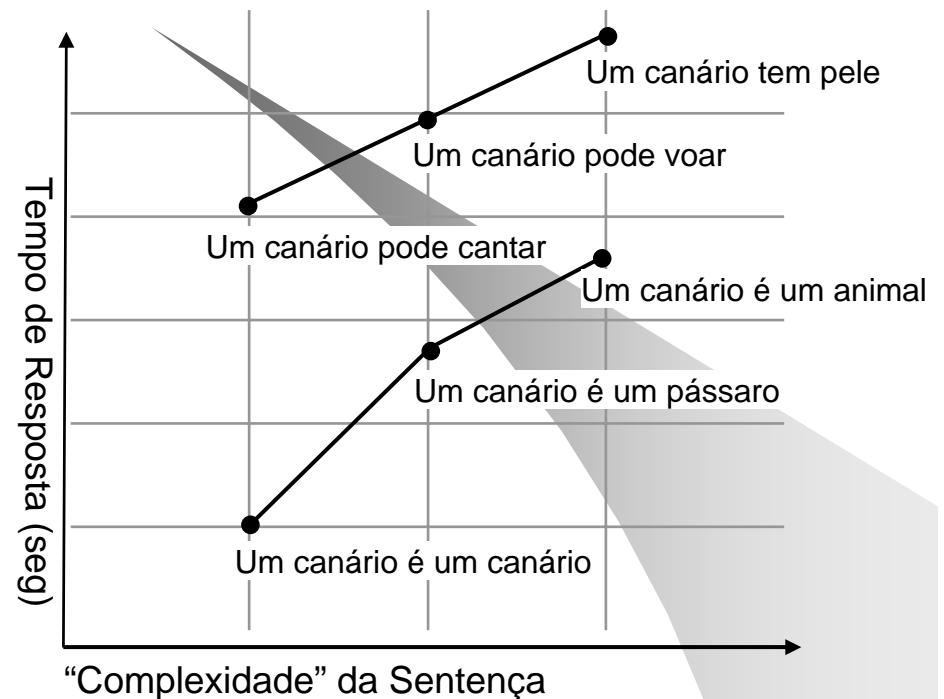
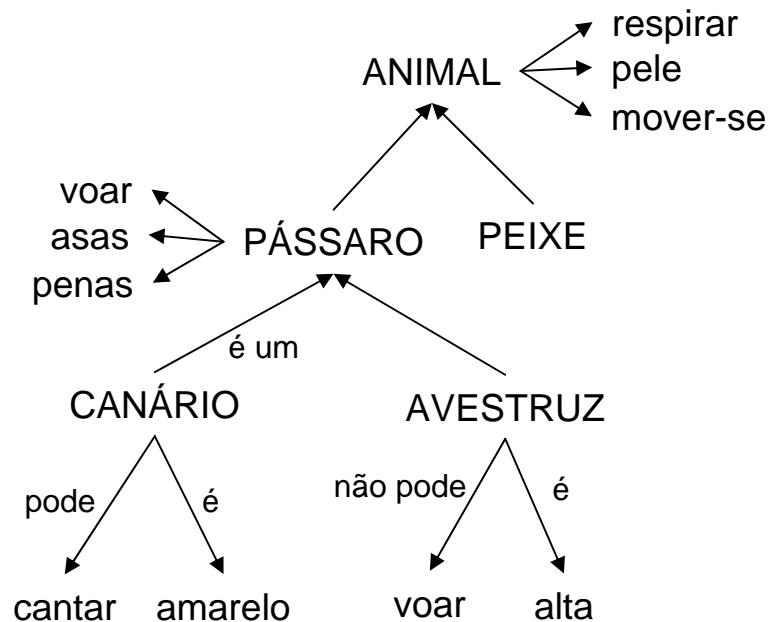
Redes Semânticas

- ♦ Redes Semânticas com Valores de Verdade
 - Pode ser necessário representar fatos dos quais não se conhece o valor de verdade, ou mesmo, fatos que sabemos serem falsos. Uma maneira de fazê-lo é rotulando cada nodo-predicado com um valor VERDADEIRO, FALSO ou DESCONHECIDO.
 - “Armírio Fraga disse que não vazou informações privilegiadas.”



Redes Semânticas

Collins e Quillian (1969)



Além da habilidade de associar conceitos, os humanos também organizam hierarquicamente o seu conhecimento, de forma que a informação seja mantida em níveis apropriados mais altos da taxonomia.

Quadros (Frames) e Roteiros (Scripts)

- Os Quadros ou Cenários ("Frames"), e sua variação, os roteiros ("Scripts"), foram introduzidos para permitir a expressão das estruturas internas dos objetos, mantendo a possibilidade de representar herança de propriedades.
- As pessoas, ao enfrentarem uma nova situação, guardam o repertório do comportamento para situações similares. Alguém que já assistiu alguma vez a um júri popular sabe que tipo de "quadro" irá encontrar se for a outro. (juiz, auxiliar de justiça, réu, advogado de defesa, promotor, etc.)
- Idéias fundamentais introduzidas por Marvin Minsky em 1975 ("A framework to represent knowledge").
- Está na origem das idéias que levaram às linguagens de programação orientadas a objetos.

Quadros (Frames)

- ♦ **Minsky (1975):** "Quando alguém encontra uma nova situação (ou modifica substancialmente o seu entendimento sobre um problema), recupera da memória uma estrutura chamada 'frame'. Esta estrutura é um arcabouço memorizado que deve ser adaptado para se adequar à realidade, alterando detalhes, conforme a necessidade"
- ♦ Um quadro consiste em um conjunto de atributos ("slots") que através de seus valores, descrevem as características do objeto representado pelo quadro.
- ♦ Os valores atribuídos aos atributos podem ser, além dos valores do objeto em particular, valores default, ponteiros para outros quadros, e conjuntos de regras de procedimento que podem ser implementados.
- ♦ Se os valores dos atributos forem apontadores para outros quadros, cria-se uma rede de dependências entre os quadros.
- ♦ Os conjuntos de procedimentos indicam que procedimento deve ser executado quando certas condições forem satisfeitas, por exemplo: ao ser criado o atributo, ao ser lido o valor do atributo, ao ser modificado o valor do atributo, ou ao ser destruído o valor do atributo.

Quadros (Frames)

- Os quadros também são organizados em uma hierarquia de especialização, criando outra dimensão de dependência entre eles (herança). Permite a especificação de propriedades de uma classe de objetos através da declaração de que esta classe é uma subclasse de outra que goza da propriedade em questão.
- O processo de herança e instanciação favorece a reutilização de código evitando definições repetitivas e aproveitando funções de acesso definidas para as facetas se-lido, se-escrito, se-necessário, etc
- Deve-se notar que as estruturas de quadros são ativas, pois sua manipulação causa o disparo automático das facetas.
- São úteis para domínio de problemas onde a forma e o conteúdo do dado desempenham um papel importante na solução do problema.

Quadros (Frames)

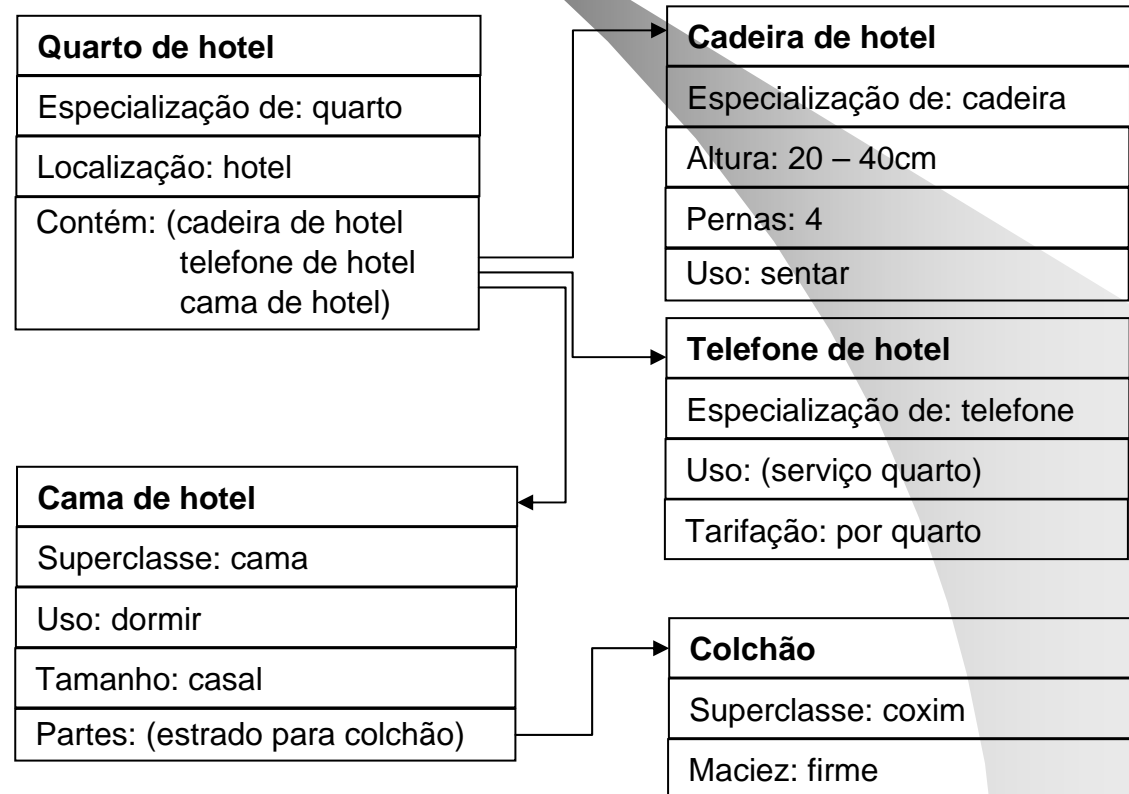
- ♦ Exemplo
 - Quadro: Cadeira
 - Slot: número de pernas - inteiro (default: 4)
 - Slot: tipo-de-encosto - curvo, reto, não-tem (default: curvo)
 - Slot: tipo-de-assento - redondo, anatômico, reto (default: anatômico)
 - Slot: número-de-braços - 2,1,0 (default: 0)
 - Slot: cor - preta, branca, incolor, azul (default: incolor)
 - Quadro: Cadeira-do-Renato
É-UM Cadeira
 - Slot: número de pernas - 4
 - Slot: tipo-de-encosto - (default: curvo)
 - Slot: tipo-de-assento - redondo
 - Slot: número-de-braços - 0
 - Slot: cor - (default: incolor)

Quadros (Frames)

- Parte de uma descrição por frame de um quarto de hotel.
- Cada frame individual pode ser visto como uma estrutura de dados.

Slots do frame contém:

- Identificação frame
- Relação com outros frames
- Descritores de requisitos
(altura do acento)
- Informação sobre uso
- Informação default
(cadeira tem 4 pernas)



Quadros (Frames)

- ♦ Quadros superam o poder das redes semânticas pois permitem que objetos complexos sejam representados como um único frame, em vez de uma grande estrutura de rede.
- ♦ Os frames tornam mais fácil organizar o conhecimento hierarquicamente.

Roteiros (Scripts)

- ♦ Originalmente concebido por Schank e seu grupo de pesquisa (1977).
- ♦ Pesquisa em roteiros examina a organização do conhecimento na memória e o papel que esta organização desempenha no raciocínio.
- ♦ São estruturas semelhantes aos quadros que descrevem seqüências estereotipadas de eventos em contextos particulares.
- ♦ A principal diferença que se pode estabelecer em relação aos quadros fica a nível das estruturas adotadas.
- ♦ Os eventos descritos formam uma cadeia causal.
- ♦ O início da cadeia é o conjunto de condições de entrada que permite a possibilidade de ocorrência do primeiro evento do roteiro.
- ♦ O fim da cadeia é o conjunto de resultados que permitirá a ocorrência de eventos posteriores.
- ♦ O raciocínio com roteiros serve especialmente para verificar se determinado evento ocorreu e também para verificar a relação entre os eventos, o que pode ser conseguido pelo exame da cadeia causal.

Roteiros (Scripts)

Exemplo

Roteiro: comer-em-restaurante

Apoio: (restaurante, dinheiro, alimento, menu, mesas, cadeiras)

Funções: (pessoas com fome, encontro de pessoas)

Ponto-de-vista: (pessoas com fome)

Tempo-de-ocorrência: (tempo-de-operação do restaurante)

Lugar-de-ocorrência: (localização do restaurante)

Seqüência-de-eventos

primeiro : Inicie o roteiro entrada-no-restaurante

então: Se (há-convite-para-sentar ou reservas) então siga roteiro orientação-do-garçom

então: siga roteiro aguarde-sentado

então: siga roteiro solicite-comida

então: siga roteiro comer, a menos que haja uma longa espera, caso em que seguirá o roteiro sai-do-restaurante-furioso.

- ♦ Outros Roteiros: refeição rápida (fast-food).

Representação de Conhecimento

Lógica

- ♦ Estudo das regras do raciocínio válido.
- ♦ Pode ser usada para representar conhecimento.
- ♦ O formalismo lógico parece atraente, pois, recorrendo-se à dedução matemática somos capazes de derivar novos conhecimentos a partir de outros já existentes.
- ♦ **Lógica das Proposições**
 - Proposições são afirmações que admitem um valor lógico, "verdadeiro" ou "falso".
 - Seja, por exemplo, uma fbf do cálculo proposicional:
 - `cor(gato,preto).`
 - Pode ter valor verdadeiro ou falso dependendo se o gato em questão é ou não preto.
 - Na representação do conhecimento, ela representa um fato e é suposta verdadeira no mundo que representa.

Lógica

♦ Lógica dos Predicados

- A capacidade de representação da lógica das proposições é pequena, a lógica dos predicados apresenta uma capacidade bastante ampliada neste sentido.
- A lógica dos predicados inclui funções, variáveis, quantificadores e predicados.
- É indecidível, ou seja, existem procedimentos que encontrarão a prova de um teorema proposto, se de fato houver o teorema, mas não há a garantia de parar se a afirmação proposta não for um teorema.
- Pode também ser usada para representar conhecimento. Seja o exemplo: $\forall(x,y,z)(\text{filho}(x,y) \wedge (\text{filho}(y,z) \rightarrow \text{neto}(x,z)))$
- Esta fbf encerra o que se pode chamar de regra.
- Esta regra é suposta verdadeira no mundo considerado.
- Pode-se interpretar a regra como a definição de "neto" na nossa linguagem.

Lógica

- ♦ A lógica separa entre si a representação e o procedimento, tornando difícil incluir aspectos heurísticos. Isto faz com que sua aplicação a problemas grandes complique.
- ♦ A representação de conhecimento usando Lógica usa fbfs da Lógica de Primeira Ordem e a todas elas é dado o valor de verdade verdadeiro, formando uma base de regras e fatos e constituindo a Base de Conhecimentos. Um mecanismo externo a esta base irá manipulá-la, com regras de inferência (ex. modus ponens) para resolver o problema desejado.

Lógica

- ♦ Calabar foi enforcado;
 - ♦ Getúlio foi presidente;
 - ♦ Todo traidor é enforcado;
 - ♦ Todos os índios eram selvagens;
 - ♦ Tiradentes não era índio;
 - ♦ Tiradentes foi considerado traidor.
- Enforcado(Calabar);
 - presidente(Getúlio);
 - $\forall x \text{ traidor}(x) \Rightarrow \text{enforcado}(x)$;
 - $\forall x \text{ índio}(x) \Rightarrow \text{selvagem}(x)$;
 - $\neg \text{índio}(\text{Tiradentes})$;
 - traidor(Tiradentes).
- ♦ As representações ocasionaram a perda de informações, como é o caso dos tempos das ocorrências dos fatos.
 - ♦ Podemos inferir que Tiradentes foi enforcado, mas não podemos inferir que Calabar era um traidor.

Lógica Clássica

Introdução

- ♦ Importância como teoria matemática.
- ♦ Adequada como método de representação de conhecimento.
- ♦ É O SISTEMA FORMAL MAIS SIMPLES DE QUE APRESENTA UMA TEORIA SEMÂNTICA INTERESSANTE DO PONTO DE VISTA DA REPRESENTAÇÃO DO CONHECIMENTO.
- ♦ Ainda hoje grande parte da pesquisa em IA está ligada direta ou indiretamente à Lógica.

Lógica Clássica

Introdução

- ♦ De maneira geral um sistema lógico consiste em um conjunto de fórmulas e um conjunto de regras de inferência.
- ♦ As fórmulas são sentenças pertencentes a uma linguagem formal cuja sintaxe é dada.
- ♦ A parte de lógica que estuda os valores de verdade é chamada teoria de modelos.
- ♦ Uma regra de inferência é uma regra sintática que quando aplicada repetidamente a uma ou mais fórmulas verdadeiras gera apenas novas fórmulas verdadeiras.
- ♦ A seqüência de fórmulas geradas através da aplicação de regras de inferência sobre um conjunto de inicial de fórmulas é chamada de prova.
- ♦ A parte de lógica que estuda as provas é chamada teoria de provas.

Lógica Clássica

Introdução

- Gödel e Herbrand na década de 30 mostraram que qualquer fórmula verdadeira pode ser provada.
- Church e Turing em 1936 mostraram que não existe um método geral capaz de decidir, em um número finito de passos, se uma fórmula é verdadeira.
- Um dos primeiros objetivos da IA foi a Prova Automática de Teoremas, a partir da segunda metade da década de 60, sendo que a partir daí a lógica passou a ser estudada com método computacional para a solução de problemas.
- O método explora o fato de expressões lógicas poderem ser colocadas em formas canônicas (apenas com operadores "e", "ou" e "não"). O resultado permite a manipulação computacional bastante eficiente.

Lógica Clássica

Introdução

- ♦ Teoria da Resolução de Robinson - 1965. Transforma a expressão a ser provada para a forma normal conjuntiva ou forma clausal. Existe uma regra de inferência única, chamada regra da resolução. Utiliza um algoritmo de casamento de padrões chamado algoritmo de unificação.
- ♦ Base para a Linguagem Prolog.
- ♦ Recentemente Lógicas Não-Padrão ou Não-Clássicas tem sido cada vez mais utilizadas, não somente em IA. Lógica Temporal tem sido utilizada em estudos de programas concorrentes.
- ♦ Em IA estas lógicas vem sendo usadas para tratamento de imprecisão, informações incompletas e evolução com o tempo em que evolui o programa de IA.

Lógica Proposicional

Introdução

- ♦ A Lógica das Proposições se interessa pelas SENTENÇAS DECLARATIVAS, as PROPOSIÇÕES, que podem ser Verdadeiras ou Falsas.
- ♦ No âmbito da IA, a lógica permite a representação de conhecimento e o processo de raciocínio para um sistema inteligente.
- ♦ Como uma linguagem para representação de conhecimento no computador, ela deve ser definida em dois aspectos, A SINTAXE e a SEMÂNTICA.
- ♦ A SINTAXE de uma linguagem descreve as possíveis configurações que podem constituir sentenças.
- ♦ A SEMÂNTICA determina os fatos do mundo aos quais as sentenças se referem., ou seja, ou sistema "acredita" na sentença correspondente.

Lógica Proposicional

- ♦ Sintaxe do Cálculo Proposicional
 - Elementos Válidos:
 - Letras Sentenciais - $P, Q, R, S, T, A1, b3, C$, etc.
 - Conectivos ou Operadores Lógicos:
 - Negação - não é o caso que (\sim) (\neg)
 - Conjunção - e $(\&)$ (\wedge)
 - Disjunção - ou (\vee)
 - Condicional ou implicação: se ...então (\rightarrow) (\Rightarrow)
 - Bicondicional: se e somente se (\leftrightarrow) (\Leftrightarrow)
 - Parênteses
 - $(,)$

Lógica Proposicional

- ♦ Sintaxe do Cálculo Proposicional
 - Tendo já apresentado a lista de símbolos primitivos, quanto a combinação deles, há apenas 3:
 1. Uma letra sentencial sozinha é gramaticalmente correta ou uma fórmula bem formada.
 2. Se qualquer fórmula **A** (tal como (PVQ)) é bem formada, então também o é sua negação $\sim A$ ($\sim(PVQ)$ neste caso).
 3. Se **A** e **B** são fórmulas bem formadas, então também o são $(A \wedge B)$, $(A \vee B)$ e $(A \rightarrow B)$.

Lógica Proposicional

- ♦ Semântica do Cálculo Proposicional
 - Uma fbf pode ter uma interpretação a qual define a semântica da linguagem. Uma interpretação pode ser considerada como um mapeamento do conjunto das fbfs para um conjunto de valores de verdade $\{V, F\}$ ou {Verdadeiro, Falso}.
 - Símbolos Proposicionais
 - podem ter qualquer significado
 - Símbolos constantes
 - verdadeiro: como o mundo é
 - falso: como o mundo não é
 - Sentenças complexas: o significado é derivado das partes
 - Conectivos: podem ser pensados como funções nas quais entram dois valores verdade e sai um

Lógica Proposicional

- ♦ Semântica do Cálculo Proposicional
 - TABELA VERDADE

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	V	F	V	V	F
F	F	V	F	F	V	V

Lógica Proposicional

- ♦ Tabelas Verdade
 - Elas fornecem um teste rigoroso e completo para a validade ou invalidade de formas de argumento da lógica proposicional, além de se constituir em um algoritmo. Quando existe um algoritmo que determina se as formas de argumento expressáveis em um sistema formal são válidas ou não, esse sistema é dito DECIDÍVEL. Desta forma, elas garantem a decidibilidade da lógica proposicional.
 - Uma forma de argumento é válida se e somente se todas as suas instâncias são válidas.
 - Uma instância de uma forma é válida se é impossível que a sua conclusão seja falsa enquanto as suas premissas são verdadeiras.
 - Se a forma for válida, então qualquer instância dela deve ser igualmente válida. Daí podemos utilizar a Tabela-Verdade para estabelecer a validade não só de argumentos, mas também de argumentos específicos.

Lógica Proposicional

- ♦ Traduzindo fatos do mundo real para proposições
 - Exemplo: Encontrar a proposição que traduz a seguinte declaração do mundo real-
"Você não pode andar de patins se você tem menos do que 1,20 m a não ser que você tenha mais do que 16 anos".
 - Definindo:
 - P =você pode andar de patins
 - Q =você tem menos de 1,2 metros
 - R =você tem mais de 16 anos
 - A sentença pode ser escrita formalmente como:
 - $(Q \wedge \sim R) \rightarrow \sim P$

Lógica Proposicional

- ♦ Tabela-Verdade

P	Q	R	$\sim P$	$\sim R$	$(Q \wedge \sim R)$	$(Q \wedge \sim R) \rightarrow \sim P$
F	F	F	V	V	F	V
F	F	V	V	F	F	V
F	V	F	V	V	V	V
F	V	V	V	F	F	V
V	F	F	F	V	F	V
V	F	V	F	F	F	V
V	V	F	F	V	V	F
V	V	V	F	F	F	V

Lógica Proposicional

- ♦ Tabelas Verdade para Formas de Argumento
 - Tabelas-Verdade podem ser usadas, não apenas para definir a semântica do conectivos, mas também para testar a validade de sentenças.
 - A Rainha ou a Princesa comparecerá à cerimônia.
 - A Princesa não comparecerá.
 - Logo, a Rainha comparecerá.

$R \vee P, \neg P \therefore R$

P	R	$\neg P$	$R \vee P$	R	$((R \vee P) \wedge \neg P) \rightarrow R$
V	V	F	V	V	V
V	F	F	V	F	V
F	V	V	V	V	V
F	F	V	F	F	V

Lógica Proposicional

- ♦ Tabelas Verdade
 - As Tabelas-Verdade garantem a decidibilidade da lógica proposicional, porém elas são enfadonhas e ineficazes (NP-COMPLETAS) para um número muito grande de fórmulas-atômicas.
- ♦ Uma sentença pode ser verdadeira ou falsa dependendo da semântica e do estado do mundo.
- ♦ Raciocínio ou Inferência
 - processo pelo qual conclusões são alcançadas
- ♦ Sentenças válidas ou necessariamente verdadeiras
 - é verdadeira baixo todas as possíveis interpretações do mundo
 - Ex.: Hoje vai chover ou hoje não vai chover.

Lógica Proposicional

- ♦ Regras de Inferência

1. Modus Ponens (MP)

- De um condicional e seu antecedente, podemos inferir o seu conseqüente.
- $A \rightarrow B, A \vdash B$

2. Eliminação da Negação (\sim E)

- De uma fbf $\sim(\sim A)$, podemos inferir A .
- $\sim(\sim A) \vdash A$

3. Introdução da Conjunção (\wedge I)

- De quaisquer fbfs A e B podemos inferir $A \wedge B$.
- $A, B \vdash A \wedge B$

Lógica Proposicional

4. Eliminação da Conjunção ($\wedge E$)

- De uma conjunção podemos inferir qualquer uma de suas sentenças.
- $A \wedge B \vdash A$

5. Introdução da Disjunção ($\vee I$)

- De uma fbf A , podemos inferir a disjunção de A com qualquer fbf.
- $A \vdash A \vee B$

6. Eliminação da Disjunção ($\vee E$)

- De fbfs da forma $A \vee B$, $A \rightarrow C$ e $B \rightarrow C$, podemos inferir C .

Lógica Proposicional

7. Introdução do Bicondicional (\leftrightarrow I)
 - De quaisquer fbfs da forma $A \rightarrow B$ e $B \rightarrow A$, podemos inferir $A \leftrightarrow B$.
8. Eliminação do Bicondicional (\leftrightarrow E)
 - De uma fbf da forma $A \leftrightarrow B$, podemos inferir $A \rightarrow B$ e $B \rightarrow A$.
9. Prova do Condicional (PC)
 - Dada uma derivação de uma fbf B a partir de uma hipótese A, podemos descartar a hipótese e inferir $A \rightarrow B$. A Prova do Condicional é também chamada Teorema da Dedução e é normalmente utilizada se o conseqüente é da forma $A \rightarrow B$.
10. Redução ao Absurdo (RAA)
 - Dada uma derivação de uma contradição a partir de uma hipótese A, podemos descartar a hipótese e inferir $\sim A$.

Lógica Proposicional

♦ Regras Derivadas de Inferência

1. Modus Tollens (MT)

- De fbfs da forma $A \rightarrow B$ e $\sim B$, infere-se $\sim A$.

2. Silogismo Hipotético (SH)

- De fbfs da forma $A \rightarrow B$ e $B \rightarrow C$, infere-se $A \rightarrow C$.

3. Regra da Absorção (ABS)

- De fbfs da forma $A \rightarrow B$, infere-se $A \rightarrow (A \wedge B)$.

4. Regra do Dilema Construtivo (DC)

- De fbfs da forma $A \vee B$, $A \rightarrow C$ e $B \rightarrow D$, infere-se $C \vee D$.

5. Regra da Repetição (RE)

- De fbf da forma A , infere-se A .

6. Regra do Silogismo Disjuntivo (SD)

- De fbfs da forma $A \vee B$ e $\sim A$, infere-se B .

Lógica Proposicional

♦ Árvore de Refutação

- São uma outra maneira de garantir a decidibilidade da Lógica Proposicional.

- REGRAS PARA ÁRVORE DE REFUTAÇÃO

1. Inicia-se colocando-se as PREMISSAS e a NEGAÇÃO DA CONCLUSÃO.

(A idéia é encontrar contradições de modo a poder concluir a validade da conclusão.)

2. Aplica-se repetidamente uma das regras a seguir:

2.1. Negação (\neg): Se um ramo aberto contém uma fórmula e sua negação, coloca-se um "X" no final do ramo, de modo a representar um ramo fechado.

(um ramo termina se ele se fecha ou se as fórmulas que ele contém são apenas fórmulas-atômicas ou suas negações, tal que mais nenhuma regra se aplica às suas fórmulas. Desta forma tem-se um ramo fechado, que é indicado por um X, enquanto o ramo aberto não é representado por um X.)

Lógica Proposicional

♦ Árvore de Refutação

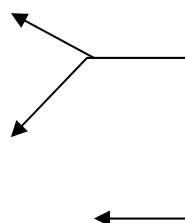
2.2. Negação Negada ($\neg \neg$): Se um ramo aberto contém uma fórmula não ticada da forma $\neg \neg \emptyset$, tica-se $\neg \neg \emptyset$ e escreve-se \emptyset no final de cada ramo aberto que contém $\neg \neg \emptyset$ ticada.

2.3. Conjunção (\wedge): Se um ramo aberto contém uma fórmula não ticada da forma $\emptyset \wedge \beta$, tica-se, $\emptyset \wedge \beta$ e escreve-se \emptyset e β no final de cada ramo aberto que contém $\emptyset \wedge \beta$ ticada.

$$P \wedge Q \vdash \neg \neg P$$

$$\begin{array}{l} 1. P \wedge Q \\ 2. \neg \neg \neg P \\ 3. P \\ 4. Q \\ 5. \neg P \\ 6. X \end{array}$$

$$\begin{array}{l} 1 \wedge \\ 1 \wedge \\ 2 \neg \neg \\ 3 \neg \end{array}$$



A árvore de refutação está **COMPLETA**, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou **CONTRADIÇÕES**, e portanto, a **FORMA É VÁLIDA**.

Lógica Proposicional

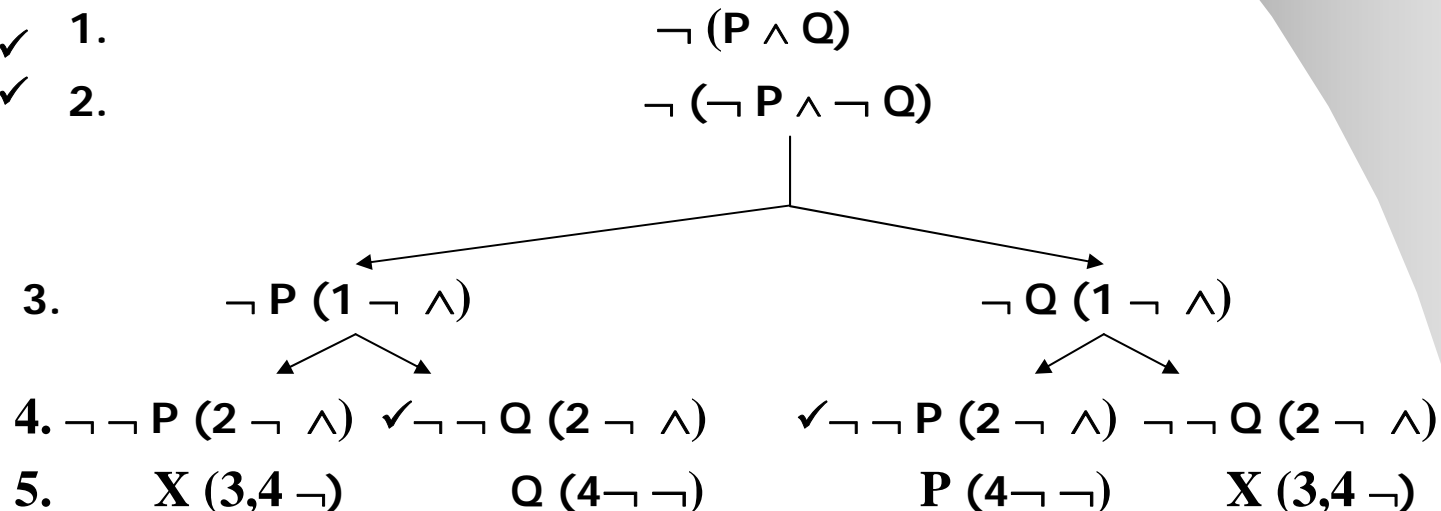
♦ Árvore de Refutação

2.4. **Conjunção Negada ($\neg \wedge$):** Se um ramo aberto contém uma fórmula não tícada da forma $\neg (\emptyset \wedge \beta)$, tica-se, $\neg (\emptyset \wedge \beta)$ e BIFURCA-SE o final de cada ramo aberto que contém $\neg (\emptyset \wedge \beta)$ tícada, no final do primeiro ramo se escreve $\neg \emptyset$ e no final do segundo ramo se escreve $\neg \beta$.

$$\neg (P \wedge Q) \vdash \neg P \wedge \neg Q$$

✓ 1.

✓ 2.



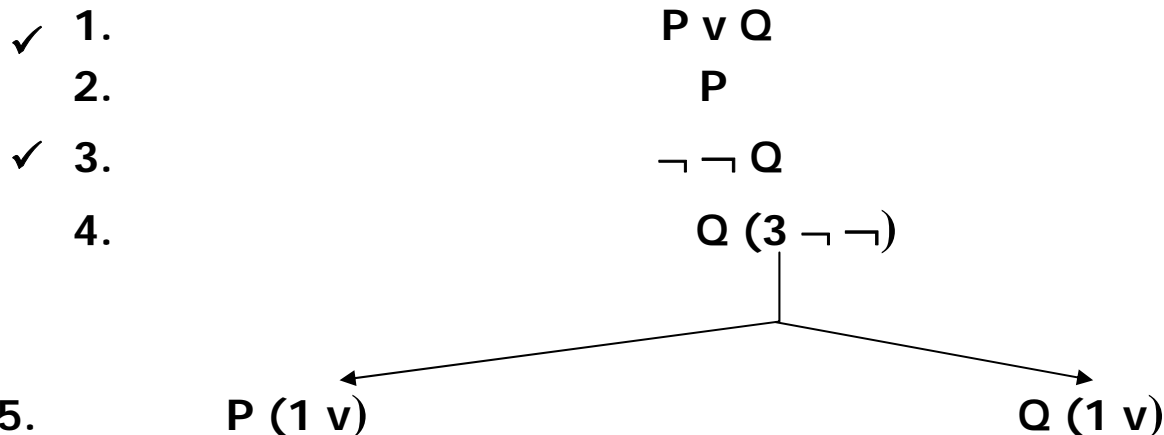
O exemplo acima nos mostra que há dois ramos abertos, conseqüentemente a fórmula é inválida, o que significa que estes ramos são contra-exemplos.

Lógica Proposicional

♦ Árvore de Refutação

2.5. Disjunção (\vee): Se um ramo aberto contém uma fórmula não tícada da forma $\emptyset \vee \beta$, tica-se, $\emptyset \vee \beta$ e BIFURCA-SE o final de cada ramo aberto que contém $\emptyset \vee \beta$ tícada, no final do primeiro ramo se escreve \emptyset e no final do segundo ramo se escreve β .

$$P \vee Q, P \vdash \neg Q$$



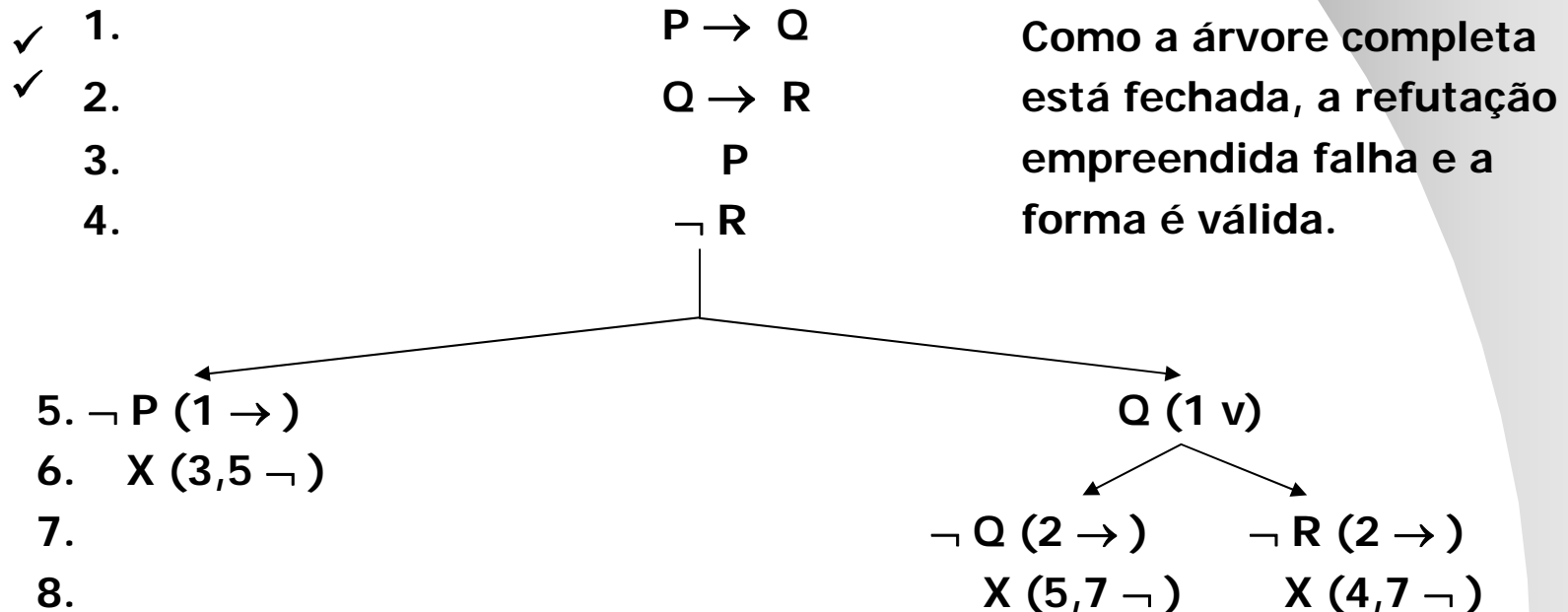
O exemplo acima nos mostra que há dois ramos abertos, conseqüentemente a fórmula é inválida, o que significa que estes ramos são contra-exemplos.

Lógica Proposicional

♦ Árvore de Refutação

2.6. Condicional (\rightarrow): Se um ramo aberto contém uma fórmula não tificada da forma $\emptyset \rightarrow \beta$, tica-se, $\emptyset \rightarrow \beta$ e BIFURCA-SE o final de cada ramo aberto que contém $\emptyset \rightarrow \beta$ tificada, no final do primeiro ramo se escreve $\neg \emptyset$ e no final do segundo ramo se escreve β .

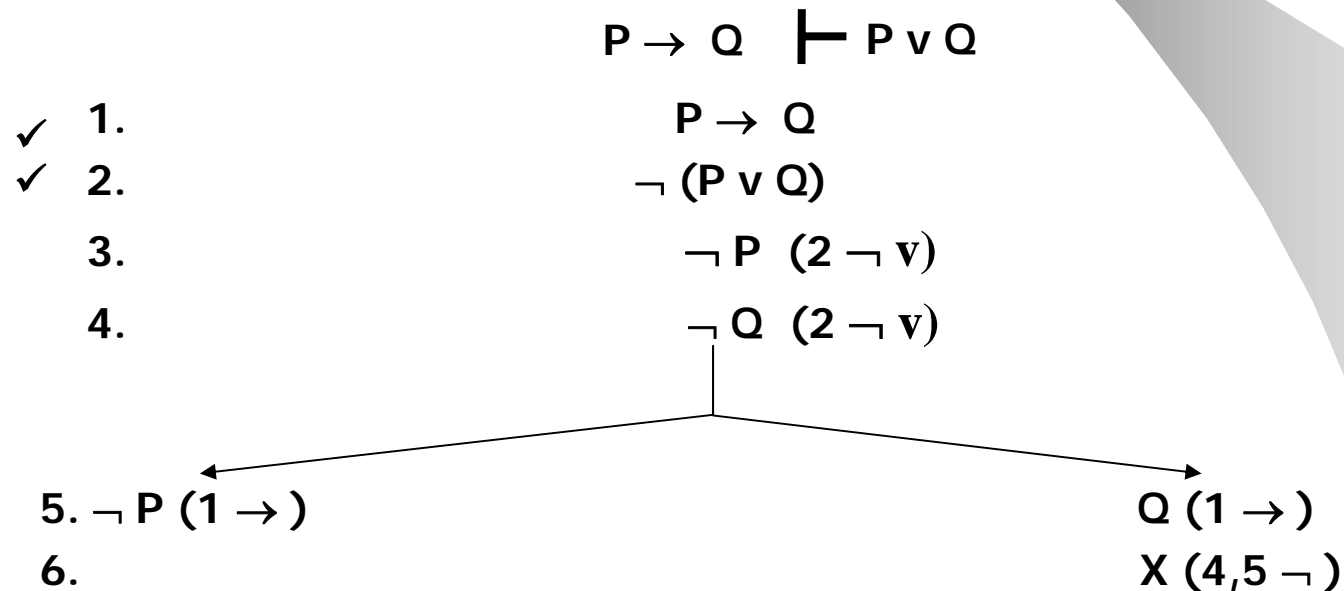
$P \rightarrow Q, Q \rightarrow R, P \vdash R$



Lógica Proposicional

♦ Árvore de Refutação

2.7. Disjunção Negada ($\neg \vee$): Se um ramo aberto contém uma fórmula não ticada da forma $\neg (\emptyset \vee \beta)$, tica-se, $\neg (\emptyset \vee \beta)$ e **ESCREVE-SE** $\neg \emptyset$ e $\neg \beta$ no final de cada ramo aberto que contém $\neg (\emptyset \vee \beta)$ ticada.

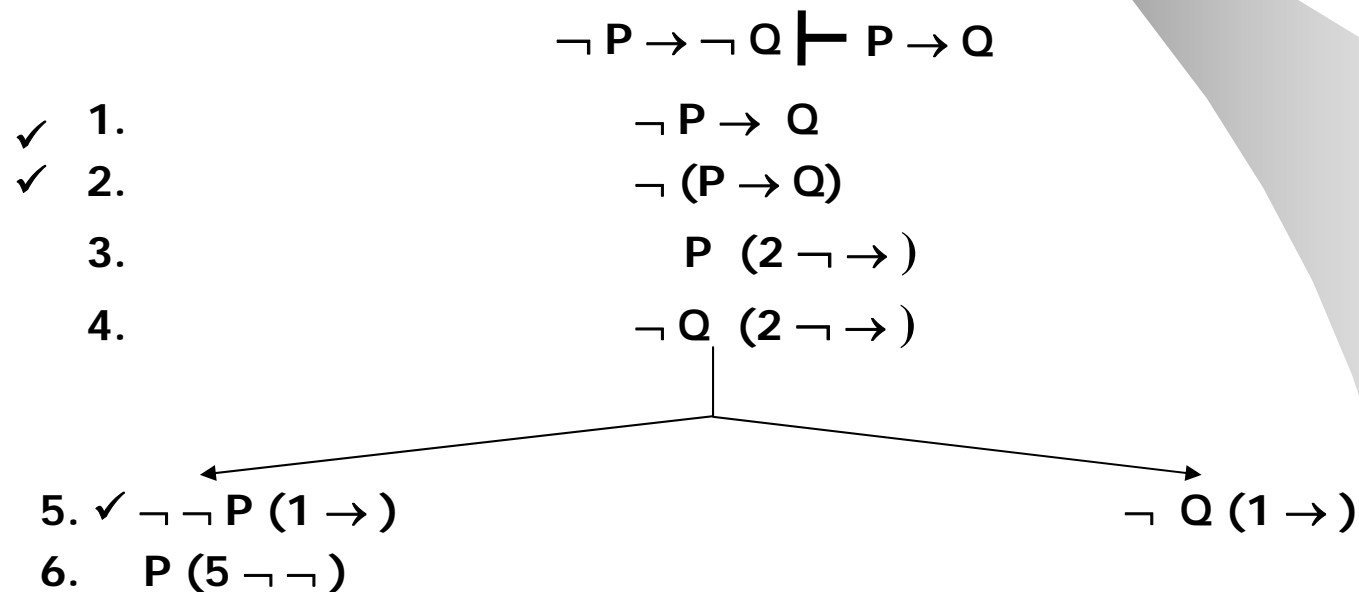


O ramo aberto indica que a forma é inválida

Lógica Proposicional

♦ Árvore de Refutação

2.8. Condicional Negado ($\neg \rightarrow$): Se um ramo aberto contém uma fórmula não ticada da forma $\neg(\emptyset \rightarrow \beta)$, tica-se, $\neg(\emptyset \rightarrow \beta)$ e **ESCREVE-SE** \emptyset e $\neg \beta$ no final de cada ramo aberto que contém $\neg(\emptyset \rightarrow \beta)$ ticada.



Os ramos abertos indica que a forma é inválida

Lógica Proposicional

♦ Árvore de Refutação

2.9. Bicondicional (\leftrightarrow): Se um ramo aberto contém uma fórmula não ticada da forma $\emptyset \leftrightarrow \beta$, tica-se, $\emptyset \leftrightarrow \beta$ e BIFURCA-SE o final de cada ramo aberto que contém $\emptyset \leftrightarrow \beta$ ticada, no final do primeiro ramo se escreve \emptyset e β e no final do segundo ramo se escreve $\neg \emptyset$ e $\neg \beta$.

2.10. Bicondicional Negado ($\neg \leftrightarrow$): Se um ramo aberto contém uma fórmula não ticada da forma $\neg (\emptyset \leftrightarrow \beta)$, tica-se, $\neg (\emptyset \leftrightarrow \beta)$ e BIFURCA-SE o final de cada ramo aberto que contém $\neg (\emptyset \leftrightarrow \beta)$ ticada, no final do primeiro ramo se escreve \emptyset e $\neg \beta$ e no final do segundo ramo se escreve $\neg \emptyset$ e β .

$$P \leftrightarrow Q, \neg P \vdash \neg Q$$

Lógica de Primeira Ordem

♦ Introdução

- A Lógica das Proposições tem um poder de representação limitado.
- Na Lógica Proposicional se utiliza apenas sentenças completas, isto é, as proposições para representar o conhecimento sobre o Mundo.
- A Lógica de Primeira Ordem ou Lógica dos Predicados, ou Cálculo dos Predicados, é uma extensão da Lógica das Proposições em que se consideram variáveis e quantificadores sobre as variáveis.
- A Lógica dos Predicados se preocupa em introduzir noções lógicas para expressar qualquer conjunto de fatos através de Classes de Atributos e de Quantificadores.

Lógica de Primeira Ordem

♦ Introdução

- Lógica de Primeira Ordem considera o mundo com:
 - Objetos (casas, cores, etc.)
 - Relações (maior que, dentro, tem cor, etc.)
 - Propriedades (vermelho, redondo, etc.)
 - Funções (pai de, melhor amigo, etc.)

Lógica de Primeira Ordem

- ♦ Quantificadores:
 - São operadores lógicos que em vez de indicarem relações entre sentenças, expressam relações entre conjuntos designados pelas classes de atributos lógicos.
 - Quantificador Universal (\forall):
 - Este tipo de quantificador é formado pelas expressões "todo" e "nenhum".
 - Quantificador Existencial (\exists):
 - Este tipo de quantificador é formado pelas expressões "existe um", "existe algum", "pelo menos um" ou "para algum".

Lógica de Primeira Ordem

- ♦ Quantificadores:

- Exemplos:

- Todo homem é mortal, ou seja, qualquer que seja x (do Universo), se x é Homem, então x é Mortal.

- $\forall x (H(x) \rightarrow M(x))$.

- Nenhum homem é vegetal, ou sejam qualquer que seja x , se x é Homem, em x NÃO É Vegetal.

- $\forall x (H(x) \rightarrow \sim V(x))$.

- Pelo menos um homem é inteligente, ou seja, existe pelo menos um x em que x seja Homem e x seja Inteligente.

- $\exists x (H(x) \wedge I(x))$

Lógica de Primeira Ordem

- ♦ Variáveis:
 - Designam objetos "desconhecidos" do Universo. "Alguém". São normalmente representados por letras minúsculas de "u" a "z".
- ♦ Letras Nominais:
 - Designam objetos "conhecidos" do Universo. "João", "Pedro", etc. São normalmente representados por letras minúsculas de "a" a "t".
- ♦ Predicados:
 - Descrevem alguma coisa ou característica de um ou mais objetos. São normalmente denotados por letras maiúsculas.
 - João ama Maria: $A(a,b)$
 - João ama alguém: $\exists x A(a,x)$
 - João ama todo mundo: $\forall x A(a,x)$

Lógica de Primeira Ordem

- ♦ Regras de Inferência para o Cálculo de Predicados
 - Todas as regras definidas no Cálculo Proposicional continuam válidas no Cálculo de Predicados, apenas referenciando-as para os quantificadores.

• Ex.: $\sim F(a) \vee \exists x F(x), \exists x F(x) \rightarrow P \vdash F(a) \rightarrow P$.

Prova:

1.	$\sim F(a) \vee \exists x F(x)$	Premissa
2.	$\exists x F(x) \rightarrow P$	Premissa
3.	$F(a)$	Hipótese
4.	$\sim \sim F(a)$	3 DN
5.	$\exists x F(x)$	1,3 SD
6.	P	2,5 MP
7.	$F(a) \rightarrow P$	3,6 PC

Lógica de Primeira Ordem

- ♦ Regras de Inferência para o Cálculo de Predicados

- Intercâmbio de Quantificadores

1. $\sim(\forall x \sim F(x)) = \exists x F(x)$

2. $\sim(\forall x F(x)) = \exists x \sim F(x)$

3. $\forall x \sim F(x) = \sim(\exists x F(x))$

4. $\forall x F(x) = \sim(\exists x \sim F(x))$

$$\forall x \neg \text{GostarPagar}(x, \text{Impostos}) \equiv \neg \exists x \text{GostarPagar}(x, \text{Impostos})$$

- Como \forall é na verdade uma conjunção sobre o universo de objetos e o \exists é uma disjunção, não é surpreendente que eles obedeçam as Lei de De Morgan.

Lógica de Primeira Ordem

- ♦ Igualdade ou Identidade
 - É um símbolo que se adiciona ao Cálculo de Predicados com o propósito de expressar o fato de dois termos se referirem ao mesmo objeto, ou seja, "é idêntico a" ou "é a mesma coisa que".

Exemplos:

- O Pai de João é Henrique.
 $\text{Pai_de}(\text{João}) = \text{Henrique}$
Pai de João e Henrique se referem ao mesmo objeto.
- O Pai de João é também Avô de Pedro.
 $\text{Pai_de}(\text{João}) = \text{Avô_de}(\text{Pedro})$

Lógica de Primeira Ordem

- ♦ Regras de Inferência para o Cálculo de Predicados

1. Eliminação Universal (EU)

- De uma fbf quantificada universalmente $\forall x F(x)$, infere-se uma fbf da forma $F(a)$, a qual resulta de se substituir cada ocorrência da variável x em F por uma letra nominal a .

2. Introdução do Existencial (IE)

- De uma fbf F contendo uma letra nominal a , infere-se uma fbf da forma $\exists x F(x)$, onde $F(x)$ é o resultado de se substituir uma ou mais ocorrências de a em F por uma variável x QUE NÃO OCORRA em F .
- a pode ocorrer em uma hipótese não utilizada ainda, ou em uma premissa, normalmente a é um termo independente (ground term);
- IE permite introduzir somente um quantificador existencial por vez e somente do lado esquerdo da fórmula.

Lógica de Primeira Ordem

- ♦ Regras de Inferência para o Cálculo de Predicados

3. Eliminação do Existencial (EE)

- De uma fbf quantificada existencialmente $\exists x F(x)$ podemos inferir $F(a)$, contanto que a letra nominal NÃO OCORRA em $F(x)$, NEM EM QUALQUER HIPÓTESE, NEM EM QUALQUER PASSO ANTERIOR DA DERIVAÇÃO.

- Exemplo:

- A lei diz que é crime um Americano vender armas a nações hostis.
- O Nação Iraque, um inimigo da América, possui alguns mísseis, e todos os seus mísseis foram vendidos a ele pelo coronel West, que é um Americano.

Lógica de Primeira Ordem

- Exemplo:

1. $\forall x,y,z \text{Americano}(x) \wedge \text{Arma}(y) \wedge \text{Nação}(z) \wedge \text{Hostil}(z) \wedge \text{Vender}(x,y,z) \rightarrow \text{Criminoso}(x)$
2. $\exists x \text{Possui}(\text{Iraque},x) \wedge \text{Míssil}(x)$
3. $\forall x \text{Possui}(\text{Iraque},x) \wedge \text{Míssil}(x) \rightarrow \text{Vender}(\text{West},\text{Iraque},x)$
4. $\forall x \text{Míssil}(x) \rightarrow \text{Arma}(x)$
5. $\forall x \text{Inimigo}(\text{América},x) \rightarrow \text{Hostil}(x)$
6. $\text{Americano}(\text{West})$
7. $\text{Nação}(\text{Iraque})$
8. $\text{Inimigo}(\text{Iraque},\text{América})$
9. $\text{Nação}(\text{América})$

Lógica de Primeira Ordem

- Prova

- 10. Possui(Iraque,M1) \wedge Míssil(M1) - 2 EE
- 11. Possui(Iraque,M1) - 10 \wedge E
- 12. Míssil(M1) - 10 \wedge E
- 13. Míssil(M1) \rightarrow Arma(M1) - 4 EU
- 14. Arma(M1) - 12,13 MP
- 15. Possui(Iraque,M1) \wedge Míssil(M1) \rightarrow Vender(West,Iraque,M1)-3EU
- 16. Vender(West,Iraque,M1) - 11,12 MP
- 17. Americano(West) \wedge Arma(M1) \wedge Nação(Iraque) \wedge Hostil(Iraque) \wedge Vender(West,Iraque,M1) \rightarrow Criminoso(West) - 1 EU
- 18. Inimigo(América,Iraque) \rightarrow Hostil(Iraque) - 5 EU
- 19. Hostil(Iraque) - 8,18 MP
- 20. Americano(West) \wedge Arma(M1) \wedge Nação(Iraque) \wedge Hostil(Iraque) \wedge Vender(West,Iraque,M1) - 6,14,7,19,16 \wedge I
- 21. Criminoso(West) - 20,17 MP

Lógica de Primeira Ordem

- ♦ Se formularmos o processo de achar uma prova como um processo de busca, então esta prova é a solução de um problema de busca:
 - Estado Inicial = Base de Conhecimento (sentenças 1 - 9)
 - Operadores = regras de inferência aplicáveis
 - Estado Final = Base de Conhecimento contendo a sentença Criminoso(West)
- ♦ Isto é muito difícil pois a solução está na profundidade 12 e o fator de ramificação é bastante grande. Porque?

Lógica de Primeira Ordem

- **Árvores de Refutação**
 - São uma generalização da técnica utilizada na Lógica Proposicional.
 - A técnica de árvore de refutação generalizada incorpora as regras da lógica proposicional e acrescenta 6 novas regras para inferir em sentenças que contêm quantificadores e o predicado de identidade.
 - Algumas árvores do cálculo dos predicados empregam somente as regras do cálculo proposicional.
 - NO CÁLCULO DE PREDICADOS, AS ÁRVORES DE REFUTAÇÃO NÃO APRESENTAM UMA LISTA COMPLETA DE CONTRA-EXEMPLOS, MAS SIM, UM "MODELO DE UNIVERSO" QUE CONTEM EXATAMENTE OS OBJETOS MENCIONADOS PELO NOME NO RAMO.

Lógica de Primeira Ordem

- Árvores de Refutação

$$\forall x P(x) \rightarrow \forall x G(x), \neg \forall x G(x) \vdash \neg \forall x P(x)$$

✓ 1. $\forall x P(x) \rightarrow \forall x G(x)$

2. $\neg \forall x G(x)$

3. $\forall x P(x)$

4. $\neg \forall x P(x) \rightarrow \forall x G(x) \rightarrow$

5. $\text{X } 3,4 \neg \quad \text{X } 2,4 \neg$

A árvore de refutação está **COMPLETA**, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou **CONTRADIÇÕES**, e portanto, a **FORMA É VÁLIDA**.

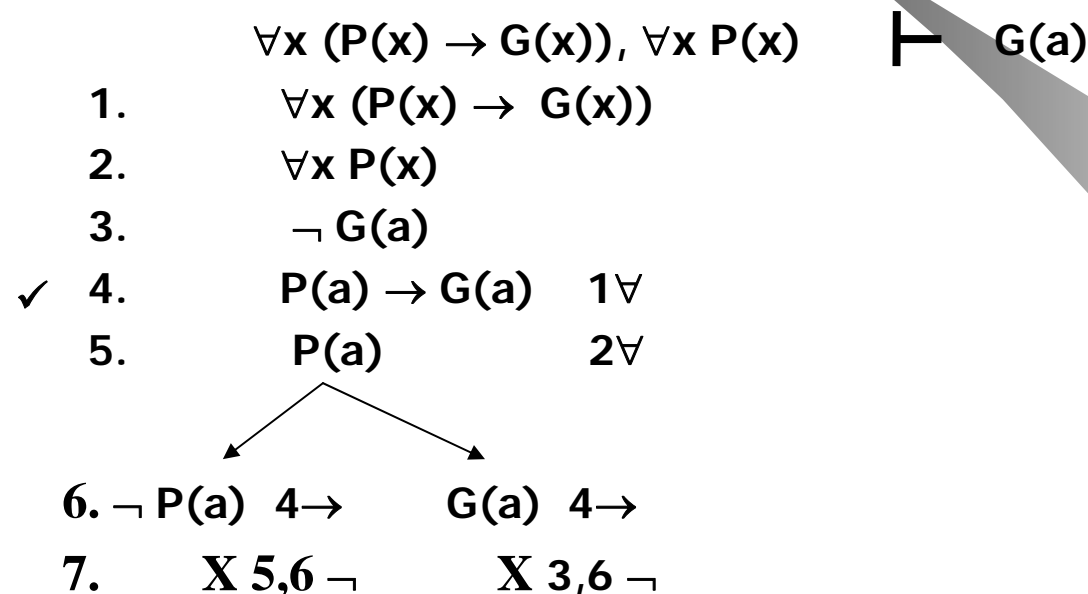
Lógica de Primeira Ordem

- Árvores de Refutação
 - 1. Quantificação Universal (\forall):
 - Se uma fórmula bem formada do tipo $\forall \beta \emptyset$ aparece num ramo aberto e se α é uma constante (ou letra nominal) que ocorre numa fbf naquele ramo, então ESCREVE-SE \emptyset^α / β (o resultado de se substituir todas as ocorrências β em \emptyset por α) no final do ramo.
 - Se nenhuma fbf contendo uma letra nominal aparece no ramo, então escolhemos uma letra nominal α e ESCREVE-SE \emptyset^α / β no final do ramo.
 - Em cada caso, NÃO TICAMOS $\forall \beta \emptyset$.

Lógica de Primeira Ordem

- Árvores de Refutação

- 1. Quantificação Universal (\forall):



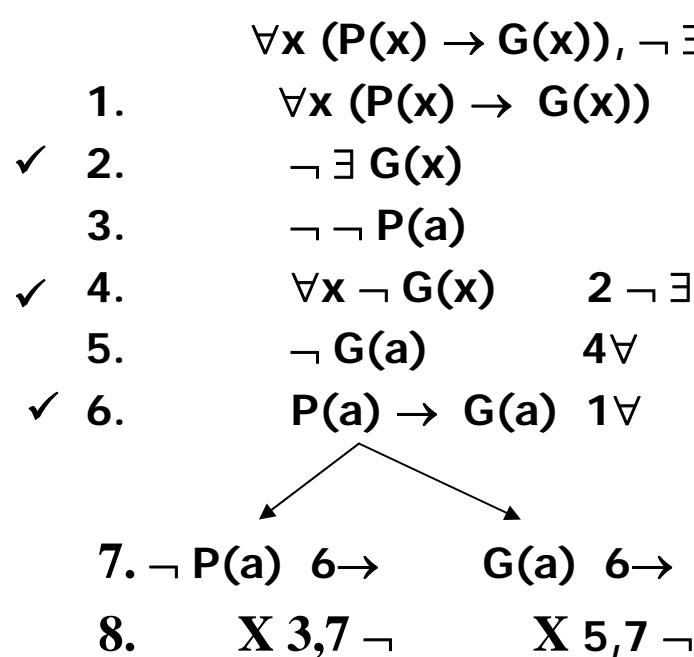
A árvore de refutação está **COMPLETA**, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou **CONTRADIÇÕES**, e portanto, a **FORMA É VÁLIDA**.

Lógica de Primeira Ordem

- Árvores de Refutação

- 2. Quantificação Existencial Negada ($\neg \exists$):

- Se uma fórmula bem formada não tificada da forma $\neg \exists \beta \emptyset$ aparece num ramo aberto, tica-se a fórmula e **ESCREVE-SE** $\forall \beta \neg \emptyset$ no final de cada ramo aberto que contém a fbf tificada.



A árvore de refutação está **COMPLETA**, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou **CONTRADIÇÕES**, e portanto, a **FORMA É VÁLIDA**.

Lógica de Primeira Ordem

- Árvores de Refutação

- 3. Quantificação Universal Negada ($\neg \forall$):

- Se uma fórmula bem formada não tificada da forma $\neg \forall \beta \emptyset$ aparece num ramo aberto, tica-se a fórmula e **ESCREVE-SE** $\exists \beta \neg \emptyset$ no final de cada ramo aberto que contém a fbf tificada.

	$\exists x (\forall y P(x,y)) \vdash$	$\forall x (\exists y P(y,x))$
✓ 1.	$\exists x (\forall y P(x,y))$	
✓ 2.	$\neg \forall x (\exists y P(y,x))$	
3.	$\forall y P(a,y)$	1 \exists
✓ 4.	$\exists x (\neg \exists y P(y,x))$	2 $\neg \forall$
✓ 5.	$\neg \exists y P(y,b)$	4 \exists
6.	$\forall y \neg P(y,b)$	5 $\neg \exists$
7.	$\neg P(a,b)$	6 \forall
8.	$P(a,b)$	3 \forall
9.	X	7,8 \neg

A fórmula testada é válida

Lógica de Primeira Ordem

- Árvores de Refutação

- 4. Quantificação Existencial (\exists):

- Se uma fórmula bem formada não tificada da forma $\exists \beta \emptyset$ aparece num ramo aberto, tica-se a fórmula e escolhe-se uma letra nominal α QUE NAO APARECEU NAQUELE RAMO e ESCREVE-SE \emptyset^α / β (o resultado de se substituir todas as ocorrências β em \emptyset por α) no final do ramo.

	$\exists x P(x) \vdash$	$\forall x P(x)$
✓ 1.	$\exists x P(x)$	
✓ 2.	$\neg \forall x P(x)$	
3.	$P(a)$	1 \exists
✓ 4.	$\exists x \neg P(x)$	2 $\neg \forall$
5.	$\neg P(b)$	4 \exists

A fórmula testada é INVÁLIDA POR HAVER RAMOS ABERTOS (linha 5)

Lógica de Primeira Ordem

- Árvores de Refutação

- 5. Identidade (=):

- Se uma fórmula do tipo $\alpha = \beta$ aparece num ramo aberto e se uma outra fbf \emptyset contendo α ou β aparece não tica naquele ramo, então escrevemos no final do ramo qualquer fbf que não esteja no ramo, que é o resultado de se substituir uma ou mais ocorrências de qualquer uma dessas letras nominais pela outra em \emptyset .
- Não se tica $\alpha = \beta$ nem \emptyset .

$$a = b \vdash P(a,b) \rightarrow P(b,a)$$

1.	$a = b$	
2.	$\neg (P(a,b) \rightarrow P(b,a))$	A fórmula testada é válida
✓ 3.	$\neg (P(a,a) \rightarrow P(a,a))$	1,2 =
4.	$P(a,a)$	3 $\neg \rightarrow$
5.	$\neg P(a,a)$	3 $\neg \rightarrow$
6.	X	4,5 \neg

Lógica de Primeira Ordem

- Árvores de Refutação
 - Identidade Negada ($\neg=$):
 - Fechamos qualquer ramo aberto no qual uma fbf do tipo $\neg (\alpha = \alpha)$ ocorra.

	$a = b$	\vdash	$b = a$
1.	$a = b$		
2.	$\neg (b = a)$		
✓ 3.	$\neg (a = a)$		1,2 =
4.	X		3 $\neg =$

A fórmula testada é válida

Prova Automática de Teoremas

- A capacidade de se demonstrar teoremas é uma das partes integrantes da inteligência humana.
- Este tipo de prova foi pesquisada e desenvolvida a partir da segunda metade dos anos 60.
- A partir da introdução, por Robinson e Smullyan, em 1960, de procedimentos eficientes para demonstração automática de teoremas por computador, a lógica passou a ser estudada também como método computacional para a solução de problemas.
- Uma das áreas que mais faz uso desta técnica é a dos Sistemas Especialistas (SEs).
- O objetivo principal da Prova Automática de Teoremas é provar que uma fórmula (teorema) é consequência lógica de outras fórmulas.

Prova Automática de Teoremas

- Os métodos adotados normalmente não utilizam a prova direta (através de regras de inferência), mas sim a PROVA POR REFUTAÇÃO (prova indireta), demonstrando que a negação da fórmula leva a inconsistências.
- SE A NEGAÇÃO DE UM TEOREMA É FALSA, ENTÃO ELE SERÁ VERDADEIRO.
- Os procedimentos de prova exploram o fato de expressões lógicas (fórmulas) poderem ser colocados em formas canônicas, isto é, apenas com os operadores "e", "ou" e "não".
- O método da prova por refutação aplicado à lógica de primeira ordem é muito conveniente e com seu emprego não haverá perda de generalidade, porém, exige-se que as fórmulas estejam na forma de cláusulas.

Prova Automática de Teoremas

- A TEORIA DA RESOLUÇÃO, proposta por Robinson em 1965 a partir dos trabalhos de Herbrand, Davis e Putnam, parte da transformação da fórmula a ser provada para a forma canônica conhecida como forma clausal.
- O método é baseado em uma regra de inferência única, chamada REGRA DA RESOLUÇÃO, e utiliza intensivamente um algoritmo de casamento de padrões chamado ALGORITMO DE UNIFICAÇÃO.
- O fato de ser possível associar uma semântica operacional a um procedimento de prova automática de teoremas permitiu a definição de uma linguagem de programação baseada em lógica, a linguagem PROLOG.
- Ainda hoje a área de prova automática de teoremas permanece bastante ativa, sendo objeto de diversas conferências internacionais.

Prova Automática de Teoremas

Algumas Definições

- PROVA: É a demonstração de que um teorema (ou fórmula) é verdadeiro.
- FORMA NORMAL CONJUNTIVA: É quando uma fórmula F for composta de uma conjunção de outras fórmulas $(F1 \wedge F2 \wedge \dots \wedge F_n)$.
- FORMA NORMAL DISJUNTIVA: É quando uma fórmula F for composta de uma disjunção de outras fórmulas $(F1 \vee F2 \vee \dots \vee F_n)$.
- FORMA NORMAL PRENEX: É quando numa fórmula F , na lógica de primeira ordem, todos os quantificadores existentes prefixam a fórmula, isto é, se e somente se estiver na forma $Q_1x_1 \dots Q_nx_n(M)$.

Onde: $Q_ix_i = \forall x_i$ ou $\exists x_i$, e
 (M) = uma fórmula que não contenha quantificadores.

Prova Automática de Teoremas

- **Procedimento para Obtenção da Forma Normal Prenex**

1. Eliminar os conectivos lógicos \rightarrow e \leftrightarrow usando as seguintes leis:

- $F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$
- $(F \rightarrow G) = \neg F \vee G$

2. Repetir o uso das seguintes leis:

- $\neg \neg F = F$
- $\neg (F \vee G) = \neg F \wedge \neg G$
- $\neg (F \wedge G) = \neg F \vee \neg G$
- $\neg (\forall x F(x)) = \exists x (\neg F(x))$
- $\neg (\exists x F(x)) = \forall x (\neg F(x))$

Estas leis são utilizadas para trazer os sinais de negação para antes dos átomos.

3. Padronizar as variáveis, se necessário, de modo que cada quantificador possua sua própria variável.

Prova Automática de Teoremas

- ♦ **Procedimento para Obtenção da Forma Normal Prenex**
 4. Usar as leis abaixo de forma a mover os quantificadores para a esquerda da fórmula para obter a Forma Normal PRENEX.
 - $Qx F(x) \vee G = Qx (F(x) \vee G)$
 - $Qx F(x) \wedge G = Qx (F(x) \wedge G)$
 - $\forall x F(x) \wedge \forall x G(x) = \forall x (F(x) \wedge G(x))$
 - $\exists x F(x) \vee \exists x G(x) = \exists x (F(x) \vee G(x))$
 - $Q_1x F(x) \vee Q_2x G(x) = Q_1x Q_2z (F(x) \vee G(z))$
 - $Q_3x F(x) \wedge Q_4x G(x) = Q_3x Q_4z (F(x) \wedge G(z))$

EXEMPLO 1

$$\forall x P(x) \rightarrow \exists x Q(x)$$

- $\forall x P(x) \rightarrow \exists x Q(x) = \neg \forall x P(x) \vee \exists x Q(x)$
- $\exists x (\neg P(x)) \vee \exists x Q(x)$
- $\exists x (\neg P(x) \vee Q(x))$

Prova Automática de Teoremas

- ♦ Procedimento para Obtenção da Forma Normal Prenex

EXEMPLO 2

$$\forall x \forall y ((\exists z (P(x,z) \wedge P(y,z)) \rightarrow \exists u Q(x,y,u)) =$$

- $\forall x \forall y (\neg (\exists z (P(x,z) \wedge P(y,z))) \vee \exists u Q(x,y,u)) =$
- $\forall x \forall y (\forall z (\neg P(x,z) \vee \neg P(y,z))) \vee \exists u Q(x,y,u)) =$
- $\forall x \forall y \forall z \exists u (\neg P(x,z) \vee \neg P(y,z) \vee Q(x,y,u))$

Prova Automática de Teoremas

- ♦ **Eliminação dos quantificadores existenciais (Skolemização ou Funções de Skolem)**
 - Quando uma fórmula está na forma normal Prenex, pode-se eliminar os quantificadores existenciais por uma função, se as variáveis estiverem no escopo do quantificador universal; caso estejam fora, substitui-se por uma constante.
 - As constantes e funções usadas para substituir as variáveis existenciais são chamadas constante e funções de Skolem
 - Ex.: $\forall x \exists y P(x,y)$ Skolemizando: $\forall x P(x,f(x))$
 - onde $f(x)$ tem por único propósito garantir que existe algum valor (y) que depende de x pois está dentro do seu escopo. No entanto, se o quantificador existencial não residir no escopo do quantificador universal, como em $\exists y \forall x P(x,y)$, a variável quantificada existencialmente será substituída por uma constante $\forall x P(x,a)$ que assegure sua existência, assim como sua independência de qualquer outra variável.

Prova Automática de Teoremas

- ♦ **Procedimento para Obtenção da Forma Clausal**
 - Cláusula é uma disjunção de literais
 - 1. Passar para a forma normal PRENEX.
 - 2. Skolemizar as variáveis quantificadas existencialmente.
 - 3. Abandona-se os quantificadores pré-fixados.

EXEMPLO

$$\forall x \forall y ((\exists z (P(x,z) \wedge P(y,z)) \rightarrow \exists u Q(x,y,u)) =$$

- $\forall x \forall y (\neg (\exists z (P(x,z) \wedge P(y,z))) \vee \exists u Q(x,y,u)) =$
- $\forall x \forall y (\forall z (\neg P(x,z) \vee \neg P(y,z))) \vee \exists u Q(x,y,u)) =$
- $\forall x \forall y \forall z \exists u (\neg P(x,z) \vee \neg P(y,z) \vee Q(x,y,u))$
- $\forall x \forall y \forall z (\neg P(x,z) \vee \neg P(y,z) \vee Q(x,y,f(x,y,z)))$
- $\neg P(x,z) \vee \neg P(y,z) \vee Q(x,y,f(x,y,z))$

que é perfeitamente equivalente à fórmula original.

Prova Automática de Teoremas

- ♦ **Resolução: Um Procedimento Completo de Inferência**
 - Seria útil, do ponto de vista computacional, que tivéssemos um procedimento de prova que realizasse, em uma única operação, a variedade de processos envolvidos no raciocínio, com declarações da lógica dos predicados.
 - Este procedimento é a RESOLUÇÃO, que ganha sua eficiência por operar em declarações que foram convertidas à forma clausal, como mostrado anteriormente.
 - A Resolução produz provas por REFUTAÇÃO, ou seja, para provar uma declaração (mostrar que ela é válida), a resolução tenta demonstrar que a negação da declaração produz uma contradição com as declarações conhecidas (não é possível de ser satisfeita).

Prova Automática de Teoremas

- ♦ Resolução: Um Procedimento Completo de Inferência

A BASE DA RESOLUÇÃO

- É um processo iterativo onde, em cada passo, duas cláusulas, denominadas cláusulas paternas, são comparadas (resolvidas), resultando em uma nova cláusula, dela inferida.
- A nova cláusula representa maneiras em que as duas cláusulas paternas interagem entre si.

Prova Automática de Teoremas

- ♦ Resolução: Um Procedimento Completo de Inferência

A BASE DA RESOLUÇÃO

Exemplo:

- Inverno \vee Verão
- \neg Inverno \vee Frio

As duas cláusulas deverão ser verdadeiras (embora pareçam independentes, são realmente conjuntas).

- Agora, observamos que apenas um entre Inverno e \neg Inverno será verdadeiro, em qualquer ponto. Se Inverno for verdadeiro, então Frio também deverá ser, para garantir a verdade da segunda cláusula. Se \neg Inverno for verdadeiro, então também Verão deverá ser, para garantir a verdade da primeira cláusula.

Prova Automática de Teoremas

- ♦ **Resolução: Um Procedimento Completo de Inferência**
 - Assim, dessas duas cláusulas, podemos deduzir que
 - Verão \vee Frio
 - Esta é a dedução feita pelo procedimento de resolução.
 - A resolução opera tirando suas cláusulas que contenham cada uma, o mesmo literal, neste exemplo Inverno.
 - O literal deverá ocorrer na forma positiva numa cláusula e na forma negativa na outra.
 - O resolvente é obtido combinando-se todos os literais das duas cláusulas paternas, exceto aqueles que se cancelam.
 - Se a cláusula produzida for vazia, então foi encontrada uma CONTRADIÇÃO, o que valida a fórmula.

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA PROPOSICIONAL
 - Na Lógica Proposicional, o procedimento para produzir uma prova pela resolução da proposição S , com relação a um conjunto de axiomas F , é o seguinte:
 1. Converter todas as proposições de F em cláusulas.
 2. Negar S e converter o resultado em cláusulas.
Acrescente-as ao conjunto de cláusulas obtidas no passo 1.

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA PROPOSICIONAL
 3. Repetir até que seja encontrada uma contradição ou não se possa fazer progresso:
 - 3.1. Escolher duas cláusulas, que serão chamadas cláusulas pais.
 - 3.2. Resolva-as. A cláusula resultante, denominada resolvente, será a disjunção de todos os literais de ambas as cláusulas pais, com a seguinte exceção:

Se houver qualquer par de literais L e $\neg L$, tal que uma das cláusulas pais contenha L e a outra $\neg L$, então elimine tanto L como $\neg L$ do resolvente.
 - 3.3. Se o resolvente for uma cláusula vazia, terá sido encontrada uma contradição. Se não for, acrescente-o ao conjunto de cláusulas disponíveis para o procedimento.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL

EXEMPLO: $P, (P \wedge Q) \rightarrow R, S \vee T \rightarrow Q, T \vdash R$

- Primeiro convertamos os axiomas em cláusulas.

1. P
2. $\neg P \vee \neg Q \vee R$
3. $\neg S \vee Q$
4. $\neg T \vee Q$
5. T
6. $\neg R$

- Começamos então a escolher a par de cláusulas para resolver. Embora qualquer par de cláusulas possa ser resolvido, apenas aqueles pares que contenham literais complementares produzirão um resolvente com possibilidade de produzir uma cláusula vazia.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL

EXEMPLO: $P, (P \wedge Q) \rightarrow R, S \vee T \rightarrow Q, T \mid \vdash R$

- Começamos por resolver com a cláusula $\neg R$, pois ela é uma das cláusulas que deverão estar envolvidas na contradição que estamos tentando encontrar.

1.	P
2.	$\neg P \vee \neg Q \vee R$
3.	$\neg S \vee Q$
4.	$\neg T \vee Q$
5.	T
6.	$\neg R$

7.	$\neg P \vee \neg Q$	(2 e 6)
8.	$\neg Q$	(1 e 7)
9.	$\neg T$	(4 e 8)
10.	VAZIA	(5 e 9)

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DOS PREDICADOS
 - Na Lógica Proposicional é fácil determinar que dois literais não possam ser verdadeiros ao mesmo tempo. (Simplesmente procure L e $\neg L$)
 - Na Lógica dos Predicados este processo de casamento ("matching") é mais complicado. Por exemplo $\text{Homem}(\text{Henry})$ e $\neg \text{Homem}(\text{Henry})$ é uma contradição, enquanto que $\text{Homem}(\text{Henry})$ e $\neg \text{Homem}(\text{Spot})$ não o é.
 - Assim, para determinar contradições, precisamos de um procedimento de matching que compare dois literais e descubra se existe um conjunto de substituições que os torne idênticos.
 - O ALGORITMO DE UNIFICAÇÃO é um procedimento recursivo direto que faz exatamente isto.

Prova Automática de Teoremas

♦ O ALGORITMO DE UNIFICAÇÃO

- Para apresentar a unificação, consideramos as fórmulas como lista em que o primeiro elemento é o nome do predicado e os elementos restantes são os argumentos.
 - TentarAssassinar (Marco Cesar)
 - TentarAssassinar (Marco (Soberanode (Roma)))
- Para tentar unificar dois literais, primeiro conferimos se seus primeiros elementos são iguais. Caso contrário não há meio de serem unificados, independentemente de seus argumentos.
- Se o primeiro casar, podemos continuar com o segundo e assim por diante.
- Constantes, funções e predicados diferentes não podem casar, os idênticos podem. Uma variável pode casar com outra variável, ou com qualquer constante, função ou expressão de predicados.

Prova Automática de Teoremas

- ♦ O ALGORITMO DE UNIFICAÇÃO
UNIFICA (L1, L2)

1. Se L1 ou L2 for um átomo, então faça o seguinte:
 - 1.1. Se L1 e L2 forem idênticos, retornar NIL
 - 1.2. Caso contrário, se L1 for uma variável, faça
 - 1.2.1. Se L1 ocorrer em L2, retornar F;
 - 1.2.2. Caso contrário, retornar (L2/L1)
 - 1.3. De outro modo, se L2 for uma variável, faça
 - 1.3.1. Se L2 ocorrer em L1, retornar F;
 - 1.2.2. Caso contrário, retornar (L1/L2)
 - 1.4. Caso contrário, retornar F.
2. Se comprimento(L1) não for igual a comprimento(L2) retornar F.
3. Designar a SUBST o valor NIL. (ao final do procedimento, SUBST conterá todas as substituições utilizadas para unificar L1 e L2).

Prova Automática de Teoremas

- ♦ O ALGORITMO DE UNIFICAÇÃO
UNIFICA (L1, L2)

4. Para $i=1$ até o número de elementos de L1, faça:

4.1. Chame UNIFICA com o i -ésimo elemento de L1 e o i -ésimo elemento de L2, colocando o resultado em S.

4.2. Se $S = F$, retornar F.

4.3. Se S não for igual a NIL, faça:

4.3.1. Aplicar S tanto ao final de L1 como de L2.

4.3.2. $SUBST := APPEND(S, SUBST)$

4.3.3. Retornar SUBST

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS
 - Duas fórmulas-atômicas são contraditórias se uma delas puder ser unificada com o não da outra. Assim, por exemplo, $\text{Homem}(x)$ e $\neg \text{Homem}(\text{Spot})$ podem ser unificados.
 - Isto corresponde à intuição que diz que não pode ser verdadeiro para todos os x , que $\text{Homem}(x)$ se houver conhecimento de haver algum x , digamos Spot, para o qual $\text{Homem}(x)$ é falso.
 - Na lógica de predicados utilizaremos o algoritmo de unificação para localizar pares de fórmulas-atômicas que se cancelem.

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS
 1. Converter todas as declarações de F em cláusulas.
 2. Negar S e converter o resultado em cláusulas. Acrescentá-las ao conjunto de cláusulas obtidas em 1.
 3. Repetir até que uma contradição seja encontrada, e nenhum progresso possa ser feito, ou até que se tenha gasto um quantidade pré-determinada de esforço:
 - 3.1. Escolher duas cláusulas e chamá-las de cláusulas pais.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS

3.2. Resolvê-las. O resolvente será a disjunção de todos os literais de ambas as cláusulas pais com as substituições apropriadas realizadas, ressaltando-se o seguinte:

3.2.1. Se houver um par de literais $T1$ e $\neg T2$ tal que uma das cláusulas pais contenha $T1$ e a outra contenha $T2$, e ainda se $T1$ e $T2$ forem unificáveis, então nem $T1$ nem $T2$ devem aparecer no resolvente.

3.2.2. Chamaremos $T1$ e $T2$ literais complementares. Utilize a substituição produzida pela unificação para criar o resolvente.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS

3.2. Resolvê-las. O resolvente será a disjunção de todos os literais de ambas as cláusulas pais com as substituições apropriadas realizadas, ressaltando-se o seguinte:

3.2.1. Se houver um par de literais $T1$ e $\neg T2$ tal que uma das cláusulas pais contenha $T1$ e a outra contenha $T2$, e ainda se $T1$ e $T2$ forem unificáveis, então nem $T1$ nem $T2$ devem aparecer no resolvente.

3.2.2. Chamaremos $T1$ e $T2$ literais complementares. Utilize a substituição produzida pela unificação para criar o resolvente.

3.3. Se o resolvente for uma cláusula vazia, então foi encontrada uma contradição. Se não for, acrescente-o ao conjunto de cláusulas disponíveis para o procedimento.

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS
 - Se a escolha de cláusulas a resolver em cada passo for feita de maneira sistemática, o procedimento de resolução encontrará uma contradição, se ela existir.
 - Isto contudo, poderá levar muito tempo.
 - Existem estratégias opcionais para acelerar o processo.
 - Resolver apenas pares de cláusulas que contenham literais complementares, pois somente essas resoluções produzem cláusulas novas mais difíceis de satisfazer que seus pais.
 - Eliminar cláusulas do tipo tautologias e cláusulas que estejam incluídas em outras cláusulas ($P \vee Q$ é incluída por P).
 - Sempre que possível, resolver com uma das cláusulas que estamos tentando refutar ou com uma cláusula gerada por uma resolução com tal cláusula.
 - Sempre que possível, dar preferência a cláusulas com um único literal.

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS

- EXEMPLO:

1. Marco era um homem.
 2. Marco era um pompeiano.
 3. Todos os pompeianos eram romanos.
 4. César era um soberano.
 5. Todos os romanos ou eram leiais a César ou o odiavam.
 6. Cada um de nós é leal a alguém.
 7. As pessoas tentam assassinar soberanos a quem não sejam leiais.
 8. Marco tentou assassinar César.
- Logo, Marco odiava César?

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS
 - EXEMPLO:
 1. Homem(Marco)
 2. Pompeiano(Marco)
 3. $\forall x \text{ Pompeiano}(x) \rightarrow \text{Romano}(x)$
 4. Soberano(Cesar)
 5. $\forall x \text{ Romano}(x) \rightarrow (\text{LealA}(x, \text{Cesar}) \vee \text{Odiar}(x, \text{Cesar}))$
 6. $\forall x \exists y \text{ LealA}(x, y)$
 7. $\forall x \forall y (\text{Homem}(x) \wedge \text{Soberano}(y) \wedge \text{TentarAssassinar}(x, y) \rightarrow \sim \text{LealA}(x, y))$
 8. TentarAssassinar(Marco, Cesar)
 - Logo, Odiar(Marco, Cesar)

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS
 - EXEMPLO:
 - Primeiro convertemos os axiomas em cláusulas.
 - 1. Homem(Marco)
 - 2. Pompeiano(Marco)
 - 3. \neg Pompeiano(x1) \vee Romano(x1)
 - 4. Soberano(Cesar)
 - 5. \neg Romano(x2) \vee LealA(x2,Cesar) \vee Odiar(x2,Cesar))
 - 6. LealA(x3,f(x3))
 - 7. \neg Homem(x4) \vee \neg Soberano(y1) \vee \neg TentarAssassinar(x4,y1) \vee \neg LealA(x4,y1)
 - 8. TentarAssassinar(Marco,Cesar)
 - 9. \neg Odiar(Marco, Cesar)
- ♦ Começamos então a escolher o par de cláusulas para resolver

Prova Automática de Teoremas

- ♦ RESOLUÇÃO NA LÓGICA DE PREDICADOS

- EXEMPLO:

10. $\neg \text{Romano}(\text{Marco}) \vee \text{LealA}(\text{Marco}, \text{Cesar})$
(SUBST(Marco, x2) em 5 e 9)
11. $\neg \text{Pompeiano}(\text{Marco}) \vee \text{LealA}(\text{Marco}, \text{Cesar})$
(SUBST(Marco, x1 em 3 e 10)
12. $\text{LealA}(\text{Marco}, \text{Cesar})$ (2 e 11)
13. $\neg \text{Homem}(\text{Marco}) \vee \neg \text{Soberano}(\text{Cesar}) \vee$
 $\neg \text{TentarAssassinar}(\text{Marco}, \text{Cesar})$
(SUBST(Marco, x4) e SUBST(Cesar, y1) em 7 e 12)
14. $\neg \text{Soberano}(\text{Cesar}) \vee \neg \text{TentarAssassinar}(\text{Marco}, \text{Cesar})$ (1 e 13)
15. $\neg \text{TentarAssassinar}(\text{Marco}, \text{Cesar})$ (4 e 14)
16. VAZIA (8 e 15)

PROLOG

- ♦ Introdução e Histórico
 - PROgramming in LOGic é fruto de pesquisas na área de Prova Automática de Teoremas.
 - Foi criada por Robert Kowalski (na parte teórica), Maarten van Emden (na demonstração experimental) e Alain Colmerauer (na implementação) por volta de 1970 na Universidade de Marselha, França.
 - O primeiro compilador eficiente foi desenvolvido na Universidade de Edimburgo, Escócia.
 - A linguagem PROLOG também é um provador automático de teoremas, onde a estratégia de cláusulas adotada é a "Selective Linear Resolution for Definite Clauses".
 - É uma linguagem declarativa, onde se diz "o que fazer" para atingir um objetivo, o que leva a um nível mais elevado de abstração na solução dos problemas.

PROLOG

- ♦ Introdução e Histórico

- Segundo Bratko, "pensar a respeito do problema e aprender a programar em PROLOG constitui-se em um desafio intelectual excitante".
- Cada linha de PROLOG corresponde a uma afirmação.
- A variável compreendida na afirmação deve ser entendida como UNIVERSALMENTE quantificada. Assim, a declaração pai_de(X,Y) corresponde a $\forall X \forall Y \text{ pai_de}(X,Y)$.
- PROLOG só admite em suas declarações CLAUSULAS DE HORN.
- CLÁUSULAS DE HORN só admitem um literal positivo.
- Lembre-se que $A \rightarrow B$ pode ser escrita sob a forma $\neg A \vee B$.

PROLOG

- ♦ Introdução e Histórico
 - Sejam $A_i(x_1, x_2, \dots, x_k)$ e $B_i(x_1, x_2, \dots, x_k)$ fórmulas atômicas, então uma regra do tipo:
Se $A_1(x_1, x_2, \dots, x_k)$ e ... e $A_m(x_1, x_2, \dots, x_k)$
então $B_1(x_1, x_2, \dots, x_k)$ e ... e $B_n(x_1, x_2, \dots, x_k)$
pode ser escrita como
 - $A_1(x_1, x_2, \dots, x_k)$ ou ... ou $\neg A_m(x_1, x_2, \dots, x_k)$ e $B_1(x_1, x_2, \dots, x_k)$ ou ... ou $B_n(x_1, x_2, \dots, x_k)$.
 - No entanto, PROLOG só admite declarações do tipo:
 - Se $A_1(x_1, x_2, \dots, x_k)$ e ... e $A_m(x_1, x_2, \dots, x_k)$ então $B_1(x_1, x_2, \dots, x_k)$;
 - $B_1(x_1, x_2, \dots, x_k)$;
 - Se A_1 e ... e A_m então B_1 ; e
 - B_1 .

PROLOG

♦ Introdução e Histórico

- Normalmente, variáveis e constantes são diferenciadas pela primeira letra:
 - Símbolos iniciados por minúscula são constantes; e
 - Símbolos iniciados por letra maiúscula são variáveis.
- O escopo léxico de nomes de variáveis é apenas uma cláusula.
- Isto quer dizer que, por exemplo, se o nome de variável X25 ocorre em duas cláusulas diferentes, então ela está representando duas variáveis diferentes.
- Por outro lado, toda ocorrência de X25 dentro da mesma cláusula quer significar a mesma variável.
- Esta situação é diferente para as constantes: o mesmo nome sempre significa o mesmo objeto ao longo de todo o programa.

PROLOG

- ♦ Introdução e Histórico
 - Em PROLOG, as cláusulas são escritas na forma de regras, com a (única) conclusão no início.
 - $B1(X1, \dots, Xk) :- A1(X1, \dots, Xk), \dots, Am(X1, \dots, Xk).$
 - $B1(X1, \dots, Xk).$
 - $B1 :- A1, \dots, Am.$
 - $B1.$
 - O único literal positivo de uma cláusula (que aparece antes do símbolo $:-$) é chamado cabeça da cláusula.
 - Os literais negativos (que aparecem depois do símbolo $:-$) são chamados corpo da cláusula.

PROLOG

- ♦ Exemplo Introdutório

- ♦ progenitor(maria,josé). % Maria é progenitor de José.
- ♦ progenitor(joão, josé).
- ♦ progenitor(joão, ana).
- ♦ progenitor(josé, júlia).
- ♦ progenitor(josé, iris).
- ♦ progenitor(iris, jorge).
- ♦ masculino(joão). % João é do sexo masculino.
- ♦ masculino(josé).
- ♦ masculino(jorge).
- ♦ feminino(maria). % Maria é do sexo feminino.
- ♦ feminino(ana).
- ♦ feminino(júlia).
- ♦ feminino(íris).

PROLOG

- ♦ Exemplo Introdutório
- ♦ O efeito das entradas anteriores é o armazenamento destas fórmulas atômicas representando fatos em uma base de conhecimentos PROLOG.
- ♦ Se o programa for submetido a um sistema Prolog, este será capaz de responder algumas questões sobre a relação ali representada. Por exemplo: "José é o progenitor de Iris?".
 - ♦ ?-progenitor(josé, íris).
- ♦ Uma outra questão poderia ser: "Ana é um dos progenitores de Jorge?".
 - ♦ ?-progenitor(ana, jorge).

PROLOG

- ♦ Exemplo Introdutório
- ♦ Perguntas mais interessantes podem também ser formuladas, por exemplo: "Quem é progenitor de Iris?"
 - ♦ ?-progenitor(X, iris).
- ♦ Da mesma forma a questão "Quem são os filhos de José?" pode ser formulada com a introdução de uma variável na posição do argumento correspondente ao filhos de José
 - ♦ ?-progenitor(josé, X).
- ♦ Uma questão mais geral para o programa seria: "Quem é progenitor de quem?"
 - ♦ ?-progenitor(X, Y).

PROLOG

- ♦ Exemplo Introdutório
- ♦ Pode-se formular questões ainda mais complicadas ao programa, como "Quem são os avós de Jorge?". Como nosso programa não possui diretamente a relação avô, esta consulta precisa ser dividida em duas etapas. A saber:
 - (1) Quem é progenitor de Jorge? (Por exemplo, Y)
e
 - (2) Quem é progenitor de Y? (Por exemplo, X)
- ♦ Esta consulta em Prolog é escrita como uma seqüência de duas consultas simples, cuja leitura pode ser: "Encontre X e Y tais que X é progenitor de Y e Y é progenitor de Jorge".
 - ♦ ?-progenitor(X, Y), progenitor(Y, jorge).
 - ♦ X=josé Y=íris

PROLOG

- ♦ Pontos Básicos
- ♦ Uma relação como progenitor pode ser facilmente definida em Prolog estabelecendo-se as tuplas de objetos que satisfazem a relação;
- ♦ O usuário pode facilmente consultar o sistema Prolog sobre as relações definidas em seu programa;
- ♦ Um programa Prolog é constituído de cláusulas, cada uma das quais é encerrada por um ponto (.);
- ♦ Os argumentos das relações podem ser objetos concretos (como júlia e íris) ou objetos genéricos (como X e Y). Objetos concretos em um programa são denominados átomos, enquanto que os objetos genéricos são denominados variáveis;

PROLOG

- ♦ Pontos Básicos
- ♦ Consultas ao sistema são constituídas por um ou mais objetivos, cuja seqüência denota a sua conjunção;
- ♦ Uma resposta a uma consulta pode ser positiva ou negativa, dependendo se o objetivo correspondente foi alcançado ou não. No primeiro caso dizemos que a consulta foi bem-sucedida e, no segundo, que a consulta falhou;
- ♦ Se várias respostas satisfizerem a uma consulta, então o sistema Prolog irá fornecer tantas quantas forem desejadas pelo usuário.

PROLOG

- ♦ Exemplo Introdutório

- A capacidade do PROLOG não se limita à busca em uma base de conhecimentos; é possível armazenar regras.
- As regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira.
 - ? -mae(X,Y) :- progenitor(X,Y), feminino(X).
 - ? -pai(X,Y) :- progenitor(X,Y), masculino(X).
 - ? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
- Com estas definições podemos obter os seguintes resultados:
 - ? -mae(X,josé).
X = maria;

 - ? -pai(X,íris).
X = josé;

PROLOG

- ♦ Pontos Básicos
- ♦ Programas Prolog podem ser ampliados pela simples adição de novas cláusulas;
- ♦ As cláusulas Prolog podem ser de três tipos distintos: fatos, regras e consultas;
- ♦ Os fatos declaram coisas que são incondicionalmente verdadeiras;
- ♦ As regras declaram coisas que podem ser ou não verdadeiras, dependendo da satisfação das condições dadas;
- ♦ Por meio de consultas podemos interrogar o programa acerca de que coisas são verdadeiras;

PROLOG

- ♦ Pontos Básicos
- ♦ As cláusulas Prolog são constituídas por uma cabeça e um corpo. O corpo é uma lista de objetivos separados por vírgulas que devem ser interpretadas como conjunções;
- ♦ Fatos são cláusulas que só possuem cabeça, enquanto que as consultas só possuem corpo e as regras possuem cabeça e corpo;
- ♦ Ao longo de uma computação, uma variável pode ser substituída por outro objeto. Dizemos então que a variável está instanciada;
- ♦ As variáveis são assumidas como universalmente quantificadas nas regras e nos fatos e existencialmente quantificadas nas consultas

PROLOG

- ♦ Exemplo Introdutório

- Além disso, as definições de regras podem ser recursivas, isto é, uma cláusula de definição de um predicado pode conter este predicado em seu corpo:

? -antepassado(X,Z) :- progenitor(X,Z).

? -antepassado(X,Z) :- progenitor(X,Y),
antepassado(Y,Z).

- Com estas definições podemos obter os seguintes resultados:

? -antepassado(X,jorge).

X = íris;

X = maria;

X = joão;

X = josé;

PROLOG

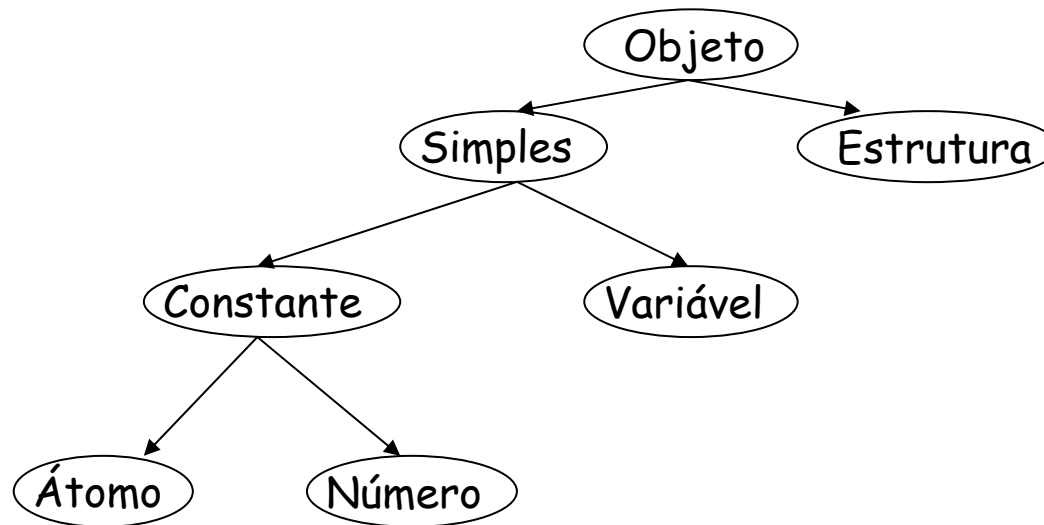
- ♦ Exemplo Introdutório

- Usa-se o "_" (underscore) para indicar a irrelevância de um objeto
 - ? -aniversario(maria,data(25,janeiro,1979)).
 - ? -aniversario(joao,data(5,janeiro,1956)).
 - ? -signo(Pessoa,aquario) :- aniversario(Pessoa,data(Dia,janeiro,_)), Dia >=20.
 - Com estas definições podemos obter os seguintes resultados:
 - ? -signo(Pessoa,aquario).
- Pessoa = maria;
no
- Usa-se a "," como operador de conjunção e usa-se o ";" como operador de disjunção (cláusulas começando com o mesmo predicado também indicam a disjunção)
 - ? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
 - ? -amiga(X) :- (X = maria ; X = joana).

PROLOG

- ♦ Sintaxe

- O sistema reconhece o tipo de um objeto no programa por meio de sua forma sintática.
- Isto é possível porque o PROLOG especifica formas diferentes para cada tipo de objeto.



PROLOG

- ♦ Sintaxe

- Átomos e Números

- No exemplo introdutório viu-se informalmente alguns exemplos de átomos e variáveis. O alabeto básico adotado consiste dos seguintes símbolos:
 - Pontuação: () . ' "
 - Conectivos: , (conjunção)
; (disjunção)
:- (implicação)
 - Letras: a, b, c, ..., z, A, B, C, ..., Z
 - Dígitos: 0, 1, 2, ..., 9
 - Especiais: + - * / < > = ...

PROLOG

♦ Sintaxe

- Variáveis

- Variáveis PROLOG são cadeias de letras, dígitos e do caracter sublinhado (_), devendo iniciar com este ou com uma letra maiúscula.

- Estruturas

- Estruturas são objetos que possuem vários componentes.
- Os próprios componentes, por sua vez, podem também ser estruturas.
- Para combinar os elementos em uma estrutura é necessário um functor. Um functor é um símbolo funcional (nome de função) que permite agrupar diversos objetos em um único objeto estruturado.
- `data(13, outubro, 1993)` - dois inteiros e um átomo.
- `data(Dia, marco, 1996)` - um dia qualquer de marco.

PROLOG

♦ Sintaxe

- Sintaticamente todos os objetos em PROLOG são denominados termos.
- O conjunto de termos PROLOG é o menor conjunto que satisfaz às seguintes condições:
 - Toda constante é um termo;
 - Toda variável é um termo;
 - Se t_1, t_2, \dots, t_n são termos e f é um átomo, então $f(t_1, t_2, \dots, t_n)$ também é um termo, onde o átomo f desempenha o papel de um símbolo funcional n -ário. Diz ainda que a expressão $f(t_1, t_2, \dots, t_n)$ é um termo funcional PROLOG.

PROLOG

♦ Consultas em Prolog

- Uma consulta em Prolog é sempre uma seqüência composta por um ou mais objetivos. Para obter a resposta, o sistema Prolog tenta satisfazer todos os objetivos que compõem a consulta, interpretando-os como uma conjunção. Satisfazer um objetivo significa demonstrar que esse objetivo é verdadeiro, assumindo que as relações que o implicam são verdadeiras no contexto do programa. Se a questão também contém variáveis, o sistema Prolog deverá encontrar ainda os objetos particulares que, atribuídos às variáveis, satisfazem a todos os sub-objetivos propostos na consulta. A particular instanciiação das variáveis com os objetos que tornam o objetivo verdadeiro é então apresentada ao usuário. Se não for possível encontrar, no contexto do programa, nenhuma instanciiação comum de suas variáveis que permita derivar algum dos sub-objetivos propostos então a resposta será "não".

PROLOG

- ♦ Unificação

- É a operação mais importante entre dois termos PROLOG.
- Dados dois termos, diz-se que eles se unificam se:
 - Eles são idênticos, ou
 - As variáveis de ambos os termos podem ser instanciadas com objetos de maneira que, após a substituição das variáveis por estes objetos, os termos se tornam idênticos.
- Exemplo
 - os termos `data(D,M,1994)` e `data(X,marco,A)` unificam. Uma instância que torna os dois termos idênticos é:
 - D é instanciada com X;
 - M é instanciada com marco;
 - A é instanciada com 1994.
 - Por outro lado, os termos `data(D,M,1994)` e `data(X,Y,94)` não unificam, assim como não unificam `data(X,Y,Z)` e `ponto(X,Y,Z)`.

PROLOG

♦ Unificação

- Se os termos não unificam dizemos, dizemos que o processo FALHA.
- Se eles unificam, então o processo é bem-sucedido.
- As regras gerais que determinam se dois termos S e T unificam são:
 - Se S e T são constantes, então S e T unificam somente se ambos representam o mesmo objeto;
 - Se S é uma VARIÁVEL E t É QUALQUER COISA, ENTÃO s E t UNIFICAM COM s INSTANCIADA EM t . Inversamente, se T é uma variável, então T é instanciada em S .
 - Se S e T são estruturas, unificam somente se:
 - S e T tem o mesmo functor principal, e
 - todos os seus componentes correspondentes também unificam. A instanciação resultante é determinada pela unificação dos componentes.

PROLOG

- ♦ Consultas em Prolog

- antepassado(X, Z) :- % X é antepassado de Z se
 progenitor(X, Z).% X é progenitor de Z. [pr1]
- antepassado(X, Z) % X é antepassado de Z se
 progenitor(X, Y), % X é progenitor de Y e
 antepassado(Y, Z).% Y é antepassado de Z. [pr2]

PROLOG

- ♦ Consultas em Prolog
- ♦ Um exemplo mais complexo:
?-antepassado(joão, íris).
- ♦ Sabe-se que progenitor(josé, íris) é um fato. Usando esse fato e a regra [pr1], podemos concluir antepassado(josé, íris). Este é um fato derivado. Não pode ser encontrado explícito no programa, mas pode ser derivado a partir dos fatos e regras ali presentes. Ou seja:
- ♦ "de progenitor(josé, íris) segue, pela regra [pr1] que antepassado(josé, íris)".
- ♦ Além disso sabemos que progenitor(joão, josé) é fato. Usando este fato e o fato derivado, antepassado(josé, íris), podemos concluir, pela regra [pr2], que o objetivo proposto, antepassado(joão, íris) é verdadeiro.

PROLOG

- ♦ Consultas em Prolog
- ♦ Mostrou-se assim o que pode ser uma seqüência de passos de inferência usada para satisfazer um objetivo. Tal seqüência denomina-se seqüência de prova. A extração de uma seqüência de prova do contexto formado por um programa e uma consulta é obtida pelo sistema na ordem inversa da empregada anteriormente.

PROLOG

- ♦ Consultas em Prolog
- ♦ Ao invés de iniciar a inferência a partir dos fatos, o Prolog começa com os objetivos e , usando as regras, substitui os objetivos correntes por novos objetivos até que estes se tornem fatos.
- ♦ Assim, para saber se João é antepassado de Iris, o sistema tenta encontrar uma cláusula no programa a partir da qual o objetivo seja consequência imediata. Obviamente, as únicas cláusulas relevantes para essa finalidade são [pr1] e [pr2], que são sobre a relação antepassado, porque são as únicas cujas cabeças podem ser unificadas com o objetivo formulado.
- ♦ Tais cláusulas representam dois caminhos alternativos que o sistema pode seguir. Inicialmente o Prolog irá tentar a que aparece em primeiro lugar no programa:

PROLOG

- ♦ Consultas em Prolog
- ♦ `antepassado(X, Z) :- progenitor(X, Z).`
- ♦ Uma vez que o objetivo é `antepassado(joão, íris)`, as variáveis na regra devem ser instanciadas por `X=joão` e `Y=íris`. O objetivo inicial, `antepassado(joão, íris)` é então substituído por um novo objetivo:
`progenitor(joão, íris)`
- ♦ Não há, entretanto, nenhuma cláusula no programa cuja cabeça possa ser unificada com `progenitor(joão, íris)`, logo este objetivo falha. Então o Prolog retorna ao objetivo original (backtracking) para tentar um caminho alternativo que permita derivar o objetivo `antepassado(joão, íris)`. A regra [pr2] é então tentada:
- ♦ `antepassado(X, Z) :-
 progenitor(X, Y),
 antepassado(Y, Z).`

PROLOG

- ♦ Consultas em Prolog
- ♦ Como anteriormente, as variáveis X e Z são instanciadas para João e Íris, respectivamente. A variável Y, entretanto, não está instanciada ainda. O objetivo original, antepassado(João, Íris) é então substituído por dois novos objetivos derivados por meio da regra [pr2]:
- ♦ progenitor(João, Y), antepassado(Y, Íris).
- ♦ Encontrando-se agora face a dois objetivos, o sistema tenta satisfazê-los na ordem em que estão formulados. O primeiro deles é fácil: progenitor(João, Y) pode ser unificado com dois fatos do programa: progenitor(João, José) e progenitor(João, Ana). Mais uma vez, o caminho a ser tentado deve corresponder à ordem em que os fatos estão escritos no programa. A variável Y é então instanciada com José nos dois objetivos acima, ficando o primeiro deles imediatamente satisfeito.

PROLOG

- ♦ Consultas em Prolog
- ♦ O objetivo remanescente é então:
- ♦ antepassado(josé, íris).
- ♦ Para satisfazer tal objetivo, a regra [pr1] é mais uma vez empregada.
- ♦ Essa segunda aplicação de [pr1], entretanto, nada tem a ver com a sua utilização anterior, isto é, o sistema Prolog usa um novo conjunto de variáveis na regra cada vez que esta é aplicada.

PROLOG

- ♦ Consultas em Prolog
- ♦ A cabeça da regra deve então ser unificada com o nosso objetivo corrente, que é antepassado(josé, íris). A instanciação de X' e Y' fica: $X' = \text{josé}$ e $Y' = \text{íris}$ e o objetivo corrente é substituído por:
- ♦ `progenitor(josé, íris)`
- ♦ Esse objetivo é imediatamente satisfeito, porque aparece no programa como um fato. O sistema encontrou então um caminho que lhe permite provar, no contexto oferecido pelo programa dado, o objetivo originalmente formulado, e portanto responde "sim".

PROLOG

- ♦ Semântica

- PROLOG tem três semânticas: a DECLARATIVA, a PROCEDURAL e a OPERACIONAL.
- A semântica DECLARATIVA é aquela em que se escreve e lê o programa PROLOG e que é uma representação do que se conhece da definição do problema a resolver.
- A semântica declarativa pode ser interpretada de três modos distintos:
 - Para resolver um problema dá-se uma nova cláusula, a pergunta, e o PROLOG tenta verificar se esta cláusula é compatível com o mundo definido;
 - Considera-se que os literais na cabeça e cauda de cada cláusula são objetivos a serem atingidos. Uma pergunta tem resposta afirmativa se o objetivo que ela define é satisfeito usando as regras do programa;
 - Olha as cláusulas como regras de uma gramática, onde cada regra de PROLOG corresponde a uma regra da gramática.

PROLOG

- ♦ Semântica

- Exemplo

- Seja $P :- Q, R$

onde P , Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura declarativa destas cláusulas são:

- P é verdadeira se Q e R são verdadeiras, e
 - De Q e R , segue P .
 - A semântica declarativa determina se um dado objetivo é verdadeiro e, se for, para que valores de variáveis isto se verifica.
 - Assim, dado um programa e um objetivo G , o significado declarativo nos diz que:
 - Um objetivo G é verdadeiro (isto é, é satisfatível ou segue logicamente do programa) se e somente se há uma cláusula C no programa e uma instância I de C tal que:
 - A cabeça de I é idêntica a G , e
 - Todos os objetivos no corpo de I são verdadeiros.

PROLOG

- ♦ Semântica

- A semântica PROCEDURAL define não apenas o relacionamento lógico existente entre a cabeça e o corpo da cláusula, como também exige a existência de uma ordem na qual os objetivos serão processados.

- Exemplo

- Seja $P :- Q, R$

onde P , Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura procedural destas cláusulas são:

- Para solucionar o problema P
primeiro solucione o subproblema Q
e depois solucione o subproblema R .
 - Para satisfazer P , primeiro satisfaça Q e depois R .

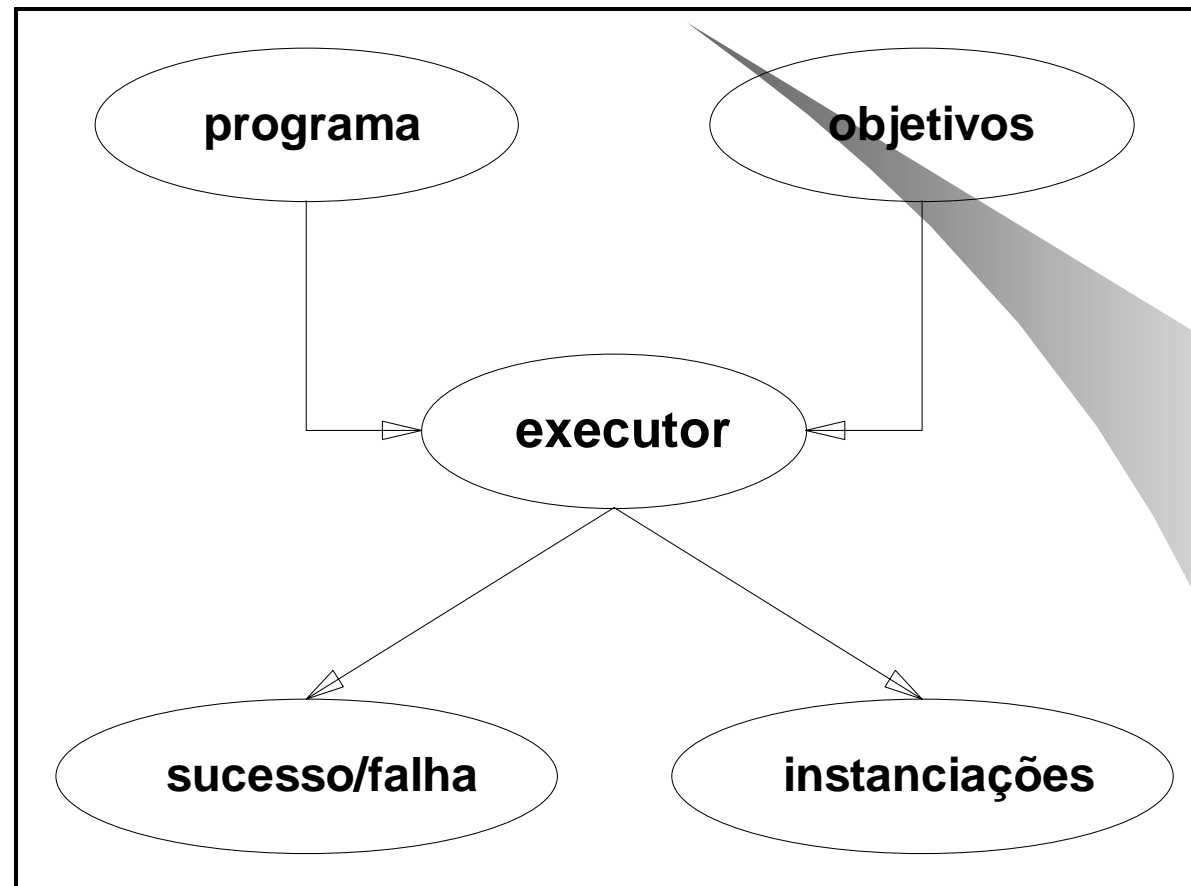
PROLOG

- ♦ Semântica

- A semântica OPERACIONAL define como PROLOG responde a uma pergunta.
- Seria ótimo se não fosse necessário conhecer esta semântica, pois neste caso, PROLOG seria realmente uma implementação do paradigma de programação em lógica. Entretanto, este não é o caso.
- PROLOG pesquisa se a pergunta é verdadeira ou falsa construindo a árvore de possibilidades para trás e faz a busca em profundidade. Se a árvore a percorrer é muito grande, o tempo pode se tornar proibitivo e é conveniente restringir o espaço de busca o mais possível, o que só é possível sabendo como o PROLOG vai visitar os nós da árvore.
- O modo como PROLOG vai visitar os nós da árvore varia se as declarações são apresentadas em ordem diferente. Consequentemente, PROLOG não é realmente uma linguagem declarativa.

PROLOG

- ♦ Semântica



PROLOG

- ♦ Semântica

- Suas entradas e saídas são:

- entrada: um programa e uma lista de objetivos;
 - saída: um indicador de sucesso/falha e instâncias de variáveis.

- O significado dos resultados de saída do *executor* é o seguinte:

- O indicador de *sucesso/falha* tem o valor "sim" se os objetivos forem todos satisfeitos e "não" em caso contrário;
 - As *instâncias* são produzidas somente no caso de conclusão bem-sucedida e correspondem aos valores das variáveis que satisfazem os objetivos.

PROLOG

- ♦ Semântica (Resumo)

- A interpretação declarativa de programas escritos em Prolog puro não depende da ordem das cláusulas nem da ordem dos objetivos dentro das cláusulas;
- A interpretação procedimental depende da ordem dos objetivos e cláusulas. Assim a ordem pode afetar a eficiência de um programa. Uma ordenação inadequada pode mesmo conduzir a chamadas recursivas infinitas;
- A semântica operacional representa um procedimento para satisfazer a lista de objetivos no contexto de um dado programa. A saída desse procedimento é o valor-verdade da lista de objetivos com a respectiva instanciação de suas variáveis. O procedimento permite o retorno automático (backtracking) para o exame de novas alternativas;

PROLOG

- ♦ Backtracking

- Na execução dos programas Prolog, a evolução da busca por soluções assume a forma de uma árvore - denominada "árvore de pesquisa" ou "search tree" - que é percorrida sistematicamente de cima para baixo (top-down) e da esquerda para direita, segundo o método denominado "depth-first search" ou "pesquisa primeiro em profundidade".

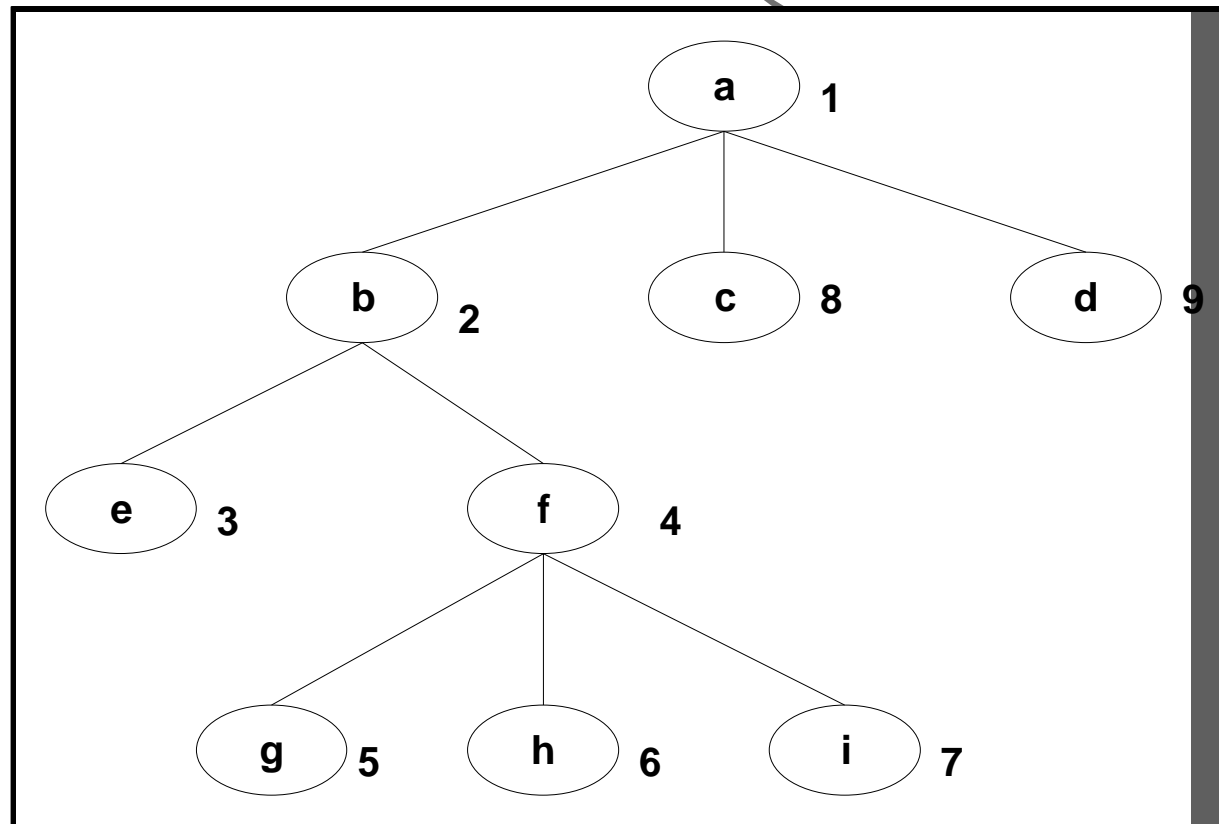
Exemplo

- Sejam a,b,c,etc... termos PROLOG

```
a :- b.  
a :- c.  
a :- d.  
b :- e.  
b :- f.  
f :- g.  
f :- h.  
f :- i.  
d.
```

PROLOG

- ♦ Backtracking
 - Ordem de visita aos nodos da árvore



a, b, e, (b), f, g, (f), h, (f), i, (f), (b), (a), c, (a), d
onde o caminho em backtracking é representado entre parênteses

PROLOG

- ♦ Backtracking

- Como foi visto, os objetivos em um programa Prolog podem ser bem-sucedidos ou falhar.
- Para um objetivo ser bem-sucedido ele deve ser unificado com a cabeça de uma cláusula do programa e todos os objetivos no corpo desta cláusula devem também ser bem-sucedidos. Se tais condições não ocorrerem, então o objetivo falha.
- Quando um objetivo falha, em um nodo terminal da árvore de pesquisa, o sistema Prolog aciona o mecanismo de backtracking, retornando pelo mesmo caminho percorrido, na tentativa de encontrar soluções alternativas.
- Ao voltar pelo caminho já percorrido, todo o trabalho executado é desfeito.

PROLOG

♦ Backtracking

Exemplo 2

- gosta(joão, jazz).
 - gosta(joão, renata).
 - gosta(joão, lasanha).
 - gosta(renata, joão).
 - gosta(renata, lasanha).
- queremos saber do que ambos, joão e renata, gostam. Isto pode ser formulado pelos objetivos:
- gosta(joão, X), gosta(renata, X).
1. Encontra que joão gosta de jazz
 2. Instancia X com "jazz"
 3. Tenta satisfazer o segundo objetivo, determinando se "renata gosta de jazz"
 4. Falha, porque não consegue determinar se renata gosta de jazz
 5. Realiza um backtracking na repetição da tentativa de satisfazer `gosta(joão, X)`, esquecendo o valor "jazz"

PROLOG

- ♦ Backtracking

Exemplo 2

6. Encontra que João gosta de Renata
7. Instancia X com "Renata"
8. Tenta satisfazer o segundo objetivo determinando se "Renata gosta de Renata"
9. Falha porque não consegue demonstrar que Renata gosta de Renata
10. Realiza um backtracking, mais uma vez tentando satisfazer `gosta(João, X)`, esquecendo o valor "Renata"
11. Encontra que João gosta de Lasanha
12. Instancia X com "Lasanha"
13. Encontra que "Renata gosta de Lasanha"
14. É bem-sucedido, com X instanciado com "Lasanha"

PROLOG

- ♦ Impurezas de PROLOG

- O backtracking automático é uma ferramenta muito poderosa e a sua exploração é de grande utilidade para o programador. Às vezes, entretanto, ele pode se transformar em fonte de ineficiência. A seguir se introduzirá um mecanismo para "podar" a árvore de pesquisa, evitando o backtracking quando este for indesejável.
- Para aumentar a eficiência no percurso da árvore de busca da solução do problema usam-se essencialmente dois operadores: CORTE ("CUT" representado por !) e FALHA ("FAIL").
- Seu uso deve ser considerado pelas seguintes razões:
 - O programa irá executar mais rapidamente, porque não irá desperdiçar tempo tentando satisfazer objetivos que não irão contribuir para a solução desejada.
 - (ii) Também a memória será economizada, uma vez que determinados pontos de backtracking não necessitam ser armazenados para exame posterior.

PROLOG

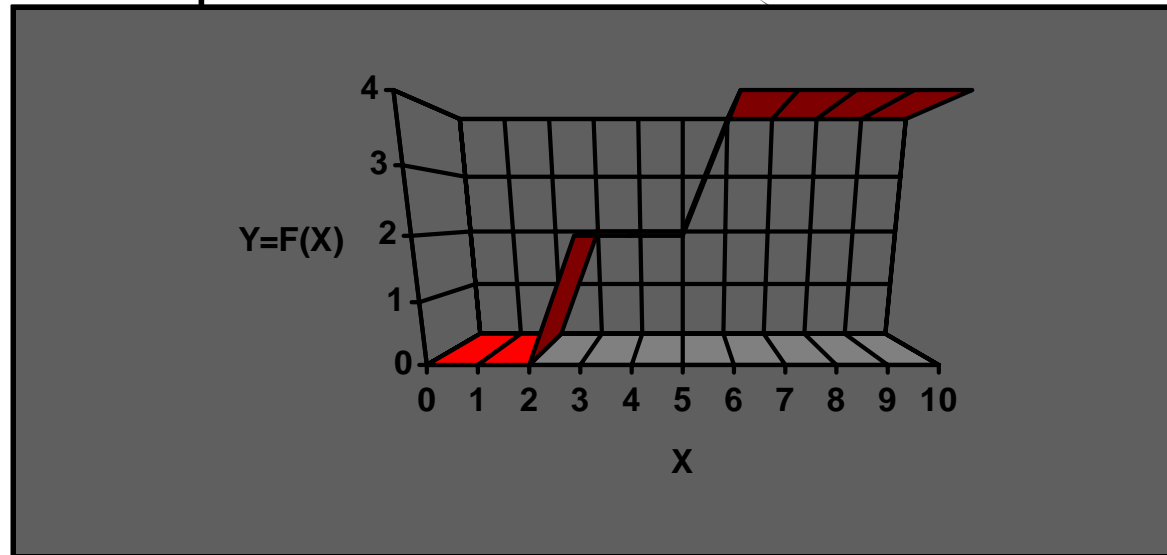
♦ CUT

- Algumas das principais aplicações do cut são as seguintes:
 - Unificação de padrões, de forma que quando um padrão é encontrado os outros padrões possíveis são descartados
 - Na implementação da negação como regra de falha
 - Para eliminar da árvore de pesquisa soluções alternativas quando uma só é suficiente
 - Para encerrar a pesquisa quando a continuação iria conduzir a uma pesquisa infinita, etc

PROLOG

- ♦ CUT

Exemplo



(1) Se $X < 3$, então $Y = 0$

(2) Se $3 \leq X$ e $X < 6$, então $Y = 2$

(3) Se $6 \leq X$, então $Y = 4$

que podem ser escritas em Prolog como uma relação binária $f(X, Y)$,
como se segue:

$f(X, 0) \text{ :- } X < 3.$

$f(X, 2) \text{ :- } 3 \leq X, X < 6.$

$f(X, 4) \text{ :- } 6 \leq X.$

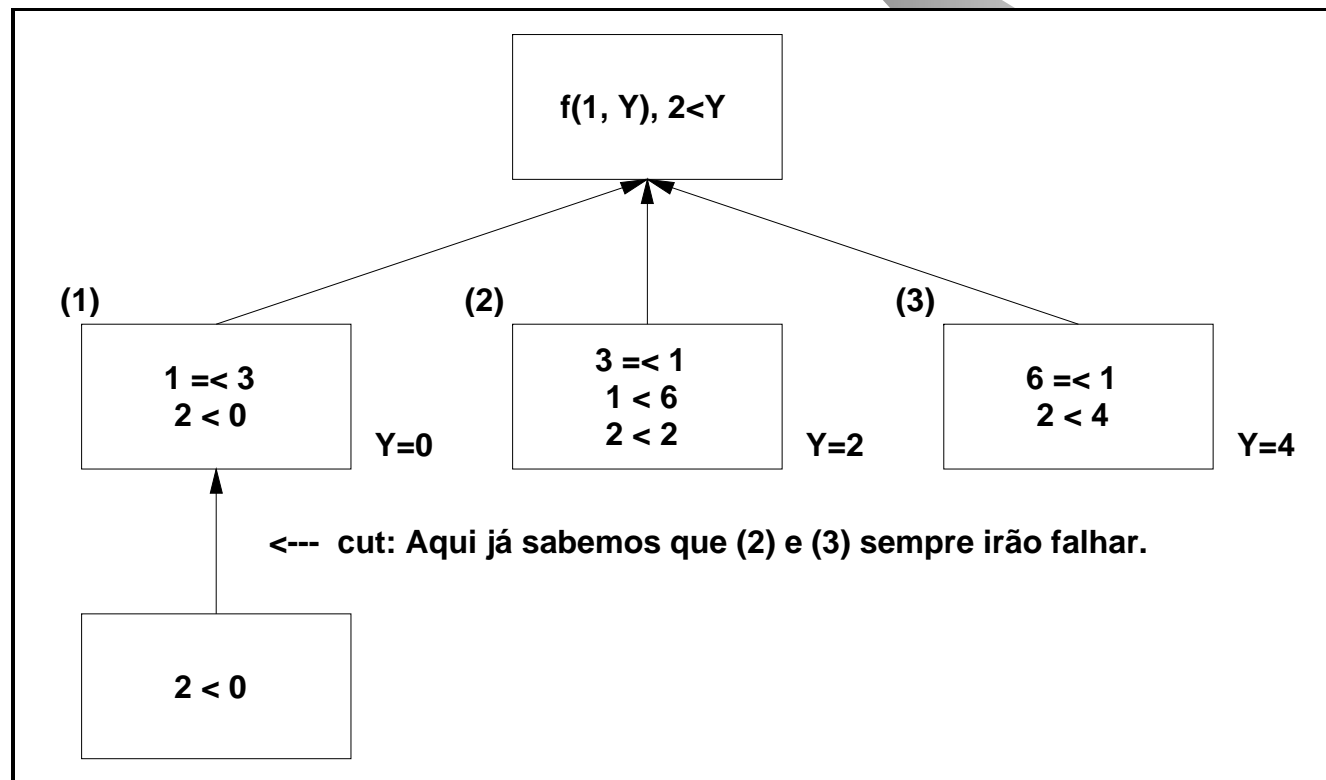
PROLOG

- ♦ CUT

Exemplo

- Vamos analisar o que ocorre quando a seguinte questão é formulada:

- ♦ $?-f(1, Y), 2 < Y$



PROLOG

- ♦ CUT

Exemplo

- O programa do exemplo, reescrito com cuts assume o seguinte aspecto:

$f(X, 0) \text{ :- } X < 3, !.$

$f(X, 2) \text{ } 3 \leq X, X < 6, !.$

$f(X, 4) \text{ } 6 \leq X.$

- ♦ Aqui o símbolo "!" evita o backtracking nos pontos em que aparece no programa.

PROLOG

- ♦ FALHA (FAIL)

- Negação por Falha
- "Maria gosta de todos os animais, menos de cobras". Como podemos dizer isto em Prolog? É fácil expressar uma parte dessa declaração: Maria gosta de X se X é um animal, isto é:

- `gosta(maria, X) :- animal(X).`

mas é necessário ainda excluir as cobras. Isto pode ser conseguido empregando-se uma formulação diferente:

- Se X é uma cobra,
então não é verdade que maria gosta de X
senão se X é um animal, então maria gosta de X.

PROLOG

♦ FALHA (FAIL)

- Podemos dizer que alguma coisa não é verdadeira em Prolog por meio de um predicado pré-definido especial, "fail", que sempre falha, forçando o objetivo pai a falhar. A formulação acima pode ser dada em Prolog com o uso do fail da seguinte maneira:

- `gosta(maria, X) :- cobra(X), !, fail.`
- `gosta(maria, X) :- animal(X).`

- Aqui a primeira regra se encarrega das cobras. Se X é uma cobra, então o cut evita o backtracking (assim excluindo a segunda regra) e o fail irá ocasionar a falha da cláusula. As duas regras podem ser escritas de modo mais compacto como uma única cláusula, por meio do uso do conetivo ";;":

- `gosta(maria, X) :- cobra(X), !, fail; animal(X).`

Raciocínio Inferencial

- ♦ As principais características do motor de inferência disponível em shells para sistemas especialistas dizem respeito às seguintes funcionalidades:
 - Método de Raciocínio,
 - Estratégia de Busca,
 - Resolução de Conflito e
 - Representação de Incerteza e Imprecisão.
- ♦ Estas características compõem o Mecanismo de Raciocínio do Sistema Especialista.

Raciocínio Inferencial

- ♦ Definição
 - É aquele que se baseia em regras válidas de inferência. Pode ser aplicado em sistemas que adotam a representação do conhecimento sob a forma de regras de produção (os sistemas de produção) ou sob a forma de lógica.
- ♦ Modo de Raciocínio
 - Existem basicamente dois modos de raciocínio:
 - Raciocínio para a Frente ou Forward Chaining, e
 - Raciocínio para Trás ou Backward Chaining.

Raciocínio Inferencial

- ♦ Raciocínio para a Frente
 - Consiste em começar com fatos encontrados em uma base de conhecimentos e manipulá-los com as regras (de inferência) tentando chegar a uma conclusão.
 - É também chamado de raciocínio dirigido por dados ("data driven").
 - A parte esquerda da regra (os antecedentes ou estado) é comparada com a descrição da situação atual contida na memória de trabalho. As regras que satisfazem a esta descrição tem a sua parte direita (ação ou novo estado) executada, o que, em geral, significa a introdução d novos fatos na memória de trabalho.

Raciocínio Inferencial

- ♦ Raciocínio para a Frente

- Exemplo:

REGRAS

- 1. $A \Rightarrow C$
- 2. $B \Rightarrow D$
- 3. $C \wedge D \Rightarrow E$

MEMÓRIA DE TRABALHO

- A e B

- C por 1
- D por 2
- E por 3

Raciocínio Inferencial

- ♦ Raciocínio para Trás
 - Começa usando a conclusão e tenta provar se são verdadeiras ou falsas as premissas.
 - É também chamado de raciocínio dirigido por objetivos ("goal driven").
 - O comportamento do sistema é controlado por uma lista de objetivos. Um objetivo por ser satisfeito diretamente por um elemento da memória de trabalho, ou podem existir regras que permitam inferir algum dos objetivos correntes, isto é, que contenham uma descrição deste objetivo em suas partes direitas.
 - As regras que satisfazem esta condição têm as instâncias correspondentes às suas partes esquerdas adicionadas à lista de objetivos correntes.

Raciocínio Inferencial

- ♦ Raciocínio para Trás
 - Caso uma destas regras tenha todas as suas condições satisfeitas diretamente pela memória de trabalho, o objetivo em sua parte direita é também adicionado à memória de trabalho.
 - Um objetivo que não possa ser satisfeito diretamente pela memória de trabalho, nem inferido através de uma regra, é abandonado.
 - Quando o objetivo inicial é satisfeito, ou não há mais objetivos, o processamento termina.

Raciocínio Inferencial

- ♦ Raciocínio para Trás
 - O encadeamento para trás destaca-se em problemas nos quais há um grande número de conclusões que podem ser atingidas, mas o número de meios pelos quais elas podem ser alcançadas não é grande (um sistema de regras de alto grau de fan out), e em problemas nos quais não se pode reunir um número aceitável de fatos antes de iniciar-se a busca por respostas.
 - O encadeamento para trás também é mais intuitivo para o desenvolvedor, pois é fundamentada na recursão, um meio elegante e racional de programação, para onde a própria Programação em Lógica se direccionou.

Raciocínio Inferencial

- ♦ Raciocínio para Trás

- Exemplo:

REGRAS

- 1. $A \Rightarrow C$
- 2. $B \Rightarrow D$
- 3. $C \wedge D \Rightarrow E$

MEMÓRIA DE TRABALHO

- A e B

LISTA DE OBJETIVOS

- E

- C e D por 3
- C por 1 e A na M.T.
- D por 2 e B na M.T.

Raciocínio Inferencial

- ♦ O tipo de encadeamento normalmente é definido de acordo com o tipo de problema a ser resolvido.
- ♦ Problemas de planejamento, projeto e classificação tipicamente utilizam encadeamento para a frente,
- ♦ Problemas de diagnóstico, onde existem apenas algumas conclusões possíveis mas um grande número de estados iniciais, utilizam encadeamento para trás.

Raciocínio Inferencial

- ♦ Uma característica importante do modo de raciocínio se refere à monotonicidade ou não do método de inferência.
- ♦ Sistemas monotônicos não permitem a revisão de fatos,
- ♦ Sistemas não monotônicos permitem a alteração dinâmica dos fatos e, portanto, quando um fato verdadeiro torna-se falso, todas as conclusões baseadas neste fato também devem tornar-se falsas.

Métodos de Solução de Problemas

- ♦ Sistemas Especialistas focam um conjunto reduzido de problemas.
- ♦ O conhecimento é tanto teórico quanto prático.
- ♦ Devido a natureza heurística, os Sistemas Especialistas geralmente:
 - Suportam inspeção de seus processos de raciocínio;
 - Permite fácil modificação de habilidades a base de conhecimento (adição e exclusão);
 - Raciocinam heuristicamente.
- ♦ A facilidade de modificação da base de conhecimento é um fator muito importante na produção de um programa bem-sucedido.

Métodos de Solução de Problemas

- ♦ Waterman (1986) classifica os problemas para S.E.:
 - **Interpretação:** conclusões de alto nível de dados brutos
 - **Predição:** projetar conseqüências prováveis
 - **Diagnose:** determinar causas de mau funcionamento com base em sintomas observáveis
 - **Projeto:** encontrar configurações de componentes que alcance objetivos de desempenho
 - **Planejamento:** estabelecer seqüência de ações que alcançarão um conjunto de objetivos
 - **Monitoramento:** comparar comportamentos observados com esperado
 - **Instrução:** dar assistência ao processo de educação
 - **Controle:** governar o comportamento de um ambiente complexo

S.E. Baseados em Regras

- ♦ Sistema de Produção: Raciocínio Guiado por Objetivo
 - expressão-objetivo é colocada inicialmente na memória de trabalho
 - o sistema tenta casar as conclusões das regras com o objetivo, selecionando uma regra e colocando as suas premissas na memória de trabalho
 - Corresponde a uma decomposição do problema em subproblemas
 - O sistema trabalha retroativamente a partir do objetivo inicial até que todos sejam provados verdadeiros.

S.E. Baseados em Regras

- ♦ Sistema de Produção: Raciocínio Guiado por Objetivo

- Exemplo:

- Regra 1: se

- o motor está recebendo combustível e
 - o motor tentar pegar
 - então o problema é vela

- Regra 2: se

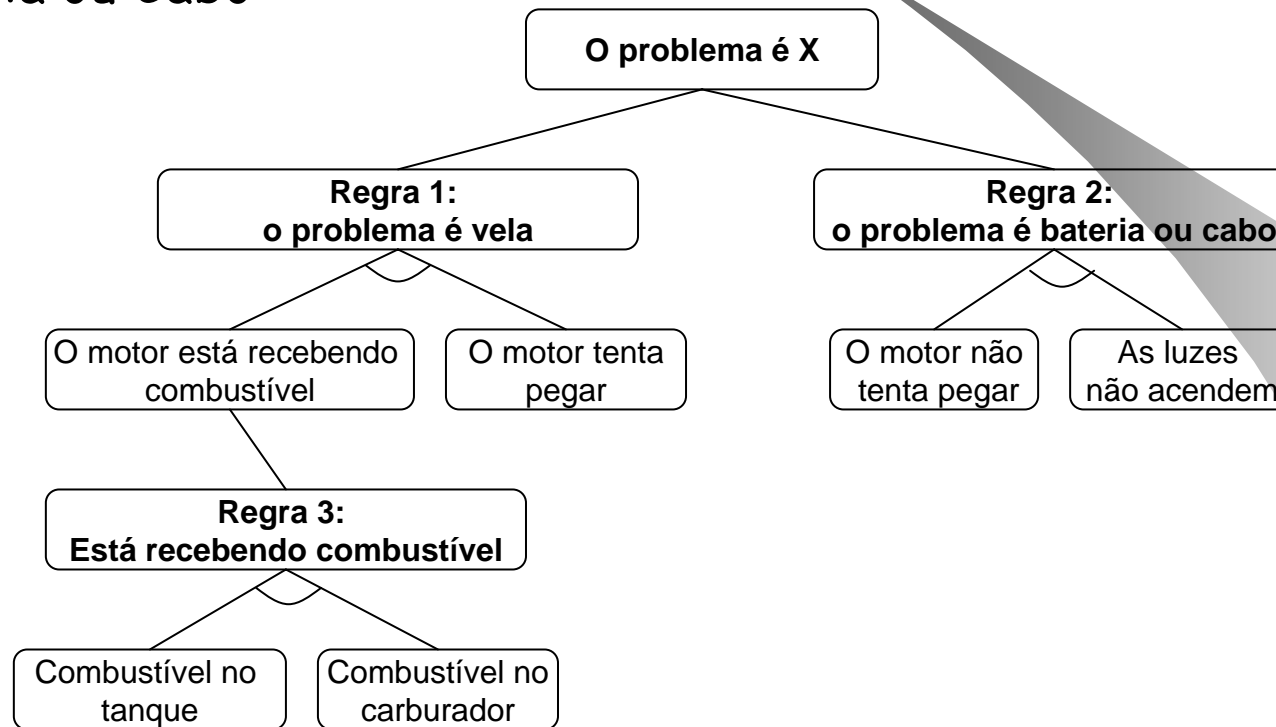
- o motor não tenta pegar e
 - as luzes não acendem
 - então o problema é bateria ou cabo

- Regra 3: se

- houver combustível no tanque de combustível e
 - houver combustível no carburador
 - então o motor está recebendo combustível

S.E. Baseados em Regras

- ♦ Sistema de Produção: Raciocínio Guiado por Objetivo
X=Bateria ou Cabo



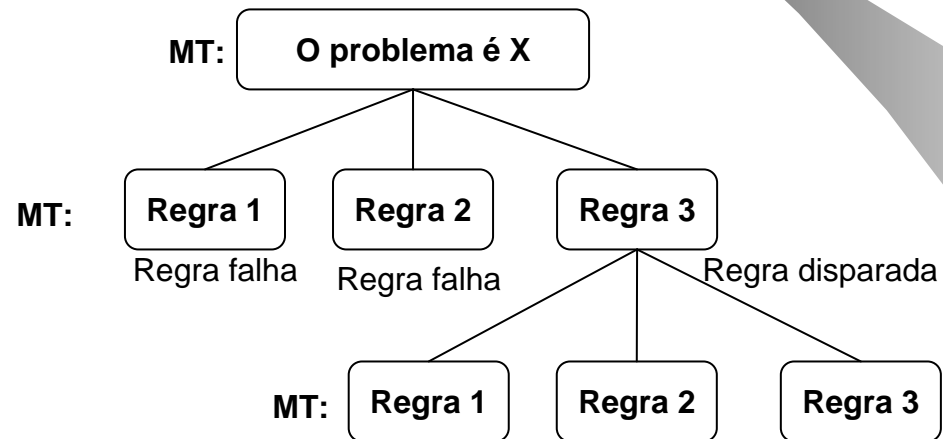
- O processo de busca se dá em profundidade, já que ele busca exaustivamente cada subobjetivo encontrado na base de regras antes de se mover para qualquer outro objetivo irmão.

S.E. Baseados em Regras

- ♦ Sistema de Produção: Raciocínio Guiado por Dados
 - Compara o conteúdo da memória de trabalho com as condições de cada regra na base de regras.
 - Se os dados na memória de trabalho permitir o disparo de uma nova regra, o resultado vai para a memória de trabalho e então o controle move para a próxima regra.
 - E após considerar todas as regras, a busca recomeça no início do conjunto de regras.

S.E. Baseados em Regras

- ♦ Sistema de Produção: Raciocínio Guiado por Dados
Informação: Motor está recebendo Combustível



- O processo de busca se dá em amplitude

S.E. Baseados em Regras

- ♦ Heurística e Controle em Sistemas Especialistas
 - A regra **se p, q e r então s** pode ser interpretada como uma série de procedimentos ou passos para resolver problemas.
 - Esse método procedimental reflete a estratégia de solução do especialista.
 - **Exemplo:** ordenar as premissas de uma regra de forma que, seja testado primeiro aquilo que seja mais provável de não ser válido ou então mais fácil de se confirmar.
 - Isso permite eliminar uma regra (e com isso parte do espaço de busca) o mais cedo possível.
 - Assim: se o motor está tentando pegar, não importa se ele está ou não recebendo combustível.
 - Esses aspectos são fundamentados em naturezas heurísticas.

S.E. Baseados em Regras

- ♦ Heurística e Controle em Sistemas Especialistas
 - O algoritmo RETE (Forgy, 1982) pode ser utilizado para otimizar a busca por todas as regras potencialmente úteis.
 - O RETE compila regras numa estrutura de rede que permite que o sistema realize o casamento de regras com dados, seguindo diretamente um ponteiro para a regra.
 - Esse algoritmo acelera bastante a execução, especialmente grandes conjuntos de regras.
 - Sistemas fortemente heurísticos podem falhar, ou por encontrar um problema que não se encaixe nas regras, ou aplicar erroneamente uma regra heurística não apropriada.
 - Outras abordagens, tais como baseadas em modelo, em casos ou híbridas, tentam conferir esta flexibilidade.

Sistemas Baseados em Modelos

- ♦ Surgiram em meados dos anos 70 e evoluíram através dos anos 80 (Davis e Hamscher, 1992).
- ♦ Sistema de raciocínio baseado em conhecimento, com análise fundamentada diretamente na especificação e na funcionalidade de um sistema físico.
- ♦ É criada uma simulação, referida como “qualitativa”, da função do que está sendo compreendido ou reparado.
- ♦ Uma falha se manifesta na discrepância entre os comportamentos previsto e observado.

Sistemas Baseados em Modelos

- ♦ Raciocínio qualitativo baseado em modelo inclui:
 - Uma descrição de cada componente do dispositivo: simular o comportamento do componente.
 - Uma descrição da estrutura interna do dispositivo: representação dos componentes e interconexões, juntamente com a habilidade de simular interações dos componentes.
 - Diagnóstico de um problema particular, com desempenho real do dispositivo, geralmente medidas de suas entradas e saídas.
- ♦ A tarefa é determinar quais destes componentes poderiam ter falhado, de modo que explique o comportamento observado.
- ♦ Em vez de raciocinar diretamente a partir de fenômenos observados buscando explicações causais, a abordagem baseada em modelo tenta representar dispositivos e configurações de dispositivos num nível causal ou funcional.

Sistemas Baseados em Casos

- ♦ Raciocínio a partir de casos, exemplos de problemas passados e suas soluções.
- ♦ Usa uma base de dados explícita de soluções de problema para tratar novas situações de solução de problema.
- ♦ Permitem que o sistema aprenda a partir da sua experiência, pois após encontrar uma solução, pode armazená-la

Sistemas Baseados em Casos

- ♦ Para cada novo caso:
 - Recupera casos da memória
 - Modificam um caso recuperado de modo que ele seja aplicável à situação corrente
 - Aplicam o caso transformado
 - Armazenam a solução, com um registro de sucesso ou fracasso, para uso futuro.

Sistemas Baseados em Casos

- ♦ Kolodner (1993) propõe um conjunto de heurísticas organizar o armazenamento e recuperação de casos:
 - Preferência orientada a objetivo
 - Preferência por características salientes
 - Preferência por maior especificidade
 - Preferência por ocorrências frequentes
 - Preferência por atualidade
 - Preferência por facilidade de adaptação

Raciocínio baseado em regras

- Vantagens:

- Capacidade de usar, de uma forma direta, o conhecimento experimental, adquirido de especialistas
- As regras são apropriadas para busca em espaço de estados
- É possível um bom desempenho em domínios limitados
- Bons recursos de explanação

- Desvantagens:

- Frequentemente regras são de natureza heurísticas e não capturam o conhecimento funcional.
- Regras heurísticas tendem a ser "frágeis" e não são capazes de lidar com informação faltante ou valores inesperados
- Explicações funcionam apenas no nível descritivo omitindo explicações teóricas
- O conhecimento tende a ser dependente da tarefa.

Raciocínio baseado em Casos

- Vantagens:

- Habilidade de codificar diretamente conhecimento histórico
- Permite atalhos no raciocínio
- Permite que um sistema evite erros passados e explore sucessos passados
- Não é necessária a análise extensiva do conhecimento do domínio
- Estratégias de indexação apropriadas aumentam a capacidade de compreensão e o poder de solução de problemas.

- Desvantagens:

- Os casos frequentemente não incluem um conhecimento profundo do domínio.
- Uma grande base de casos pode sofrer problemas de armazenamento
- É difícil determinar bons critérios para indexar e fazer casamento de casos.

Raciocínio baseado em Modelo

- Vantagens:

- Habilidade de usar conhecimento funcional ou estrutural do domínio
- Os raciocinadores tendem a ser robustos
- Algum conhecimento pode ser transferido entre tarefas
- Os raciocinadores podem fornecer explicações causais.

- Desvantagens:

- Falta de conhecimento experimental (descritivo) do domínio
- Requer um modelo explícito do domínio
- Alta complexidade
- Situações excepcionais

Lógicas Não-Clássicas e Tratamento de Incertezas

- ♦ Uma das características da lógica clássica é o axioma do terceiro excluído, isto é, não existe uma terceira alternativa para um valor verdade além do par {Verdadeiro, Falso}.
- ♦ No mundo real, é comum que os conhecimentos disponíveis não sejam nem absolutamente verdadeiros nem absolutamente falsos, podendo ser, por exemplo paradoxais, incertos, desconhecidos, indeterminados, verdadeiros em geral, verdadeiros com uma certa probabilidade, etc.
- ♦ Para estender a lógica clássica, é necessário alterar o conjunto de valores verdade.

Lógicas Não-Clássicas e Tratamento de Incertezas

- ♦ Dois tipos de formalismos foram propostos:
 - valores verdade numéricos
 - (probabilidade, lógica nebulosa, teoria das possibilidades, etc.)
 - valores verdade simbólicos
 - (3, 4 ou mais valores verdade)

Lógicas Não-Clássicas e Tratamento de Incertezas

- ♦ Lógica Multivalores - Valores de Verdade SIMBOLICOS
 - Uma lógica com três valores de verdade admite um valor de verdade que representa um valor entre verdadeiro e falso.
 - A interpretação deste terceiro valor difere nas diversas lógicas
 - pode indicar um estado de parcial ignorância;
 - pode indicar a impossibilidade de se atribuir verdadeiro ou falso;
 - pode indicar a falta de sentido de se atribuir verdadeiro ou falso.

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE KLEENE

- Concebida originalmente para acomodar declarações matemáticas não decididas.
- O terceiro valor de verdade é \perp ou "u" de "undecided" (não decidido), indica que "não se sabe se é verdadeiro ou falso".
- Não admite a interpretação de que "não é verdadeiro nem falso".
- As tabelas verdade propostas por Kleene são:

	V	F	\perp
\neg	F	V	\perp

\wedge	V	F	\perp
V	V		
F	F	F	
\perp	\perp	F	\perp

\vee	V	F	\perp
V	V		
F	V	F	
\perp	V	\perp	\perp

\rightarrow	V	F	\perp
V	V	F	\perp
F	V	V	V
\perp	V	\perp	\perp

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE KLEENE

- O valor de verdade indecidido indica este estado de ignorância, de maneira que quando uma fórmula lógica pode ter seu valor de verdade decidido, a despeito desta ignorância, este valor deve ser adotado, assim:

- $V \vee \perp = V$ e $F \wedge \perp = F$, mas
- $V \wedge \perp = \perp$ e $F \vee \perp = \perp$

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE LUKASIEWICZ

- Lukasiewicz usa \perp ou "i" para terceiro valor de verdade (i de "indeterminate").
- Sua lógica foi desenvolvida para lidar com afirmações incertas futuras, ou seja, a existência de proposições contingentes sobre o futuro.
- De acordo com sua interpretação, tais proposições não são nem verdadeiras nem falsas, mas (metafisicamente) indeterminadas.

Lógicas Não-Clássicas

- ♦ Lógica Multivalores

LÓGICA DE LUKASIEWICZ

- Há uma diferença em relação à interpretação de "u" de Kleene. O "i" não é resultante da falta de informação, mas sim do impedimento de se poder fazer uma avaliação conclusiva para verdadeiro ou falso. Algo que ainda não ocorreu é menos "real" do que algo verdadeiro ou falso.

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE LUKASIEWICZ

- A base da filosofia que suporta a lógica de Lukasiewicz é aristotélica, ou seja, considerar algo futuro como verdadeiro ou falso é adotar o fatalismo, doutrina que prega que o futuro é pré-determinado.
- A única diferença entre as tabelas verdade das lógicas de Kleene e Lukasiewicz é o valor de $\perp \rightarrow \perp$, que para Kleene é \perp e para Lukasiewicz é V.
- As tabelas verdade propostas por Kleene são:

	V	F	\perp
\neg	F	V	\perp

\wedge	V	F	\perp
V	V		
F	F	F	
\perp	\perp	F	\perp

\vee	V	F	\perp
V	V		
F	V	F	
\perp	V	\perp	\perp

\rightarrow	V	F	\perp
V	V	F	\perp
F	V	V	V
\perp	V	\perp	V

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE BOCHVAR

- O objetivo de Bochvar ao propor uma lógica de três valores verdade foi o tratamento formal dos paradoxos semânticos.
 - Paradoxo do Cretense - Um cretense afirma que todos os cretenses são mentirosos.
 - "Esta sentença é falsa".
- O terceiro valor de verdade de Bochvar corresponde a uma proposição paradoxal "m" (de meaningless).
- Ao contrário de "u" e de "i", que correspondem a um grau de informação menor que verdadeiro ou falso, o valor de verdade paradoxal é ao mesmo tempo verdadeiro e falso.

Lógicas Não-Clássicas

- ♦ Lógica Multivalores

LÓGICA DE BOCHVAR

- Os operadores \neg e \wedge propostos por Bochvar são idênticos aos de Kleene e Lukasiewicz, mas os operadores \vee e \rightarrow são distintos.

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE BOCHVAR

- De certa maneira, o valor verdade paradoxal tem um caráter "contagioso" tornando paradoxal qualquer fórmula onde um elemento seja paradoxal.

v	V	F	m	→	V	F	m
V	V			V	V	F	m
F	V	F		F	V	V	m
m	m	m	m	m	m	m	m

Lógicas Não-Clássicas

♦ Lógica Multivalores

LÓGICA DE BELNAP

- As lógicas de Kleene (1952), Lukasiewicz (1920) e Bochvar (1939) são anteriores ao início da IA.
- Em 1977, Belnap propôs uma lógica de 4 valores verdade, projetada especificamente para servir como base para um sistema computacional de perguntas e respostas capaz de, mesmo em face de contradições, continuar a gerar respostas compatíveis com as informações anteriormente armazenadas.
- O conjunto de valores verdade é o seguinte:
 $B = \{\{\}, \{V\}, \{F\}, \{V,F\}\}$
onde os valores tem as seguintes interpretações:
 $\{\}$ = desconhecido
 $\{V\}$ = absolutamente verdadeiro
 $\{F\}$ = absolutamente falso
 $\{V,F\}$ = contraditório

Lógicas Não-Clássicas

♦ Lógicas Não Monotônicas

- Fórmulas quantificadas universalmente na lógica de predicados são válidas para qualquer elemento do domínio, sem nenhuma exceção.
- Certas situações do mundo real (percepção, ambigüidade, senso comum, causalidade ou predição) são a tal ponto complexas, que qualquer conhecimento sobre elas será inevitavelmente incompleto.
- Um formalismo para raciocinar neste tipo de situação deve admitir expressões que sejam válidas em geral e capazes de reconhecer e assimilar exceções quando necessário.
- Neste caso, corre-se o risco de retirar conclusões anteriores face a novas informações, o que caracteriza a não-monotonicidade.

Lógicas Não-Clássicas

♦ Lógica Não Monotônicas

EXEMPLO

- Todo pássaro pode voar.
- Tweety pode voar?
- Na ausência de informações contrárias um pássaro normal voa - Logo, Tweety voa.
- Mas descobre-se que Tweety é um pinguim (pinguins não são pássaros normais no que se refere à capacidade de voar) - Logo, Tweety não voa.
- Mas descobre-se que Tweety é um pinguim do planeta Krypton e que ele não é um pássaro ou um pinguim normal, e que no planeta Krypton pinguins voam - Logo, Tweety voa.
- Mas descobre-se ...

Tratamento de Incertezas

- ♦ A imperfeição da informação é geralmente conhecida na literatura de sistemas baseados em conhecimento de incerteza.
- ♦ No entanto, este termo é muito restritivo; o que se convencionou chamar de tratamento de incerteza pode, na verdade, estar endereçando outras imperfeições da informação, com imprecisão, conflito, ignorância parcial, etc.

Tratamento de Incertezas

- ♦ **Informação perfeita:** O filme começa às 18h 15 min.
- ♦ **Informação imprecisa:** O filme começa entre 8h e 9h.
- ♦ **Informação incerta:** Eu acho que o filme começa às 8h.
- ♦ **Informação vaga:** O filme começa lá pelas 8h.
- ♦ **Informação probabilista:** É provável que o filme comece às 8h.
- ♦ **Informação possibilista:** É possível que o filme comece às 8h.
- ♦ **Informação inconsistente:** Maria disse que o filme começa às 8h mas João disse que ele começa às 10h.
- ♦ **Informação incompleta:** Eu não sei a que horas o filme começa, mas normalmente neste cinema os filme começam às 8h.
- ♦ **Ignorância Total:** Eu não faço a menor idéia do horário do filme.

Tratamento de Incertezas

- ♦ As informações podem variar de perfeitas a completamente imperfeitas.
- ♦ Mesmo lidando diariamente com este tipo de informações, conseguimos tomar decisões razoáveis.
- ♦ O mesmo deveria ocorrer com sistemas baseados em conhecimento, em face de informações imperfeitas.

Tratamento de Incertezas

- ♦ Sabemos que o conhecimento humano não é determinístico. Não há especialista que sempre se encontre em condições de afirmar determinada conclusão com certeza absoluta. Graus de confiança são freqüentemente atribuídos às suas respostas, principalmente quando existe mais de uma. Este, sem dúvida, é um dos mais fortes pontos críticos na elaboração de uma representação computacional do saber humano.
- ♦ Vejamos a dificuldade em representar a confiabilidade das informações:
 - Especialistas humanos não se sentem confortáveis em pensar em termos de probabilidade. Suas estimativas não precisam corresponder àquelas definidas matematicamente;
 - Tratamentos rigorosamente matemáticos de probabilidade utilizam informações nem sempre disponíveis ou simplificações que não são claramente justificáveis em aplicações práticas.

Tratamento de Incertezas

- ♦ Para cada um dos tipos de informação existem modelos formais (e também informais) para tratamento.
 - A informação de conotação probabilista pode tratada pela teoria de probabilidades e pela teoria da crença ou evidência (também conhecida como Dempster-Schafer).
 - A informação imprecisa, de caráter possibilista e/ou vaga pode ser tratada pela teoria dos conjuntos nebulosos, "rough sets" ou teoria das possibilidades.
 - Informações inconsistentes e/ou incompletas podem ser tratadas por lógicas não clássicas (Belnap, Lucziewicz, etc).

Fatores de Certeza

- ♦ O primeiro sistema a utilizar-se dos fatores de certeza foi o MYCIN, para recomendar terapias apropriadas para pacientes com infecção bacteriológicas;
- ♦ O Fator de Certeza (FC) foi originalmente definido como a diferença entre a crença e a descrença:

$$FC[H, E] = MC[H, E] - MD[H, E]$$

- ♦ $FC[H, E] \Rightarrow$ FC na hipótese H dada à evidência E
- ♦ $MC[H, E] \Rightarrow$ medida de crença em H dado E
- ♦ $MD[H, E] \Rightarrow$ medida de descrença em H dado E

Fatores de Certeza - Utilidade

- ♦ É um mecanismo simples para combinar crença e descrença em um número;
- ♦ Pode ser usado para um conjunto de hipóteses em ordem de importância.
 - Por exemplo, se um paciente tem certos sintomas os quais sugerem diversas doenças possíveis, a doença com um alto FC poderia ser a primeira a ser investigada pelos testes ordenados;

Fatores de Certeza - Utilidade

- ♦ O FC indica a rede de crença em uma hipótese sobre alguma evidência. Um FC positivo significa que a evidência suporta a hipótese desde que $MC > MD$. Um $FC=1$ significa que a evidência definitivamente prova a hipótese. Um $FC=0$ significa:
 1. que não existe evidência ou ela é irrelevante ($MC=MD=0$) ou
 2. A crença é cancelada pela descrença pois as duas são igualmente fortes ou fracas ($MC=MD \neq 0$).

Fatores de Certeza - Utilidade

- ♦ O FC negativo significa que a evidência favorece a *negação* da hipótese, desde que $MC < MD$ (i.e. , existem mais razões para a descrença em uma hipótese do que para a crença nela);
- ♦ Por exemplo, um $FC = -70\%$ significa que a descrença é 70% maior do que a crença, e vice-versa; entretanto, diferentes valores de MC e MD levam a um mesmo valor de FC :

$$FC = 0,80 = 0,80 - 0$$

$$FC = 0,80 = 0,95 - 0,15$$

Fatores de Certeza - Utilidade

Características	Valores
Variações	$0 \leq MC \leq 1$ $0 \leq MD \leq 1$ $-1 \leq FC \leq 1$
Certeza das Hipóteses Verdadeiras $P(H E) = 1$	$MC = 1$ $MD = 0$ $FC = 1$
Certeza das Hipóteses Falsas $P(\sim H E) = 1$	$MC = 0$ $MD = 1$ $FC = -1$
Perda de Evidência $P(H E) = P(H)$	$MC = 0$ $MD = 0$ $FC = 0$

Fatores de Certeza - Utilidade

- A combinação de evidências requer regras como as dadas a seguir:

Evidência, E	Certeza do Antecedente
$E1$ e $E2$	$\min[FC(H,E1), FC(H,E2)]$
$E1$ ou $E2$	$\max[FC(H,E1), FC(H,E2)]$
$\neg E$	$-FC(H,E)$

- Podemos criar ainda expressões mais complexas:

$$E = (E1 \wedge E2 \wedge E3) \vee (E4 \wedge \neg E5)$$

$$E = \max[\min(E1, E2, E3), \min(E4, E5)]$$

Fatores de Certeza

- ♦ A fórmula fundamental para o FC de uma regra "*Se E então H*" é dado pela fórmula:

$$FC(H,e) = FC(H,E) * FC(E,e)$$

- ♦ $FC(E,e)$ é o fator de certeza da evidência E baseada na evidência incerta e;
- ♦ $FC(H,E)$ é o fator de certeza da hipótese supondo que a evidência E é conhecida com certeza, quando $FC(E,e) = 1$;
- ♦ $FC(H,e)$ é o FC da hipótese baseada na incerteza da evidência e.

Fatores de Certeza

- ♦ Por exemplo, tem-se a regra $A \wedge B \wedge C \rightarrow D$ com uma evidência de 0,7 (também chamado de fator de atenuação):

$$FC(D,E) = FC(H, A \cap B \cap C) = 0,7$$

- ♦ Seja e a evidência observada que dirige a conclusão de que as E_i são conhecidas com certeza, suponha que:

$$FC(A,e) = 0,5$$

$$FC(B,e) = 0,6$$

$$FC(C,e) = 0,3$$

Fatores de Certeza

- ♦ Logo:

$$FC(E,e) = FC(A \cap B \cap C, e)$$

$$FC(E,e) = \min[FC(A,e), FC(B,e), FC(C,e)]$$

$$FC(E,e) = 0,3$$

- ♦ O fator de certeza da conclusão é:

$$FC(D,E) = FC(D,E) * FC(D,e) = 0,7 * 0,3 = 0,21$$

Fatores de Certeza - Considerações

- ♦ Ainda que o MYCIN tenha tido sucesso em diagnóstico, existem dificuldade com os fundamentos teóricos dos FCs. A maior vantagem dos FC foi a simples computação pela qual a incerteza seria propagada no sistema.
- ♦ Conclusão: não existe uma técnica, ou forma de raciocínio melhor que outros. Dependendo do problema a ser resolvido, existem escolhas mais razoáveis.

Raciocínio Probabilístico

- ♦ O Raciocínio Probabilístico é talvez o mais antigo que trata com mecanismos de incerteza.
- ♦ Apóia-se em informações probabilísticas sobre fatos de um domínio e chega a uma conclusão a respeito de um novo fato, conclusão esta, que fica associada a uma probabilidade.
- ♦ Quando se fala de probabilidade neste contexto, não se faz referência a números, e sim, a um tipo de raciocínio.
- ♦ Exemplo:
 - "A chance de que um paciente portador da doença D apresente no futuro próximo o sintoma S é p ".
- ♦ A verdade desta afirmação não é o valor preciso de p , mas um valor de crença do médico.

Raciocínio Probabilístico

- ♦ A teoria da probabilidade adota a frase epistêmica "...posto que C é conhecido" como uma primitiva da linguagem. Sintaticamente isto é denotado por:

$$P(A \mid C) = p$$

- ♦ onde A é uma dada proposição.
- ♦ Esta frase combina as noções de conhecimento e crença pela atribuição à A de um grau de crença p , dado o conhecimento de C .
- ♦ C é chamado de "contexto da crença em A ", e a notação $P(A \mid C)$ é chamada "Probabilidade Condicional de Bayes".
- ♦ O teorema de Bayes provê a base para o tratamento da imperfeição da informação. Ele computa a probabilidade de um dado evento, dado um conjunto de observações.

Raciocínio Probabilístico

- ♦ Seja:
 - $P(H_i | E)$ a probabilidade de que a hipótese H_i seja verdadeira dada a evidência E .
 - $P(E | H_i)$ a probabilidade que a evidência E será observada se a hipótese H_i for verdadeira.
 - $P(H_i)$ a probabilidade "a priori" que a hipótese H_i é verdadeira na ausência de qualquer evidência específica.
 - K o número de hipóteses possíveis
- ♦ O teorema de Bayes é formulado como:

$$P(H_i | E) = \frac{P(E | H_i).P(H_i)}{\sum P(E | H_j).P(H_j)}$$

Raciocínio Probabilístico

♦ Exemplo:

- Suponhamos que no meio da noite dispare o alarme contra ladrões da nossa casa. Queremos então saber quais são as chances de que esteja havendo uma tentativa de roubo. Suponhamos que existam 95% de chances de que o alarme dispare quando uma tentativa de roubo ocorre, que em 1% das vezes o alarme dispare por outros motivos, e que em nosso bairro existe uma chance em 10.000 de uma dada casa ser roubada em um dado dia.
- Temos então:
 $P(\text{alarme} \mid \text{roubo}) = 0,95$
 $P(\text{alarme} \mid \sim \text{roubo}) = 0,01$
 $P(\text{roubo}) = 0,0001$
- Então $P(\text{roubo} \mid \text{alarme}) = 0,00941 = 0,9\%$
- Este valor pode ser intuitivamente entendido quando verificamos que as chances de haver um roubo e o alarme tocar (0,000095) são muito pequenas em relação às chances de haver um alarme falso.

Raciocínio Probabilístico

- ♦ Hoje em dia se fala da probabilidade subjetiva.
- ♦ Ela trata com eventos que não tem uma base histórica sobre a qual se possa extrapolar.
- ♦ A probabilidade subjetiva é uma crença ou opinião expressa como uma probabilidade.
- ♦ Exemplo:
 - Em SE para diagnóstico médico, um evento poderia ser:
 - $E = \text{"O paciente está coberto com manchas vermelhas"}$
 - e a proposição é:
 - $A = \text{"O paciente tem sarampo"}$.
 - A probabilidade condicional $P(A | E)$ não é uma probabilidade no sentido clássico ou frequencista.
 - Ela pode ser interpretada como o grau de crença que A é verdadeiro dado o evento E .

Raciocínio Probabilístico

- ♦ Atualmente existem shells para o desenvolvimento de Sistemas Especialistas com raciocínio probabilista, dentre eles tem-se o SPIRIT, o HUGIN e o NETICA.
- ♦ Dificuldades com o Método Bayesiano
 - A obtenção das probabilidades das hipóteses H_i e as condicionais $P(H_i | E)$ é considerado uma tarefa difícil porque as pessoas não sabem estimar probabilidades. No entanto, as estimativas necessárias de probabilidade são feitas pelo especialista a partir de seu conhecimento e experiência no domínio pesquisado.
 - A base de conhecimento tem que ser completa. Isto é, todas as evidências relevantes às hipóteses consideradas devem estar explícitas na base de conhecimento.
 - Em probabilidade parte-se do fato que as evidências são independentes. Isto nem sempre é verdadeiro no caso das doenças, posto que alguns sintomas poderiam ser evidência de outros.

Raciocínio por Crença

- ♦ Baseia-se essencialmente nos trabalhos feitos originalmente por Dempster, que tentou modelar a incerteza por uma faixa de probabilidades, mais do que um simples número probabilístico. Shafer estendeu e refinou o trabalho de Dempster.
- ♦ A teoria de Dempster-Shafer supõe que existe um conjunto fixo de elementos mutuamente exclusivos e exaustivos, chamado "meio" e simbolizado por θ :

$$\theta = \{\theta_1, \theta_2, \theta_3, \dots, \theta_n\}$$

Raciocínio por Crença

- ♦ O meio é o conjunto de objetos que são de interesse, por exemplo:
 $\theta = \{\text{avião}, \text{submarino}, \text{trem}, \text{ônibus}\}$
- ♦ Como os elementos são mutuamente exclusivos (um trem não é um avião) e o meio exaustivo, pode existir somente um subconjunto para cada pergunta do sistema;
- ♦ Pergunta: *"Qual deles é transporte terrestre?"*
- ♦ Resposta: Um subconjunto de θ , $\{\theta_3, \theta_4\} = \{\text{trem}, \text{ônibus}\}$

Raciocínio por Crença

- ♦ Assim, cada subconjunto de θ pode ser interpretado como uma possível resposta a uma pergunta.
- ♦ Desde que os elementos são mutuamente exclusivos e o meio exaustivo, pode existir somente um subconjunto com a resposta correta.
- ♦ Neste modelo, a informação fornecida por uma fonte de conhecimento a respeito do valor real de uma variável x , definida em um universo de discurso θ , é codificada sob a forma de um corpo de evidência sobre θ .

Raciocínio por Crença

- Um corpo de evidência é caracterizado por um par (F, m) , onde F é uma família de subconjuntos de θ e m é uma função de massa.
- A função m é definida para todos os elementos de θ e todos os seus subconjuntos. Onde m é um valor que mede a quantidade de crença corretamente atribuída a um subconjunto de θ .
- Se θ contém n elementos, então existem 2^n subconjuntos de θ . Entretanto muitos destes subconjuntos não tem significado para o domínio do problema (e portanto o valor de m a eles associado será 0).
- A teoria de Dempster-Shafer não força crenças pelo desconhecimento de uma hipótese. Em vez disso a quantidade é designada somente aos subconjuntos do meio aos quais deseja-se designar crença.

Raciocínio por Crença

Seja um conjunto $\theta = \{A, B, C\}$

Suponha-se que feita uma pergunta sobre o conjunto θ surge uma evidência de 0,7 que a resposta encontra-se sobre os elementos A e C ; temos então: $m_1(\{A, C\}) = 0,7$

O restante da crença é designada ao meio.

Raciocínio por Crença

- Crença designada ao meio: $m_1(\theta) = 1 - 0,7 = 0,3$
- Note que a crença designada ao meio é diferente da descrença em relação ao fato (algo como $m_1(\neg\{A, C\})$). Isso acontece porque a crença designada ao meio inclui todo o meio θ , ou seja, $\{A, B, C\}$;
- A crença do meio não necessariamente precisa ser 1, como na teoria da probabilidade.

Raciocínio por Crença - Exemplo

- ♦ Inicia-se um universo exaustivo de hipóteses mutuamente exclusivas:
 $\theta = \{artrite, lupus, poliartrite, gota\}$
- ♦ A meta é anexar aos elementos de θ algum valor de crença. Nem todas as evidências sustentam diretamente elementos isolados. Em geral elas sustentam grupos de elementos (subconjuntos);

Raciocínio por Crença - Exemplo

- ♦ Por exemplo: febre pode sustentar {artrite, lupus} e erupções sustentam {lupus};
- ♦ Como os elementos são mutuamente exclusivos, as evidências em favor de um podem afetar a crença em outros;
- ♦ Inicialmente não se tem nenhuma informação sobre como escolher entre as quatro hipóteses (ou seja, não se tem nenhuma evidência) então $m(\theta) = 1$ o que significa que todos os outros valores são 0;

Raciocínio por Crença - Exemplo

- ♦ O valor real pode ser de um dos elementos: artrite (A), lupus (L), poliartrite (P) ou gota (G);
- ♦ Mas não se tem informações que permitam atribuir crença a algum deles (entretanto tem-se a certeza de que a resposta está em algum lugar deste conjunto);
- ♦ Supondo que surgiram evidências de febre, o que sugeriu, com fator de crença 0,6 que o diagnóstico correto é $\{A, L\}$ (lupus ou artrite);

Raciocínio por Crença - Exemplo

- Temos então os seguintes fatores:

$$m_1(\{A, L\}) = 0,6$$

$$m_1(\theta) = 0,4$$

- Outro sintoma (evidência) apresentado pelo paciente: erupções. Esta sugere com crença de 0,8 que o diagnóstico seria lúpus ($\{L\}$) temos o segundo fator de crença, dado por m_2 :

$$m_2(\{L\}) = 0,8$$

$$m_2(\theta) = 0,2$$

Raciocínio por Crença - Exemplo

As evidências podem ser combinadas numa soma ortogonal que é calculada pela somatória do produto das intersecções m .

Valores de m	$m_2(\{L\}) = 0,8$	$m_2(\theta) = 0,2$
$m_1(\{A, L\}) = 0,6$	$\{L\} = 0,48$	$\{A, L\} = 0,12$
$m_1(\theta) = 0,4$	$\{L\} = 0,32$	$(\theta) = 0,08$

$$m_3(\{L\}) = m_1 \otimes m_2(\{L\}) = 0,48 + 0,32 = 0,80$$

$$m_3(\{A, L\}) = m_1 \otimes m_2(\{A, L\}) = 0,12$$

$$m_3(\theta) = m_1 \otimes m_2(\theta) = 0,08 \text{ (não crença)}$$

Raciocínio por Crença - Exemplo

Vamos supor que surja uma terceira evidência conflitante:

$$m_3(\{P\}) = 0,95 \text{ e } m(\theta) = 0,05$$

Valores de m	$m_1 \otimes m_2 (\{L\}) = 0,8$	$m_1 \otimes m_2 (\{A, L\}) = 0,12$	$m_1 \otimes m_2 (\theta) = 0,08$
$m_3(\{P\}) = 0,95$	$\{\emptyset\} = 0,76$	$\{\emptyset\} = 0,114$	$\{P\} = 0,076$
$m_3(\theta) = 0,05$	$\{L\} = 0,04$	$\{A, L\} = 0,006$	$(\theta) = 0,004$

O conjunto nulo, $\{\emptyset\}$, ocorre porque $\{P\}$ e $\{L\}$ não têm elementos comuns, assim como entre $\{P\}$ e $\{A, L\}$

Raciocínio por Crença - Exemplo

$$m_1 \otimes m_2 \otimes m_3(\{P\}) = 0,076$$

$$m_1 \otimes m_2 \otimes m_3(\{L\}) = 0,04$$

$$m_1 \otimes m_2 \otimes m_3(\{A, L\}) = 0,006$$

$$m_1 \otimes m_2 \otimes m_3(\theta) = 0,004$$

$$m_1 \otimes m_2 \otimes m_3(\{\emptyset\}) = 0, \text{ pela definição do conjunto vazio}$$

- ♦ A soma de todos os m :

$$\Sigma m_1 \otimes m_2 \otimes m_3(X) = 0,076 + 0,04 + 0,006 + 0,004 = 0,126$$

Raciocínio por Crença - Exemplo

- ♦ Uma soma de 1 é requerida desde que as evidências combinadas seja um m válido (como retiramos os valores para o conjunto vazio, todas as demais evidências são válidas). Como a soma é menor que 1, é preciso fazer uma normalização, que nada mais é do que uma regra de 3 que diz "a soma de todos os m está para 100 assim como cada combinação de m está para x ";
- ♦ Calculamos assim, proporcionalmente, os novos valores normalizados dos m 's:

Raciocínio por Crença - Exemplo

$$m1 \otimes m2 \otimes m3(\{P\}) = 0,603$$

$$m1 \otimes m2 \otimes m3(\{L\}) = 0,317$$

$$m1 \otimes m2 \otimes m3(\{A, L\}) = 0,0476$$

$$m1 \otimes m2 \otimes m3(\theta) = 0,0031$$

- ♦ Nota-se que a existência da evidência de $\{P\}$ prejudicou a crença em $\{L\}$, o que de fato era esperado.

Raciocínio por Crença - Dificuldades

- ♦ A normalização pode levar a resultados opostos às expectativas;
- ♦ Acontece porque a normalização ignora a crença de que um objeto considerado não existe, ou seja, a solução deve estar no conjunto.
- ♦ Um exemplo citado por Zadeh é o da crença de dois médicos, *A* e *B*, em uma doença de um paciente.

Raciocínio por Crença - Dificuldades

- ♦ As crenças são:

$$ma(meningite) = 0,99$$

$$ma(tumor\ cerebral) = 0,01$$

$$mb(traumatismo) = 0,99$$

$$mb(tumor\ cerebral) = 0,01$$

- ♦ Os médicos diferem grandemente no problema principal, mas a regra de Dempster valoriza a opinião comum, resultando 1 para o tumor cerebral. O resultado é inesperado e contra a nossa intuição.

Lógica Nebulosa

• Conceito

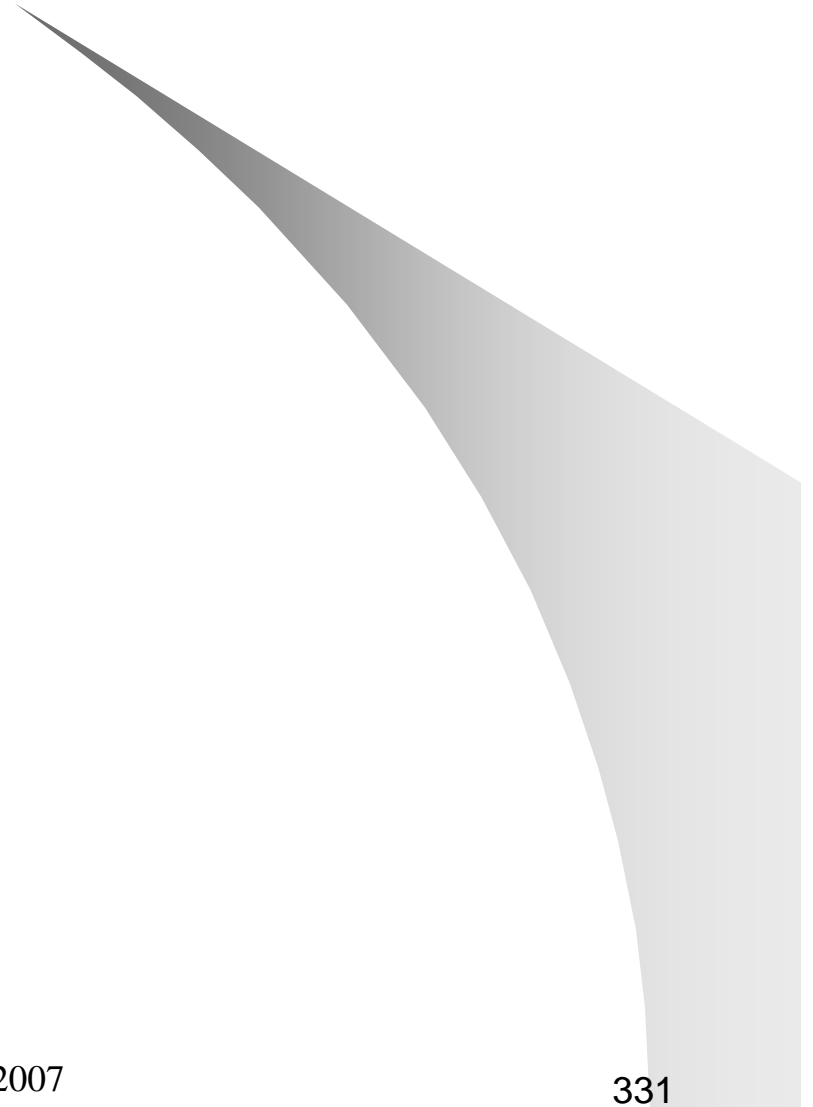
- ♦ Lógica nebulosa é uma lógica multivalorada capaz de capturar informações vagas, em geral descritas em uma linguagem natural e convertê-las para um formato numérico, de fácil manipulação pelos computadores de hoje em dia.
- ♦ É uma forma de especificar quão bem um objeto satisfaz uma descrição vaga.
- ♦ **Lógica Nebulosa e Conjuntos Nebulosos** permitem a representação de tal "vagueza".
- ♦ A lógica nebulosa pode ainda ser definida como a lógica que suporta os modos de raciocínio que são aproximados, ao invés de exatos, como estamos acostumados a trabalhar.

•Histórico

- ♦ A Lógica nebulosa foi desenvolvida por Lofti A. Zadeh da Universidade da Califórnia em Berkeley na década de 60 e combina lógica multivalorada, teoria possibilista, etc para poder representar o pensamento humano, ou seja, ligar a lingüística e a inteligência humana, pois muitos conceitos são melhores definidos por palavras do que pela matemática.

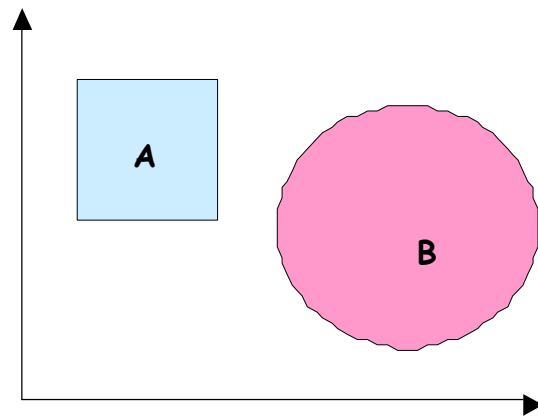
•Objetivo

- ♦ A lógica nebulosa objetiva fazer com que as decisões tomadas pela máquina se aproximem cada vez mais das decisões humanas, principalmente ao trabalhar com uma grande variedade de informações vagas e imprecisas, as quais podem ser traduzidas por expressões do tipo: a maioria, mais ou menos, talvez, etc. Antes do surgimento da lógica fuzzy essas informações não tinham como ser processadas.
- ♦ A lógica nebulosa vem sendo aplicada nas seguintes áreas: Análise de dados, Construção de sistemas especialistas, Controle e otimização, Reconhecimento de padrões, etc.

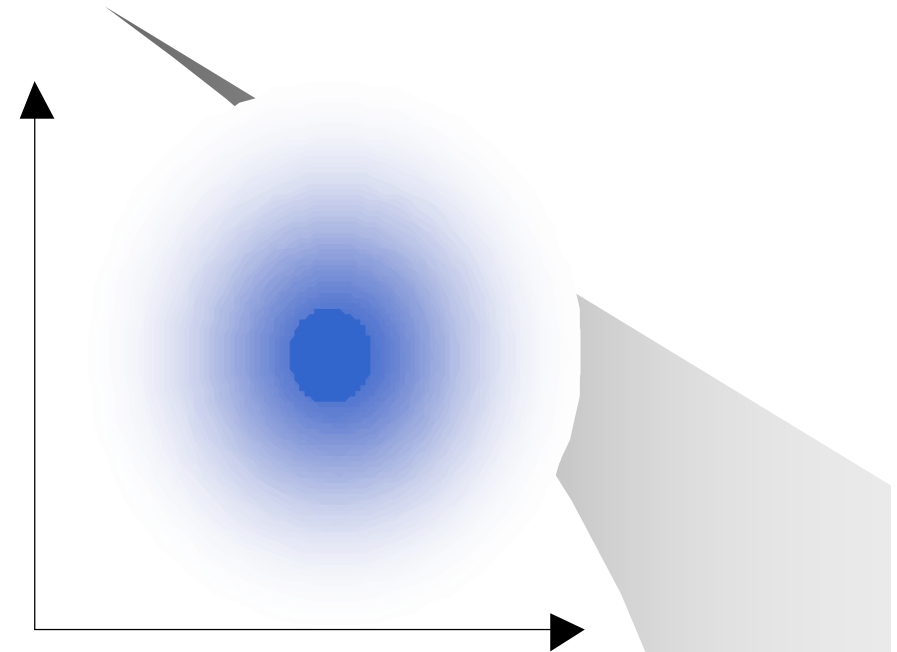




Conjuntos Clássicos x Conjuntos Nebulosos



Conjuntos Clássicos



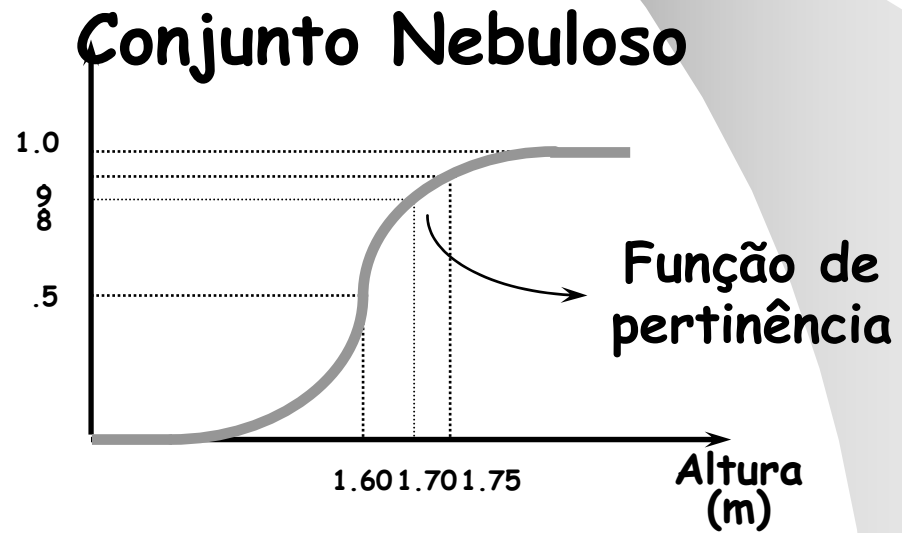
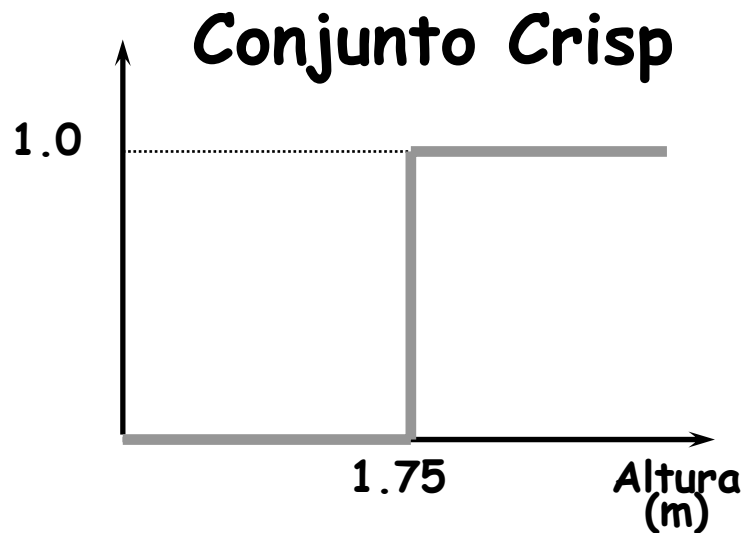
Conjunto Nebuloso



Conjuntos Nebulosos

- ♦ Conjuntos com limites imprecisos.

A = Conjunto de pessoas altas.



Conjuntos Nebulosos

- ♦ Definição formal:
 - Um conjunto nebuloso A em X é expresso como um conjunto de pares ordenados:

$$A = \{ (x, \underbrace{\mu_A(x)}_{\text{Função de pertinência (MF)}} \mid x \in X \}$$

Conjunto Nebuloso Universo ou Universo de discurso

Um conjunto Nebuloso é totalmente caracterizado por sua função de pertinência (MF).

Conjuntos Nebulosos

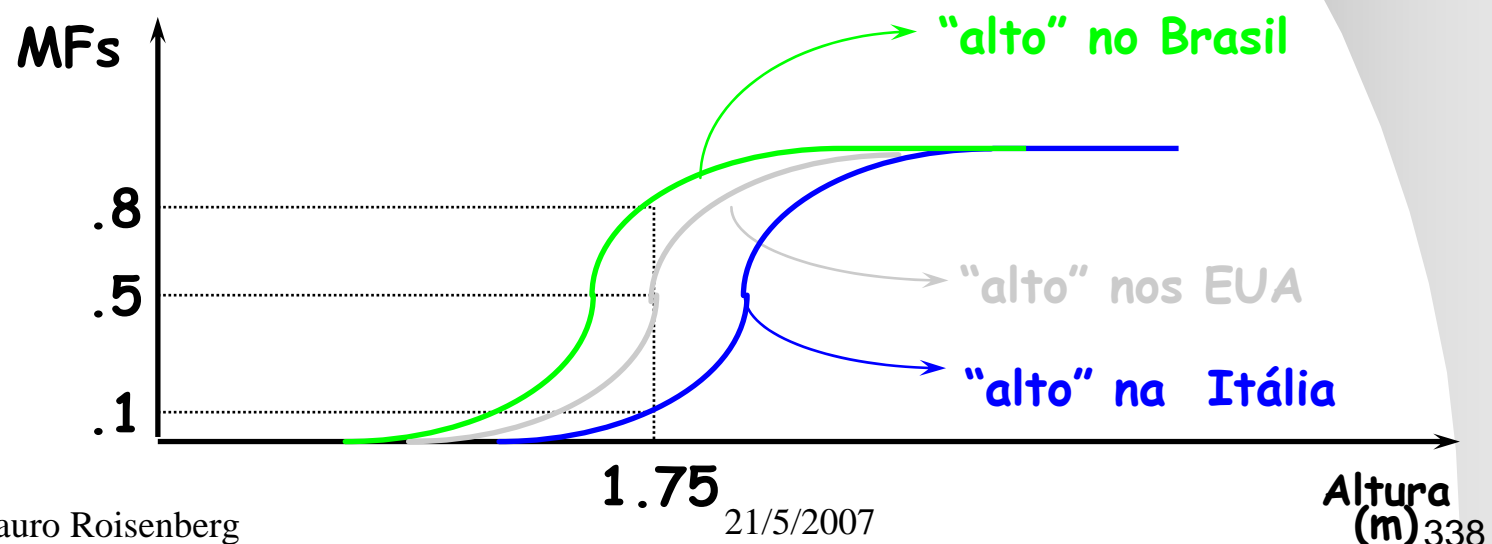
- Um conjunto difuso A definido no universo de discurso X é caracterizado por uma **função de pertinência** μ_A , a qual mapeia os elementos de X para o intervalo $[0,1]$.

$$\mu_A: X \rightarrow [0,1]$$

- Desta forma, a função de pertinência associa a cada elemento x pertencente a X um número real $\mu_{A(x)}$ no intervalo $[0,1]$, que representa o **grau de possibilidade**, ou **grau de verdade** de que o elemento x venha a pertencer ao conjunto A , isto é, o quanto é possível para o elemento x pertencer ao conjunto A .

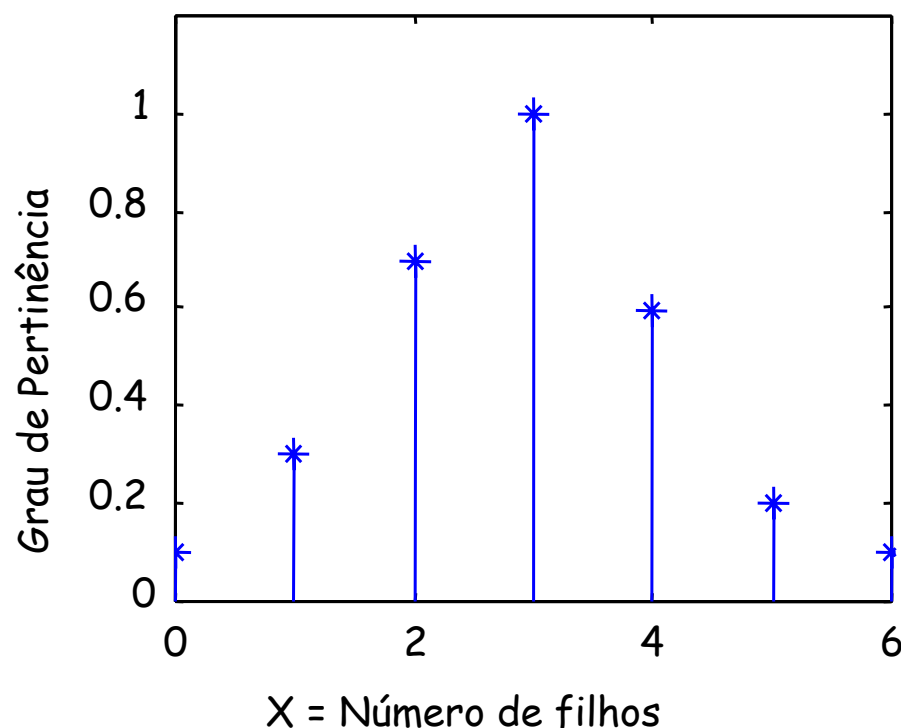
Função de Pertinência

- ♦ Reflete o conhecimento que se tem em relação a intensidade com que o objeto pertence ao conjunto difuso.
 - Características das funções de pertinência:
 - Medidas subjetivas;
 - Funções não probabilísticas monotonicamente crescentes, decrescentes ou subdividida em parte crescente e parte decrescente.



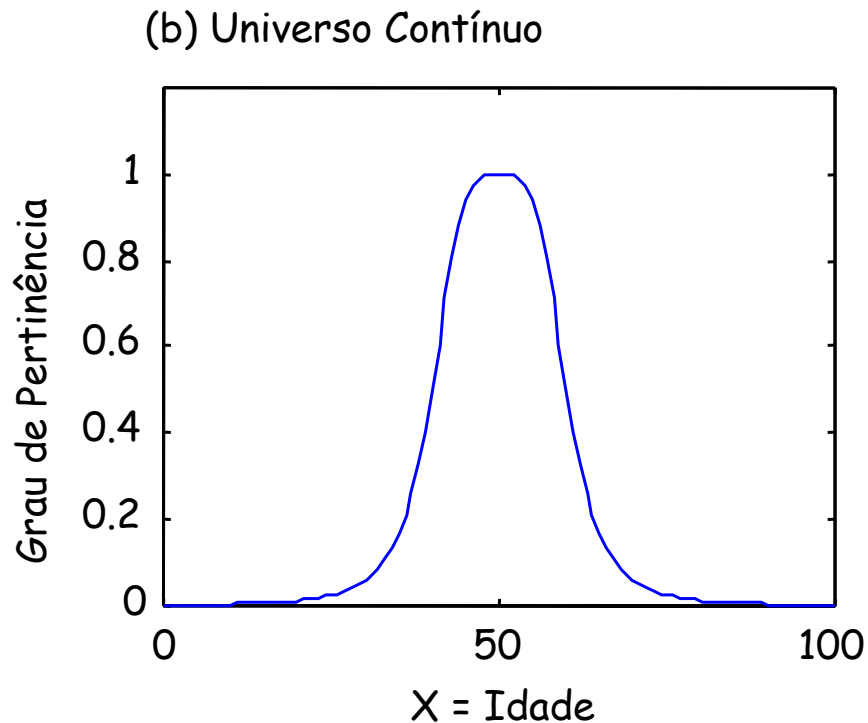
Universo Discreto

(a) Universo Discreto



- ♦ $X = \{SF, Boston, LA\}$ (discreto e não ordenado)
 - $C = \text{"Cidade desejável para se viver"}$
 - $C = \{(SF, 0.9), (Boston, 0.8), (LA, 0.6)\}$
- ♦ $X = \{0, 1, 2, 3, 4, 5, 6\}$ (discreto)
 - $A = \text{"Número de filhos razoável"}$
 - $A = \{(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)\}$

Universo Contínuo



$$\mu_B(x) = \frac{1}{1 + \left(\frac{x - 50}{10}\right)^2}$$

♦ $X = (\text{Conjunto de números reais positivos})_{(\text{contínuo})}$

- $B = \text{"Pessoas com idade em torno de 50 anos"}$

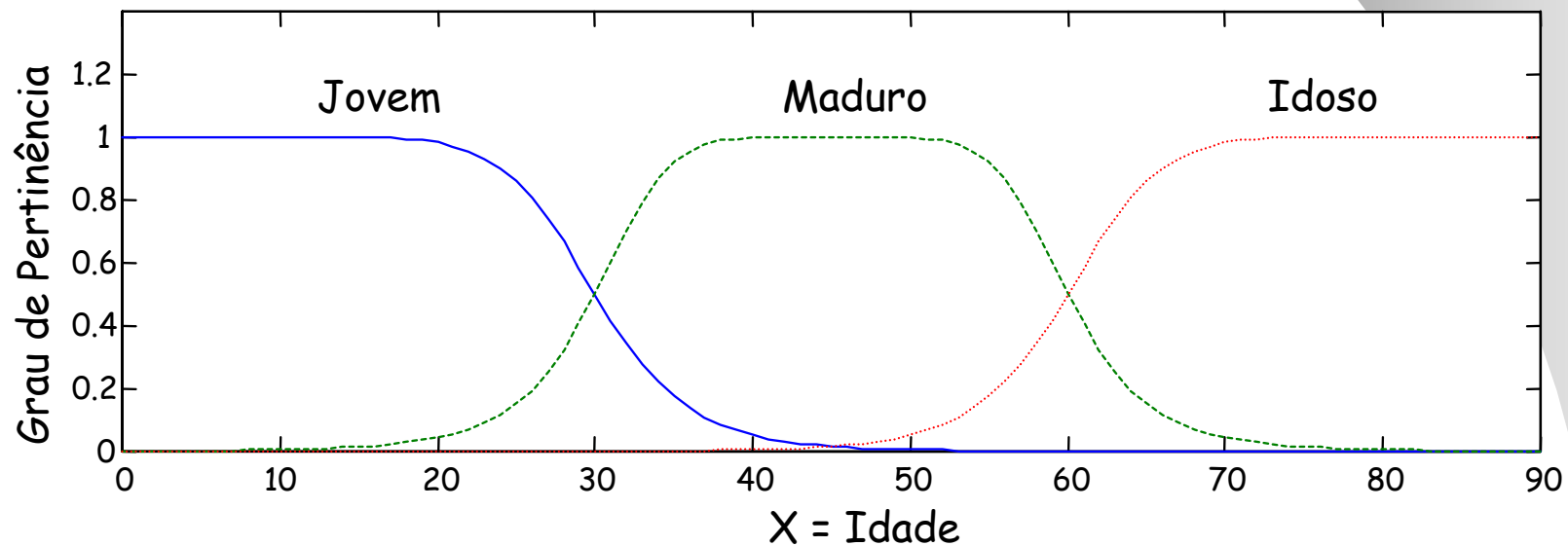
- $B = \{(x, \mu_B(x)) \mid x \text{ em } X\}$

Variáveis Lingüísticas

- ♦ Uma variável numérica possui valores numéricos:
 - Idade = 65
- ♦ Uma variável lingüística possui valores que não são números, e sim, palavras ou frases na linguagem natural.
 - Idade = idoso
- ♦ Um valor lingüístico é um conjunto fuzzy.
- ♦ Todos os valores lingüísticos formam um conjunto de termos:
 - $T(\text{idade}) = \{\text{Jovem, velho, muito jovem, ...}$
 $\text{Maduro, não maduro, ...}$
 $\text{Velho, não velho, muito velho, mais ou menos velho, ...}$
 $\text{Não muito jovem e não muito velho, ...}\}$

Partição Nebulosa

- ♦ Partição nebulosa da variável lingüística "Idade", formada pelos valores lingüísticos "jovem", "maduro" e "idoso".



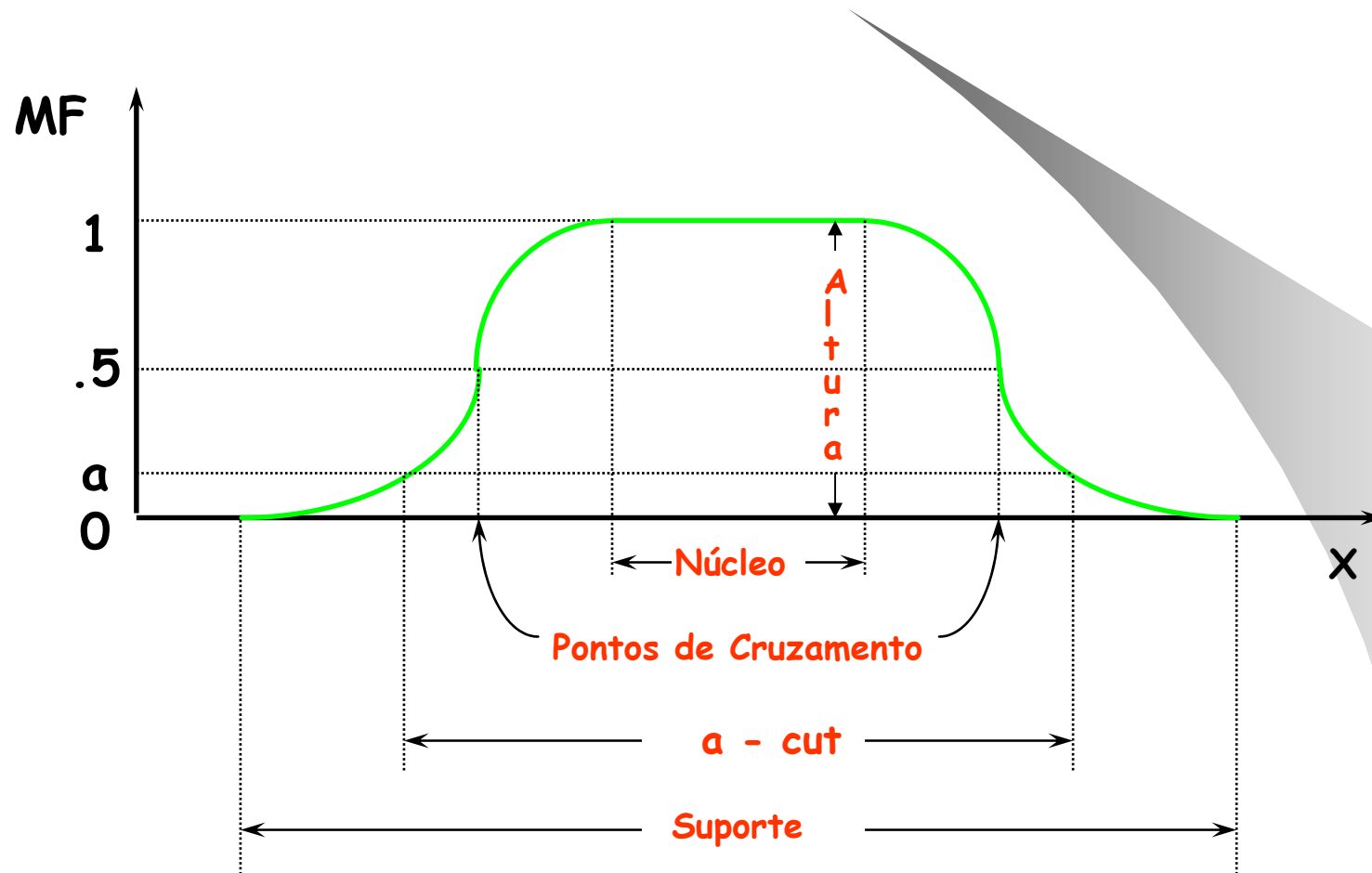
Mais definições

- ♦ Normalidade
- ♦ Altura
- ♦ Suporte
- ♦ Núcleo

- ♦ Pontos de Cruzamento
- ♦ α -cut
- ♦ strong α -cut

- ⊠ $\mu_{A(x)} = 1$
- ⊠ $\text{Height}_{(A)} = \text{Max}_x \mu_{A(x)}$
- ⊠ $\text{Supp}_{(A)} = \{x \mid \mu_{A(x)} > 0 \text{ e } x \in X\}$
- ⊠ $\text{Core}_{(A)} = \{x \mid \mu_{A(x)} = 1 \text{ e } x \in X\}$
- ⊠ $\text{Crossover}_{(A)} = \{x \mid \mu_{A(x)} = 0.5\}$
- ⊠ $A_\alpha = \{x \mid \mu_{A(x)} \geq \alpha, x \in X\}$
- ⊠ $A_{\alpha+} = \{x \mid \mu_{A(x)} > \alpha, x \in X\}$

Terminologia



Mais definições

♦ Conjunto nulo

$$\boxed{\times} \emptyset$$

♦ Força

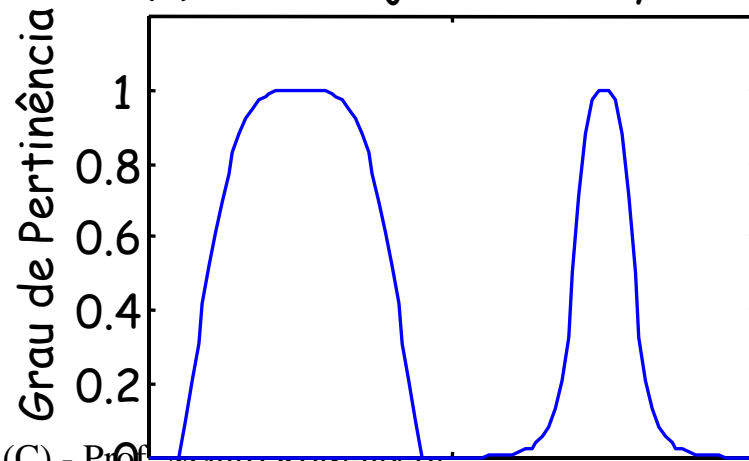
$$\boxed{\times} \sum_{i=1}^n \mu_{A(x_i)}$$

♦ Convexidade

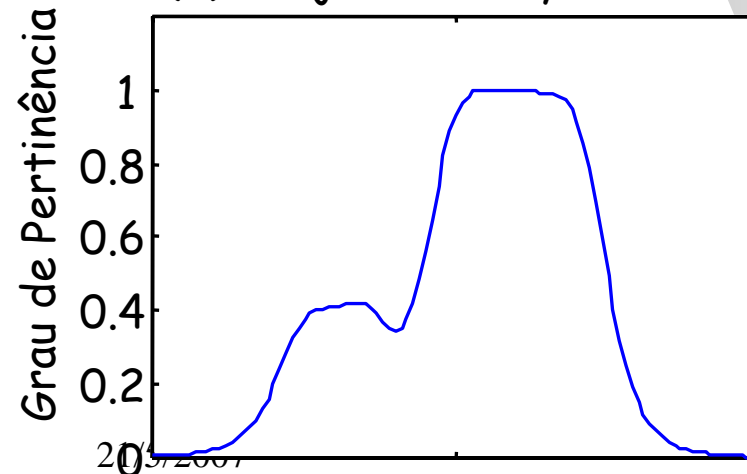
♦ Simetria

♦ Open left or right,
closed

(a) Dois conjuntos Fuzzy convexos



(b) Conjunto Fuzzy não-convexo



Exemplo:

- ♦ $X = \{a, b, c, d, e\}$
 - $A = \{1/a, 0.3/b, 0.2/c, 0.8/d, 0/e\}$ (normal)
 - $B = \{0.6/a, 0.9/b, 0.1/c, 0.3/d, 0.2/e\}$ (subnormal)
- $\text{Height}_{(A)} = 1$ e $\text{Height}_{(B)} = 0.9$
- $\text{Supp}_{(A)} = \{a, b, c, d\}$ e $\text{Supp}_{(B)} = \{a, b, c, d, e\}$
- $\text{Core}_{(A)} = \{a\}$ e $\text{Core}_{(B)} = \emptyset$

Exemplo (α -cut)

♦ $A = \{0.3/a, 1/b, 0.5/c, 0.9/d, 1/e\}$

• para $0.3 \geq \alpha \geq 0$

$$A_{\alpha} = \{a, b, c, d, e\}$$

• para $0.5 \geq \alpha > 0.3$

$$A_{\alpha} = \{b, c, d, e\}$$

• para $0.9 \geq \alpha > 0.5$

$$A_{\alpha} = \{b, d, e\}$$

Operações Básicas

- ♦ Subconjunto
- ♦ Igualdade
- ♦ Complemento
- ♦ Complemento Relativo

$$\boxtimes A \subset B, \text{ se } \mu_{B(x)} \geq \mu_{A(x)} \text{ para cada } x \in X$$

$$\boxtimes A = B, \text{ se } \mu_{A(x)} = \mu_{B(x)} \text{ para cada } x \in X$$

$$\boxtimes \sim A = X - A \rightarrow \mu_{\sim A(x)} = 1 - \mu_{A(x)}$$

$$\boxtimes \mu_{E(x)} = \text{Max} [0, \mu_{A(x)} - \mu_{B(x)}]$$

- ♦ União

$$\boxtimes C = A \cup B \rightarrow \mu_{C(x)} = \max(\mu_{A(x)}, \mu_{B(x)})$$

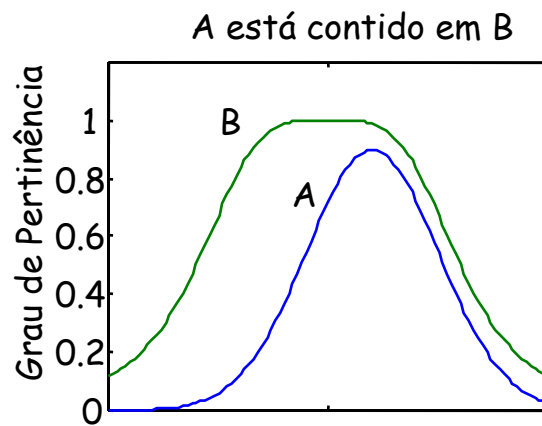
$$\boxtimes C = \mu_{A(x)} \vee \mu_{B(x)}$$

- ♦ Interseção

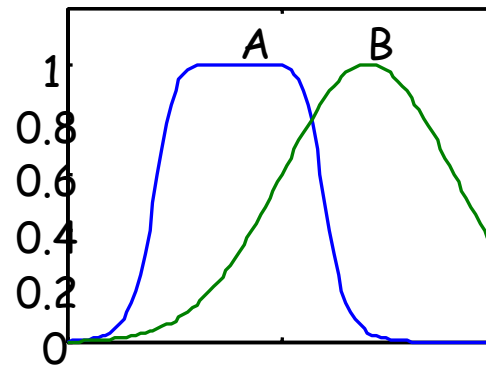
$$\boxtimes C = A \wedge B \rightarrow \mu_{C(x)} = \min(\mu_{A(x)}, \mu_{B(x)})$$

$$\boxtimes C = \mu_{A(x)} \wedge \mu_{B(x)}$$

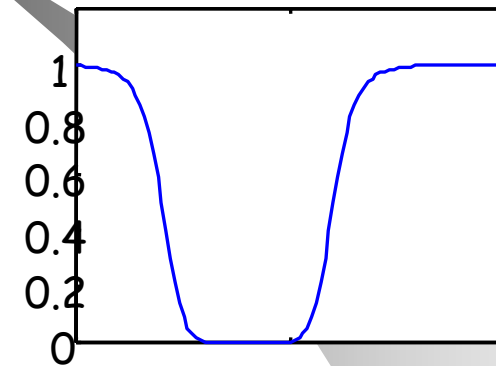
Representação



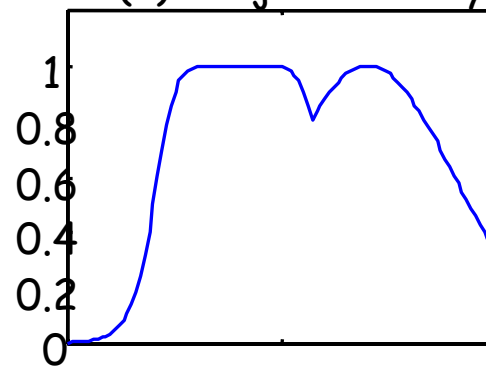
(a) Conjuntos Fuzzy A e B



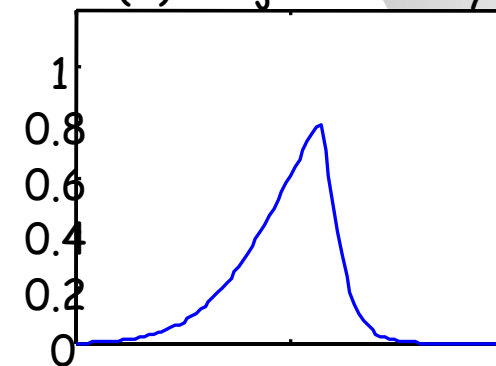
(b) Conjunto Fuzzy não "A"



(c) Conjunto Fuzzy "A ou B"



(d) Conjunto Fuzzy "A e B"



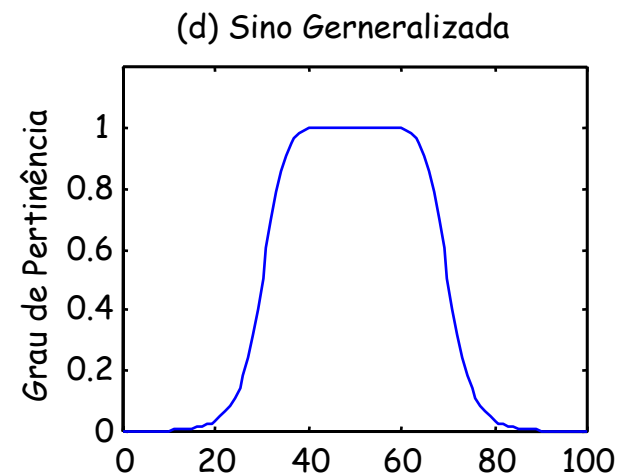
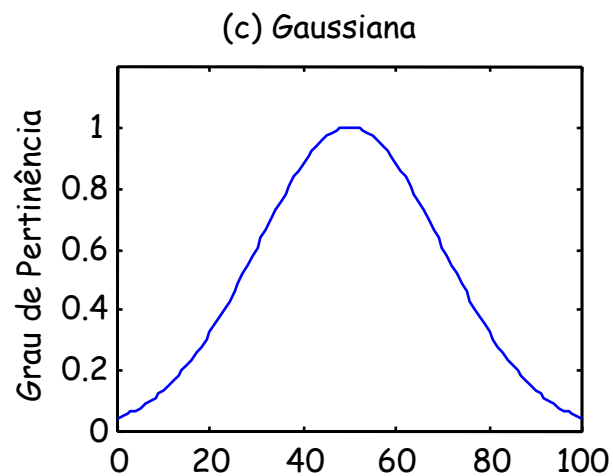
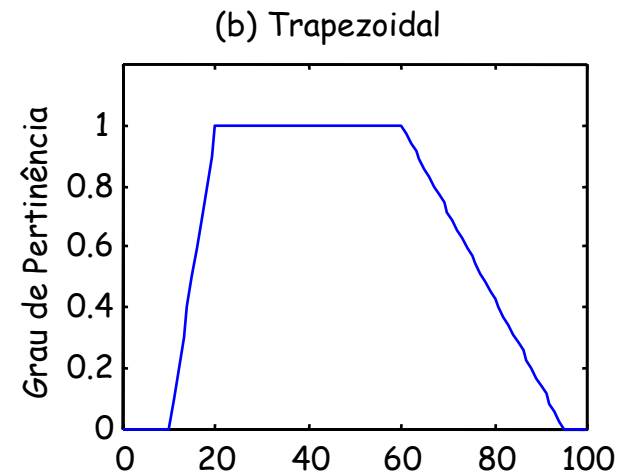
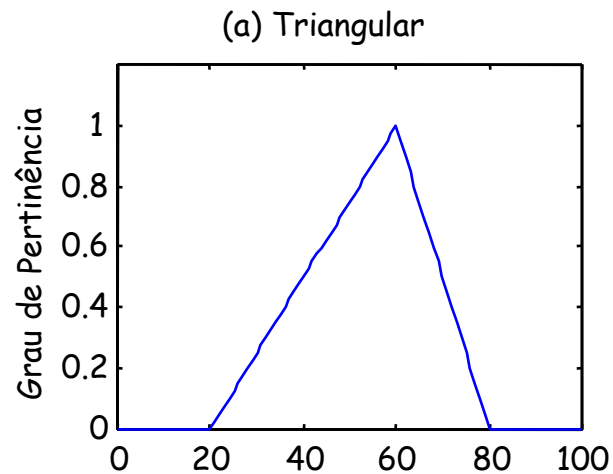
Exemplo (União|Interseção)

- ♦ $X = \{a, b, c, d, e\}$
 - $A = \{1/a, 0.7/b, 0.3/c, 0/d, 0.9/e\}$
 - $B = \{0.2/a, 0.9/b, 0.4/c, 1/d, 0.4/e\}$
 - $C = \{1/a, 0.9/b, 0.4/c, 1/d, 0.9/e\}$
 - $D = \{0.2/a, 0.7/b, 0.3/c, 0/d, 0.4/e\}$

Formulação da MF

- ♦ Função Triangular $\text{trimf} (x; a, b, c) = \max \left(\min \left(\frac{x-a}{b-a}, \frac{c-x}{c-b} \right), 0 \right)$
- ♦ Função Trapezoidal $\text{trapmf} (x; a, b, c, d) = \max \left(\min \left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right), 0 \right)$
- ♦ Função Gaussiana $\text{gaussmf} (x; a, b, c) = e^{-\frac{1}{2} \left(\frac{x-c}{\sigma} \right)^2}$
- ♦ Função Sino Generalizada $\text{gbellmf} (x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{b} \right|^{2b}}$

Formulação da MF



Propriedades (Interseção|União)

- ♦ Comutatividade
 - $A \vee B = B \vee A$
 - $A \wedge B = B \wedge A$
- ♦ Idempotência
 - $A \vee A = A$
 - $A \wedge A = A$
- ♦ Associatividade
 - $A \vee (B \vee C) = (A \vee B) \vee C = A \vee B \vee C$
 - $A \wedge (B \wedge C) = (A \wedge B) \wedge C = A \wedge B \wedge C$
- ♦ Distributividade
 - $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
 - $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

Propriedades (Interseção|União)

- $A \vee \emptyset = A$ $A \vee X = X$
- $A \wedge \emptyset = \emptyset$ $A \wedge X = A$

- $A \subset A \vee B$
- $A \supset A \wedge B$
- $A \wedge B \subset A \vee B$

- Se $A \subset B$ então
 - $B = A \vee B$
 - $A = A \wedge B$

- Se $A \subset B$ e $B \subset C$ então
 - $A \subset C$

Propriedades (Comp. | Comp. Relativo)

- ♦ Negação Dupla
 - $\sim(\sim A) = A$
- ♦ Lei de Morgan
 - $\sim(A \vee B) = \sim A \wedge \sim B$
 - $\sim(A \wedge B) = \sim A \vee \sim B$
- ♦ $\sim \phi = X$
- ♦ $\sim X = \phi$
- ♦ Se $A \subset B$ então $\sim A \supset \sim B$ e $A - B = \phi$

- ♦ $A - A = \phi$

- ♦ $\phi - A = \phi$

- ♦ $A - \phi = A$

Uma característica
significante que
distingue os conjuntos
difusos dos conjuntos
clássicos é:

- $\sim A \wedge A \neq \phi$

- $\sim A \vee A \neq X$

Sistemas Nebulosos

- ♦ Possuem grande habilidade para modelar sistemas comerciais altamente complexos.
 - sistemas convencionais tem dificuldade em resolver problemas não-lineares complexos.
- ♦ São capazes de aproximar o comportamento do sistema
 - porque apresentam várias propriedades não-lineares e pouco compreensíveis.

Sistemas Nebulosos

- ♦ Benefícios para os especialistas:
 - habilidade em codificar o conhecimento de uma forma próxima a linguagem usada por eles.
- ♦ Mas o que faz uma pessoa ser um especialista?
 - é a capacidade em fazer diagnósticos ou recomendações em termos imprecisos.
- ♦ Sistemas *Fuzzy* capturam uma habilidade próxima do conhecimento do especialista.
- ♦ O processo de aquisição do conhecimento é:
 - mais fácil,
 - mais confiável,
 - menos propenso a falhas e ambigüidades.

Sistemas Nebulosos

- ♦ É capaz de modelar sistemas envolvendo múltiplos especialistas.
- ♦ Nos sistemas do mundo real, há vários especialistas sob um mesmo domínio.
- ♦ Representam bem a cooperação múltipla, a colaboração e os conflitos entre os especialistas.
- ♦ Um exemplo das posições dos gerentes de controle, de produção, financeiro e *marketing*.
 - Nosso preço deve ser baixo.
 - Nosso preço deve ser alto.
 - Nosso preço deve ser em torno de $2 \times \text{custo}$
 - Se o preço dos concorrentes não é muito alto então nosso preço deve ser próximo do preço deles.

Sistemas Nebulosos

- ♦ Devido aos seus benefícios, como:
 - regras próximas da linguagem natural
 - fácil manutenção
 - simplicidade estrutural
- ♦ Os modelos baseados em sistemas *Fuzzy* são validados com maior precisão.
- ♦ A confiança destes modelos cresce.

Raciocínio Nebulosos

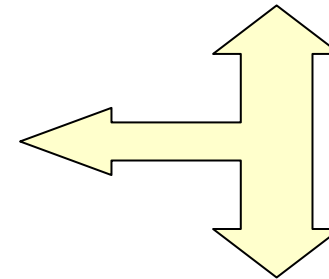
- ♦ Nos sistemas especialista convencionais:
 - as proposições são executadas seqüencialmente
 - heurísticas e algoritmos são usados para reduzir o número de regras examinadas.
- ♦ Nos sistemas especialistas *Fuzzy*.
 - o protocolo de raciocínio é um paradigma de processamento paralelo
 - todas as regras são disparadas

Etapas do Raciocínio

1ª FUZZIFICAÇÃO

AGREGAÇÃO

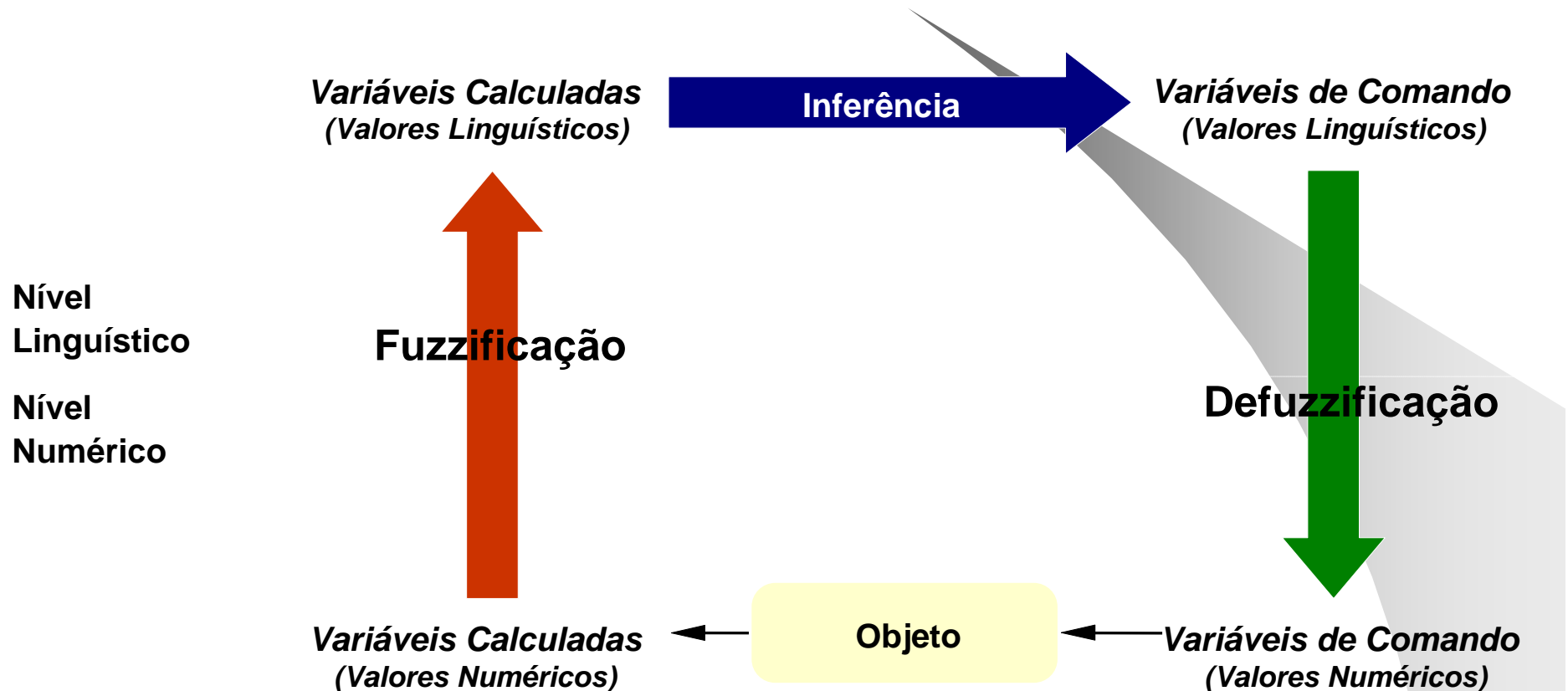
2ª INFERÊNCIA



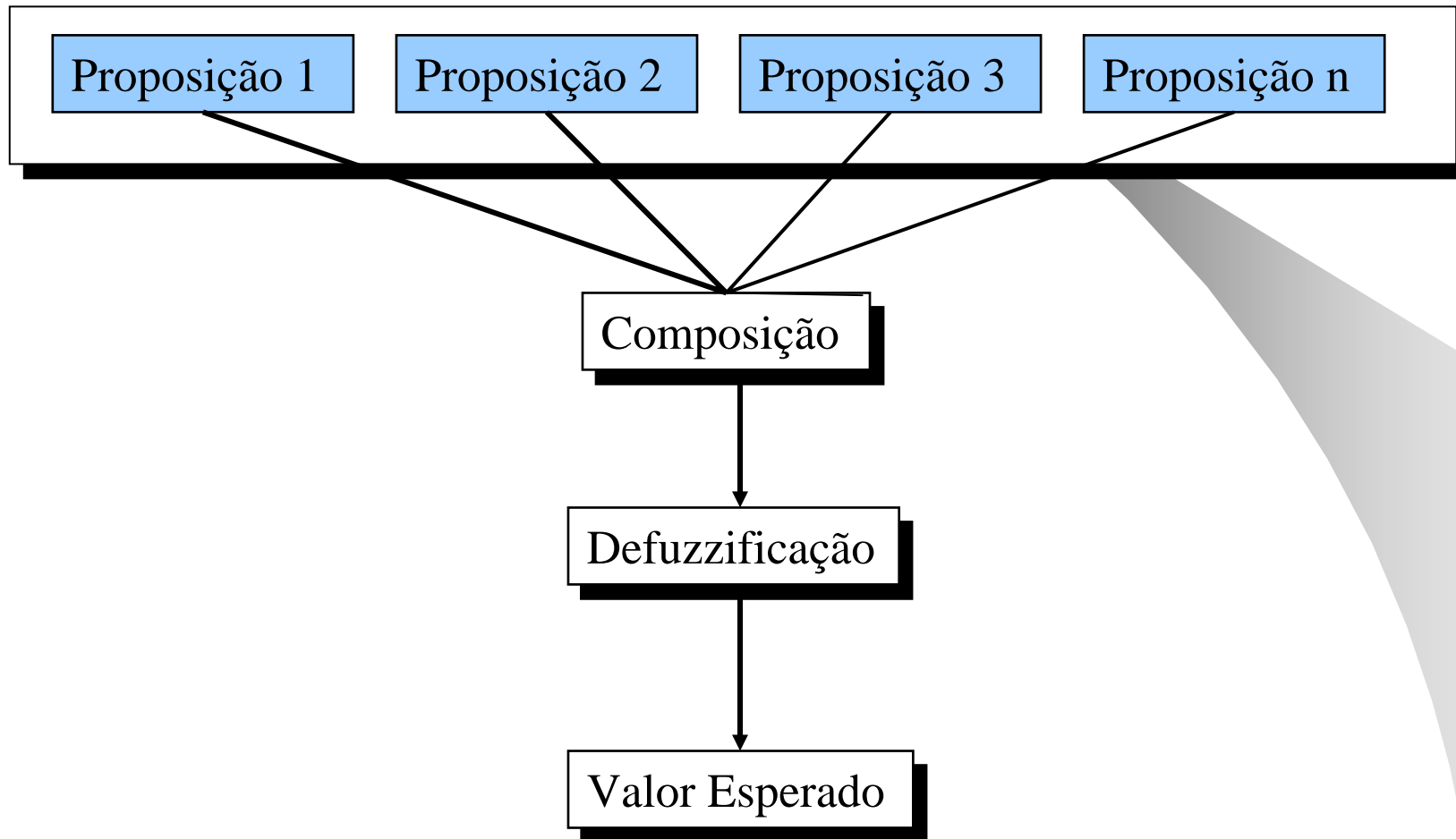
COMPOSIÇÃO

3ª DEFUZZIFICAÇÃO

Etapas do Raciocínio



Etapas do Raciocínio



Primeiro Exemplo

Objetivo do sistema: um analista de projetos de uma empresa que determina o risco de um determinado projeto.

Depende da quantidade de dinheiro e de pessoas envolvidas no projeto (variáveis de entrada)

Base de conhecimento (regras)

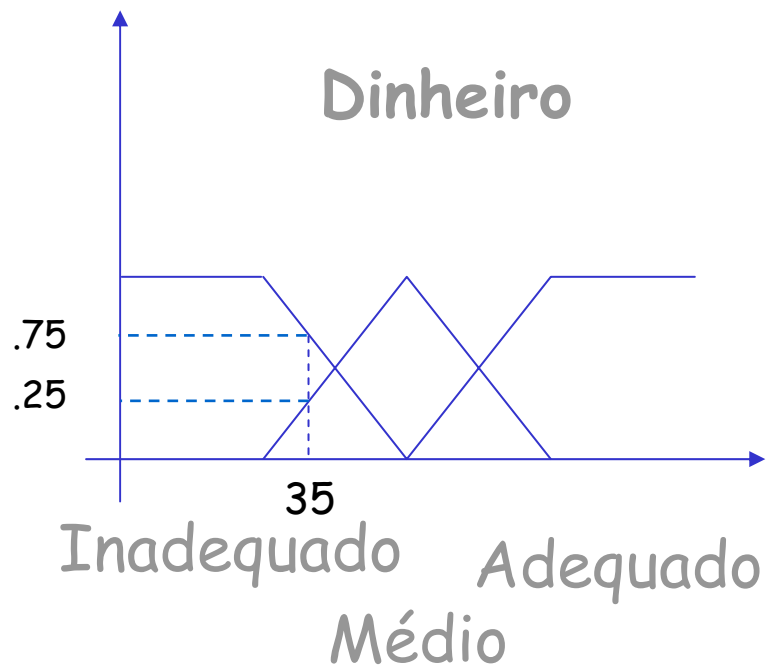
R1 - Se dinheiro é adequado ou pessoal é pequeno então risco é pequeno

R2 - Se dinheiro é médio e pessoal é alto, então risco é normal

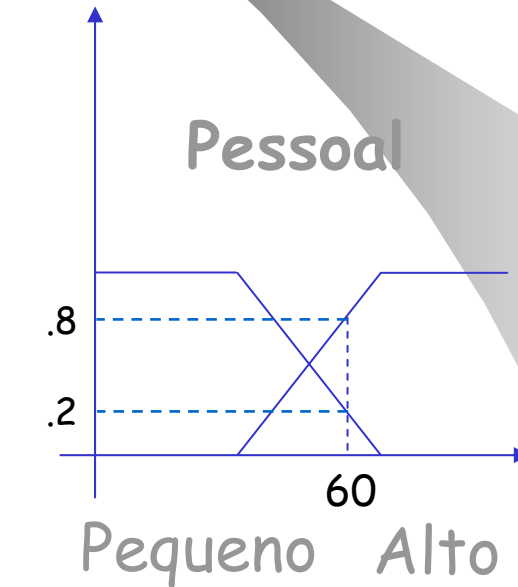
R3 - Se dinheiro é inadequado, então risco é alto

Primeiro Exemplo

- Passo 1: Fuzzificar



$$\mu_i(d) = 0,25 \& \mu_m(d) = 0,75$$

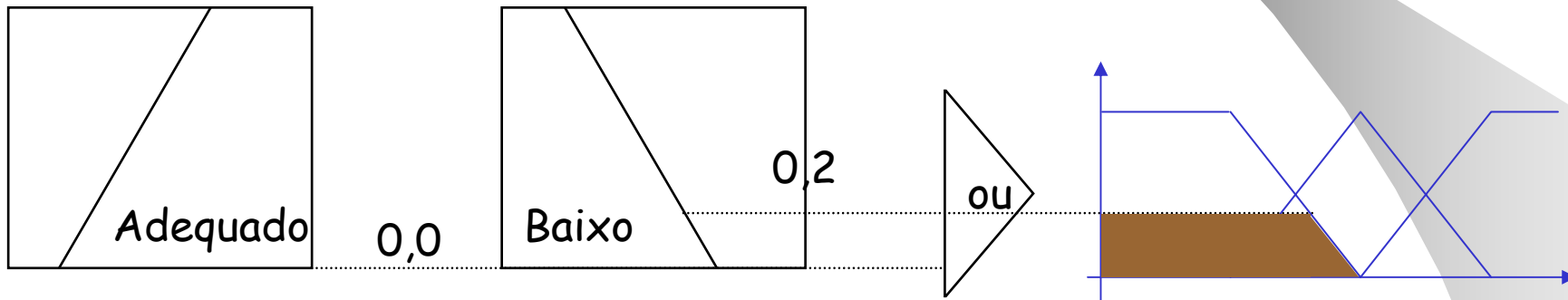


$$\mu_b(p) = 0,2 \& \mu_a(p) = 0,8$$

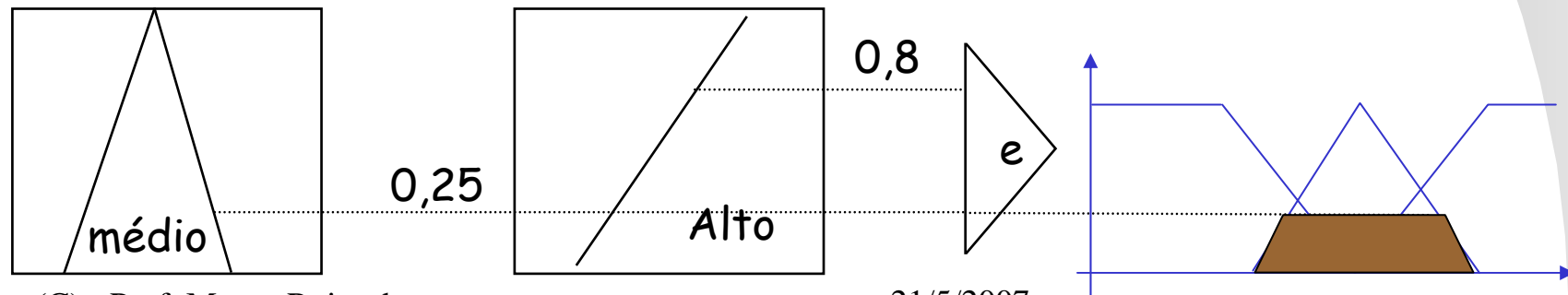
Primeiro Exemplo

- ♦ Passo 2: Avaliação das regras
 - ou \rightarrow máximo e \rightarrow mínimo

Regra 1:

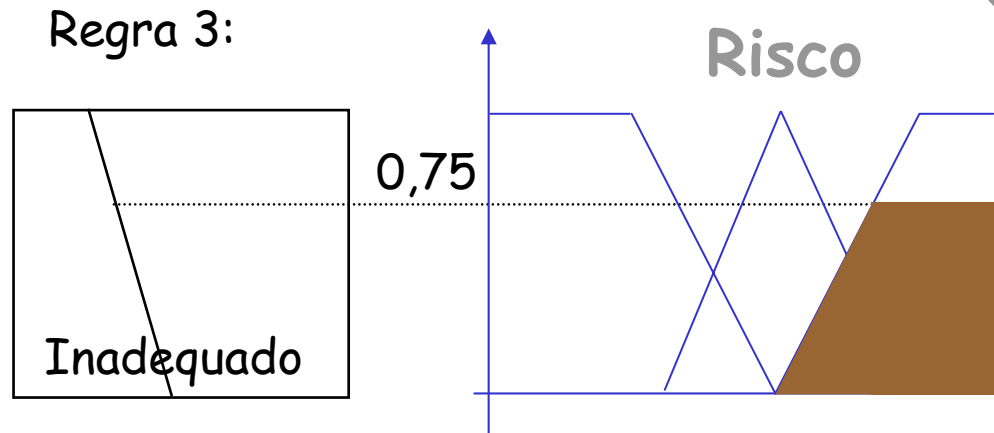


Regra 2:



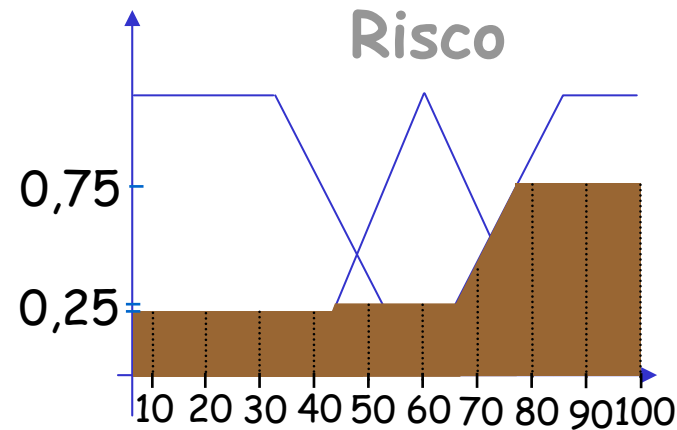
Primeiro Exemplo

• Passo 2: Avaliação das regras



Primeiro Exemplo

- Passo 3: Defuzzificação



$$C = \frac{(10+20+30+40)*0,2 + (50+60+70)*0,25 + (80+90+100)*0,75}{0,2+0,2+0,2+0,2+0,25+0,25+0,25+0,75+0,75+0,75} = \frac{267,5}{3,8} = 70,4$$

Fuzzificação

- ♦ Etapa na qual os valores numéricos são transformados em graus de pertinência para um valor lingüístico.
- ♦ Cada valor de entrada terá um grau de pertinência em cada um dos conjuntos difusos. O tipo e a quantidade de funções de pertinência usados em um sistema dependem de alguns fatores tais como: precisão, estabilidade, facilidade de implementação...

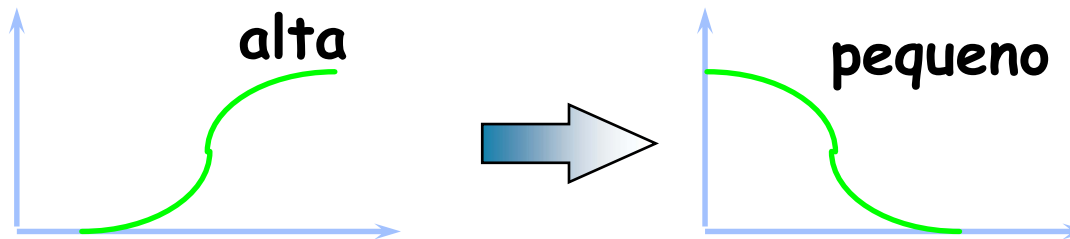
Determinação das regras

- ♦ Descrição das situações nas quais há reações através de regras de produção (If - then). Cada regra na saída especifica uma ou várias conclusões.

Regras If - then

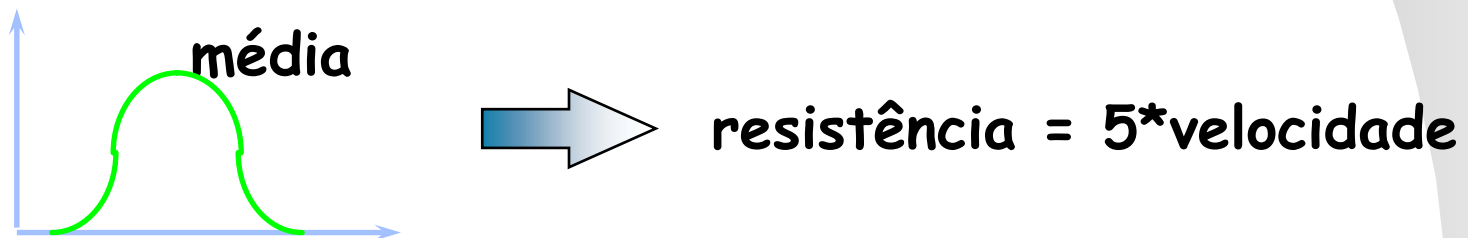
- **Estilo Mamdani**

Se a pressão é alta, então o volume é pequeno



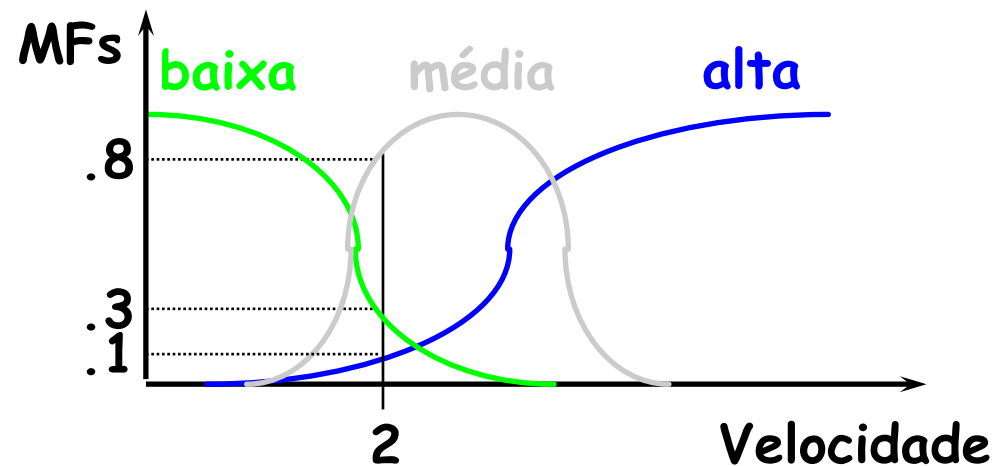
- **Estilo Sugeno**

Se a velocidade é média, então a resistência = $5 * \text{velocidade}$



Sistema de inferência

- Se velocidade é baixa então resistência = 2
- Se velocidade é média então resistência = 4 * velocidade
- Se velocidade é alta então resistência = 8 * velocidade



- Regra 1: $w1 = .3$; $r1 = 2$
- Regra 2: $w2 = .8$; $r2 = 4 * 2$
- Regra 3: $w3 = .1$; $r3 = 8 * 2$

➡ Resistência = $\frac{S(w_i * r_i)}{S w_i}$
= 7.12

Avaliação das regras

- ♦ Cada antecedente (lado if) tem um grau de pertinência. A ação da regra (lado then) representa a saída nebulosa da regra. Durante a avaliação das regras, a intensidade da saída é calculada com base nos valores dos antecedentes e então indicadas pelas saídas nebulosas da regra.
 - Alguns métodos de avaliação:
 - MinMax, MaxMin, MaxProduto, MinMin, MaxMedia, MaxMax e Soma dos produtos.

Agregação das Regras

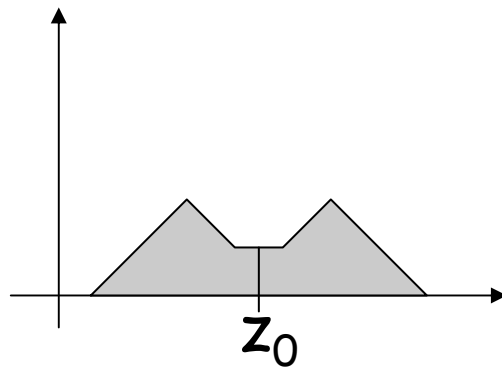
- ♦ São as técnicas utilizadas na obtenção de um conjunto difuso de saída "x" a partir da inferência nas regras.
- ♦ Determinam quanto a condição de cada regra será satisfeita.

Defuzzificação

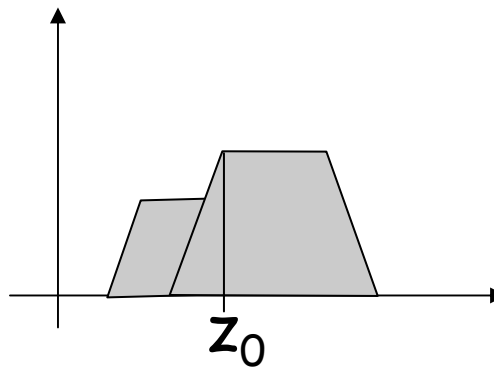
- ♦ Processo utilizado para converter o conjunto difuso de saída em um valor crisp correspondente.
 - Alguns métodos de defuzzificação:
 - Centróide,
 - Média dos máximos,
 - Distância de Hamming,
 - Barras verticais,
 - Método da altura, etc.

Defuzzificação

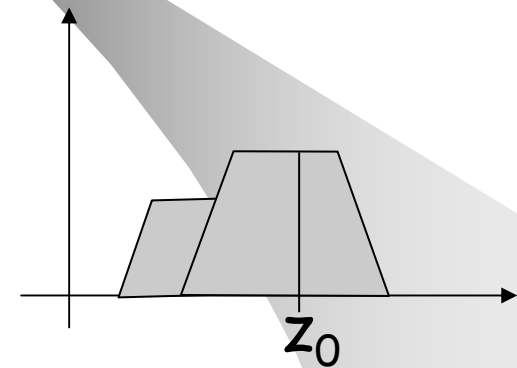
Exemplos:



Centróide



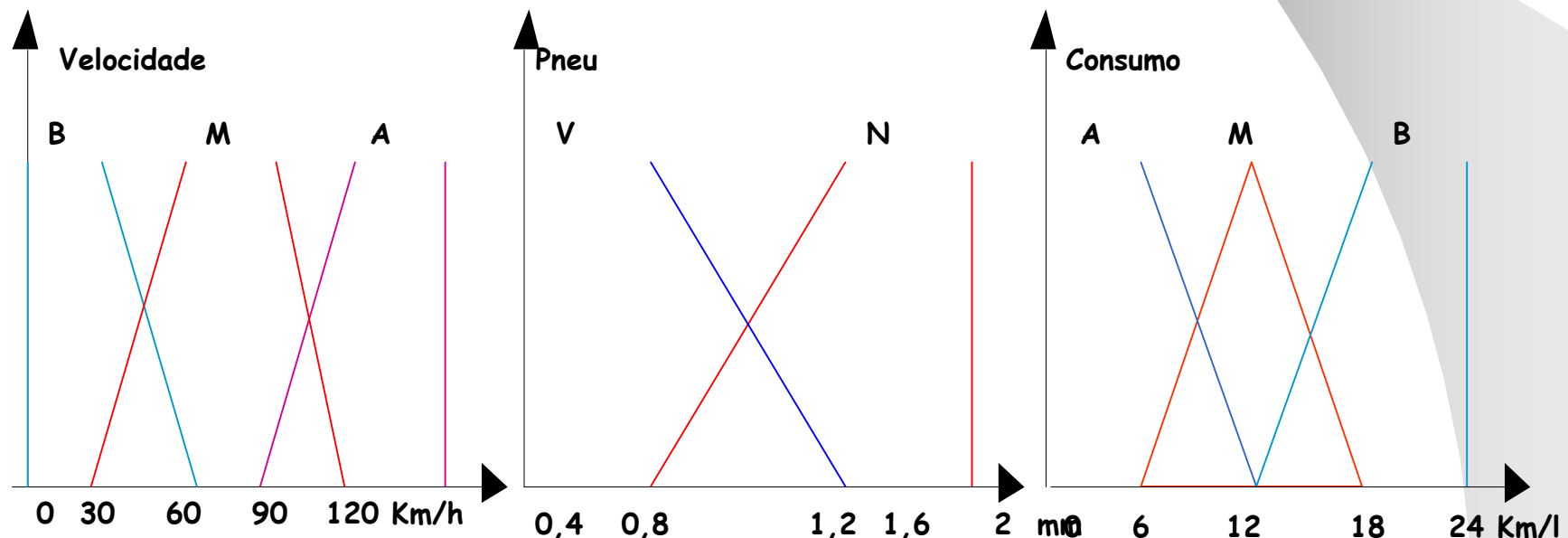
First-of-
Maxima



Critério
Máximo

Outro Exemplo

- ♦ Projeto e funcionamento de um sistema para determinação do consumo de combustível de um automóvel.
 - Passo (1): Variáveis de entrada = velocidade (Vel), pneu (Pneu)
Variável de saída = consumo (Con)
 - Passo (2): Vel = [Baixa, Média, Alta]; Pneu = [Velho, Novo]
Con = [Baixo, Médio, Alto]



Outro Exemplo

- Passo (3):

- Regra 1: Se Vel = B e Pneu = V, então Con = A.
- Regra 2: Se Vel = B e Pneu = N, então Con = M.
- Regra 3: Se Vel = M e Pneu = V, então Con = M.
- Regra 4: Se Vel = M e Pneu = N, então Con = B.
- Regra 5: Se Vel = A e Pneu = V, então Con = A.
- Regra 6: Se Vel = A e Pneu = N, então Con = M.

- Passo (4):

- Adotar centro de massa

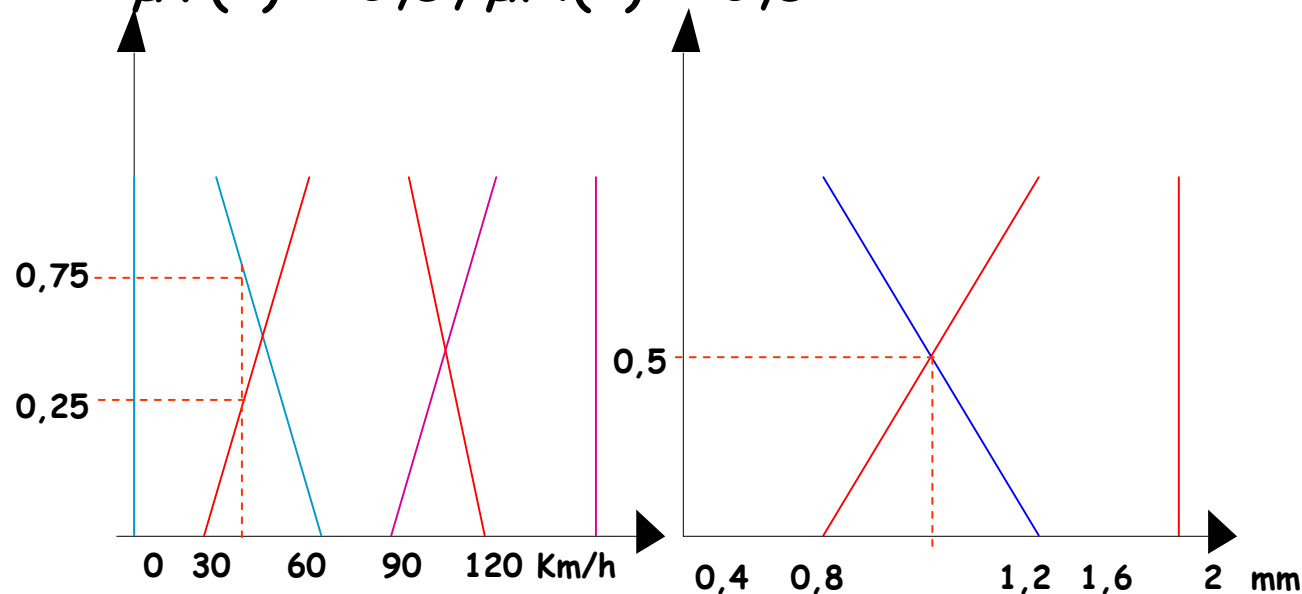
Outro Exemplo

- Para Velocidade = 35 km/h e Pneu = 1mm, qual o Consumo?

- Fuzzificação:

- $\mu_B(35) = 0,75$, $\mu_M(35) = 0,25$, $\mu_A(35) = 0,0$

- $\mu_V(1) = 0,5$, $\mu_N(1) = 0,5$



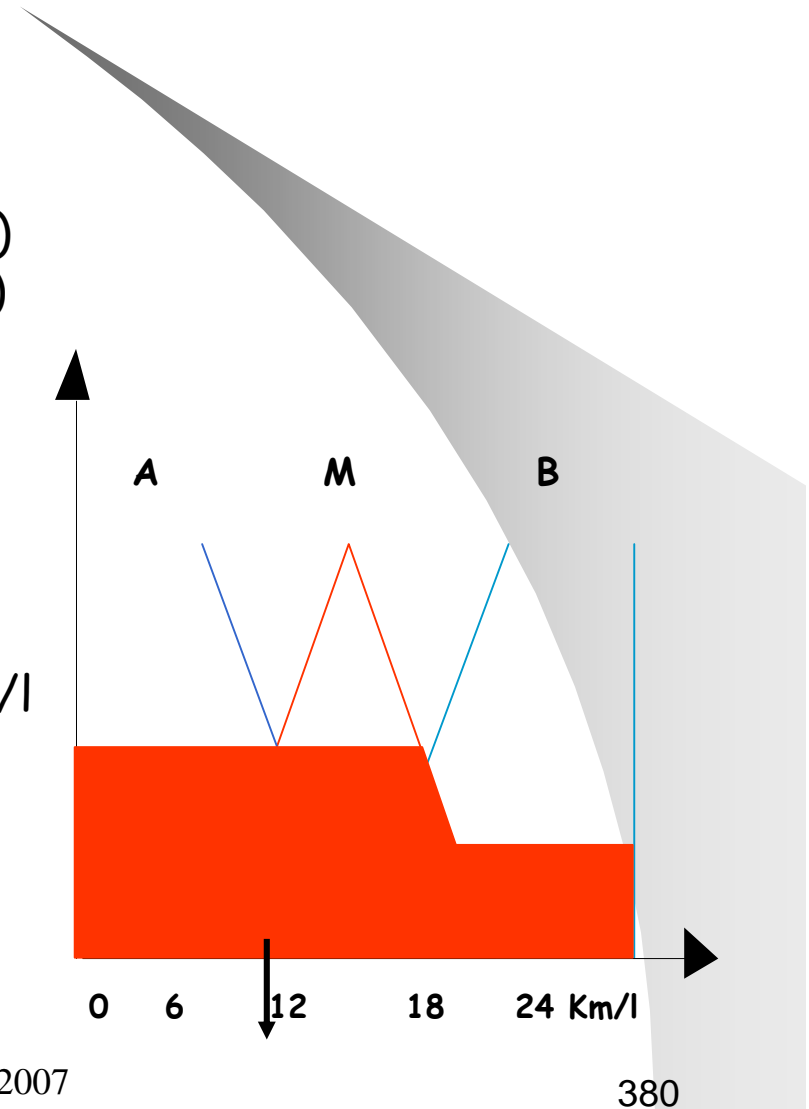
Outro Exemplo

- Inferência:

- $D1 = \min[\mu_B(35), \mu_V(1)] = 0,5$ (Con=A)
- $D2 = \min[\mu_B(35), \mu_N(1)] = 0,5$ (Con=M)
- $D3 = \min[\mu_M(35), \mu_V(1)] = 0,25$ (Con=M)
- $D4 = \min[\mu_M(35), \mu_N(1)] = 0,25$ (Con=B)
- $D5 = \min[\mu_A(35), \mu_V(1)] = 0,0$ (Con=A)
- $D6 = \min[\mu_A(35), \mu_N(1)] = 0,0$ (Con=M)
- A: $\max(0,5; 0) = 0,5$; B: $\max(0,25) = 0,25$
M: $\max(0,5; 0,25; 0) = 0,5$;

- Defuzzificação:

- Usando centro de massa: $Con \cong 11,5$ km/l



Lógica *Fuzzy* no Mundo

- Lógica *Fuzzy* tornou-se tecnologia padrão e é também aplicada em análise de dados e sinais de sensores;
- Também utiliza-se lógica fuzzy em finanças e negócios;
- Aproximadamente 1100 aplicações bem sucedidas foram publicadas em 1996; e
- Utilizada em sistemas de Máquinas Fotográficas, Máquina de Lavar Roupas, Freios ABS, Ar Condicionado e etc.

Redes Neurais

Introdução ao Estudo das Redes Neurais Artificiais

- ♦ Objetivos

- Oferecer ao aluno uma introdução à abordagem da IA conexionista, descrevendo características de funcionamento, formas de aprendizado e aplicações típicas.

O que é Inteligência Artificial?

- ♦ A INTELIGÊNCIA É SÓ HUMANA?
 - Em um primeiro momento, a inteligência era geralmente associada a uma característica unicamente humana, de representação de conhecimentos e resolução de problemas, refletindo um ponto de vista altamente antropocêntrico. Mas, ainda assim, nós, humanos, não compreendemos a nós mesmos, como funciona nossa "inteligência" e nem mesmo a origem de nossos pensamentos.

O que é Inteligência Artificial?

- Hoje em dia, para muitos pesquisadores, a idéia de inteligência passou a ser associada com a idéia de sobrevivência.
- *Carne*: "Talvez a característica básica de um organismo inteligente seja sua capacidade de aprender a realizar várias funções em um ambiente dinâmico, tais como sobreviver e prosperar".
- *Fogel*: "inteligência pode ser definida como a capacidade de um sistema de adaptar seu comportamento para atingir seus objetivos em uma variedade de ambientes".

As duas abordagens da IA

- ♦ IA Simbólica

- Um sistema simbólico é capaz de manifestar um comportamento inteligente.
- O comportamento inteligente global é simulado sem considerar os mecanismos responsáveis por este comportamento.

Princípios da IA Simbólica

- ♦ A estratégia fundamental que sustentou boa parte do sucesso inicial da IA Simbólica, se deve à proposta conhecida como "Physical Symbol Systems Hypothesis", de Newell e Simon.
- ♦ **Physical Symbol Systems - Newell & Simon(1976)**

"A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure)...the system also includes a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves "

As duas abordagens da IA

- ♦ IA Conexionista

- Se for construído um modelo suficientemente preciso do cérebro, este modelo apresentará um comportamento inteligente. Se apenas uma pequena parte do cérebro for reproduzida, a função exercida por esta parte emergirá do modelo.

Computação Baseada em Instruções

- ♦ Arquiteturas Von-Neuman, Máquinas Hiper-Cúbicas, Máquinas Sistólicas, Data-flow.
- ♦ Se baseiam na execução de instruções para realização do processamento desejado.
- ♦ Adotam uma abordagem algorítmica para a solução de problemas.
- ♦ CBI - "Computadores Baseados em Instruções".
- ♦ A abordagem algorítmica para solução de problemas pode ser extremamente eficiente desde que se conheça exatamente a seqüência de instruções a serem executadas para resolução do problema.

Computação "Neural"

- ♦ Existem uma série de problemas que os seres vivos, e os seres humanos em particular, parecem resolver de maneira inata.
- ♦ O processamento de imagens, o reconhecimento da fala, a recuperação de informações de maneira associativa, a filtragem adaptativa de sinais, o aprendizado de novos fatos e idéias, etc.
- ♦ **Se o cérebro dos seres vivos parece ser adequado para resolver os problemas não algorítmicos, deve se buscar uma abordagem que procure se inspirar no funcionamento do cérebro para solução dos problemas.**

IA Simbólica X IA Conexionista

- ♦ Conhecimento representado por regras (ou outra estrutura similar) que podem ser facilmente tratadas e analisadas.
- ♦ Permite a explicação do processo que levou a uma determinada resposta.
- ♦ Fácil inserção de novos conhecimentos obtidos a partir do especialista ou através de métodos automáticos de aquisição de conhecimento.

IA Simbólica X IA Conexionista

- ♦ Necessidade de se trabalhar com conhecimentos completos e exatos sobre um determinado problema.
- ♦ Dificuldade de explicar todos os conhecimentos relativos ao problema através de regras simbólicas.
- ♦ Dificuldade para tratar informações imprecisas ou aproximadas, e valores numéricos (dados quantitativos).
- ♦ Exemplo: regular a temperatura da água do banho.

IA Simbólica X IA Conexionista

- ♦ Outro Exemplo:
Conhecimento Teórico
 - $AND(A,B) = \text{if } A=0$
 then $AND=0$
 else if $B=0$
 then $AND=0$
 else $AND=1$Conhecimento Empírico

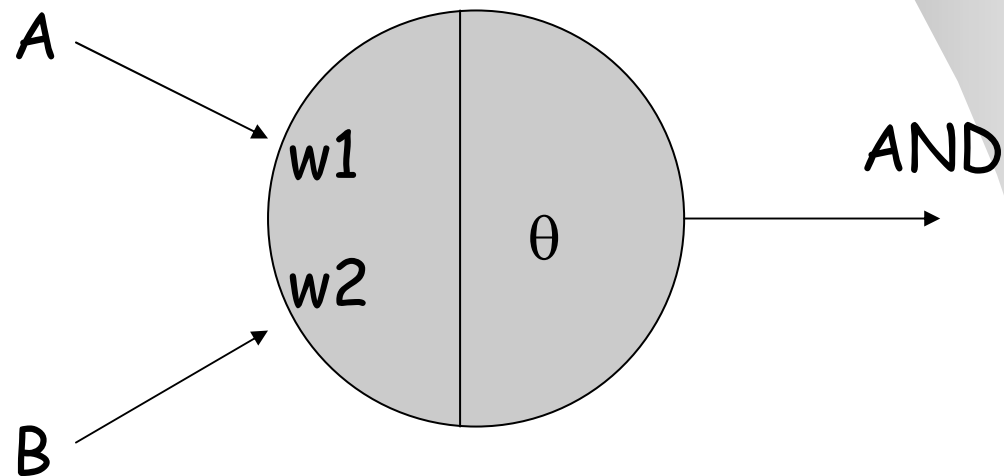
A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

IA Simbólica X IA Conexionista

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{aligned}w1 &= 1 \\w2 &= 1 \\ \theta &= 2\end{aligned}$$

$$A.w1 + B.w2 \geq \theta$$



Interesses em usar Redes Neurais

- ♦ Psicólogos

- Estão vendo possibilidades de construir redes neurais artificiais e ver aparecer comportamentos emergentes tais como o aprendizado de novos conceitos, ajudando a compreensão dos mecanismos do aprendizado.

- ♦ Neurofisiologistas

- Estão interessados em ver as RNAs como metáfora cerebral e, por simulação, melhorar o conhecimento dos mecanismos cerebrais. (estudar a capacidade de memorização, por exemplo).

Interesses em usar Redes Neurais

- ♦ Cientistas Cognitivos

- Se empenham em usar as redes neurais artificiais para um melhor conhecimento dos mecanismos envolvidos no processo cognitivo (qual o melhor método de aprendizado?).

- ♦ Engenheiros

- Olham as RNAs como um caminho para, implementando estas redes em circuitos elétricos, ter computadores realmente paralelos e distribuídos.
- Muitos encontraram no aprendizado de RNAs um campo para aplicar o que se conhece da teoria da otimização.

Interesses em usar Redes Neurais

- ♦ Cientistas de Computação
 - Encontraram um novo paradigma de programação e uma arquitetura distribuída. Explorar este paradigma, analisando suas capacidades e complexidades computacionais, desenvolvendo técnicas de programação, aplicações, etc. é um DESAFIO.

Novo Paradigma de Programação

- ♦ Paradigmas de Programação:
 - Imperativa (PASCAL, C, BASIC).
 - Declarativa (PROLOG).
 - Funcional (LISP).
 - Conexionista.
 - Programação por exemplos,
 - Capacidade de generalização,
 - Usa analogia com problemas anteriormente resolvidos,
 - Não necessita de algoritmo explícito,
 - Não necessita de descrição do problema,
 - Baseada na adaptabilidade!

Fundamentos Biológicos

♦ O Sistema Nervoso

- Controla as reações rápidas do corpo, como uma contração muscular (função motora), geralmente como resposta a algum estímulo recebido (função sensora).
- Unidade fundamental: célula nervosa (neurônio)
 - Distingue-se das outras células por apresentar excitabilidade.
 - Possibilita transmissão de impulsos nervosos a outros neurônios e a células musculares
- Recebe informações dos sensores, combina estas informações com as informações armazenadas para produzir uma resposta.
- É organizado hierarquicamente.
 - Medula da coluna vertebral: ato reflexo.
 - Cérebro:
Bulbo, Mesencéfalo, Hipotálamo, Tálamo, Cerebelo.
Córtex.

Fundamentos Biológicos

♦ O Cérebro

- Composto por Neurônios
(10^5 na mosca da fruta, 5×10^6 no rato e 10^{11} no homem)

♦ O Cérebro Humano

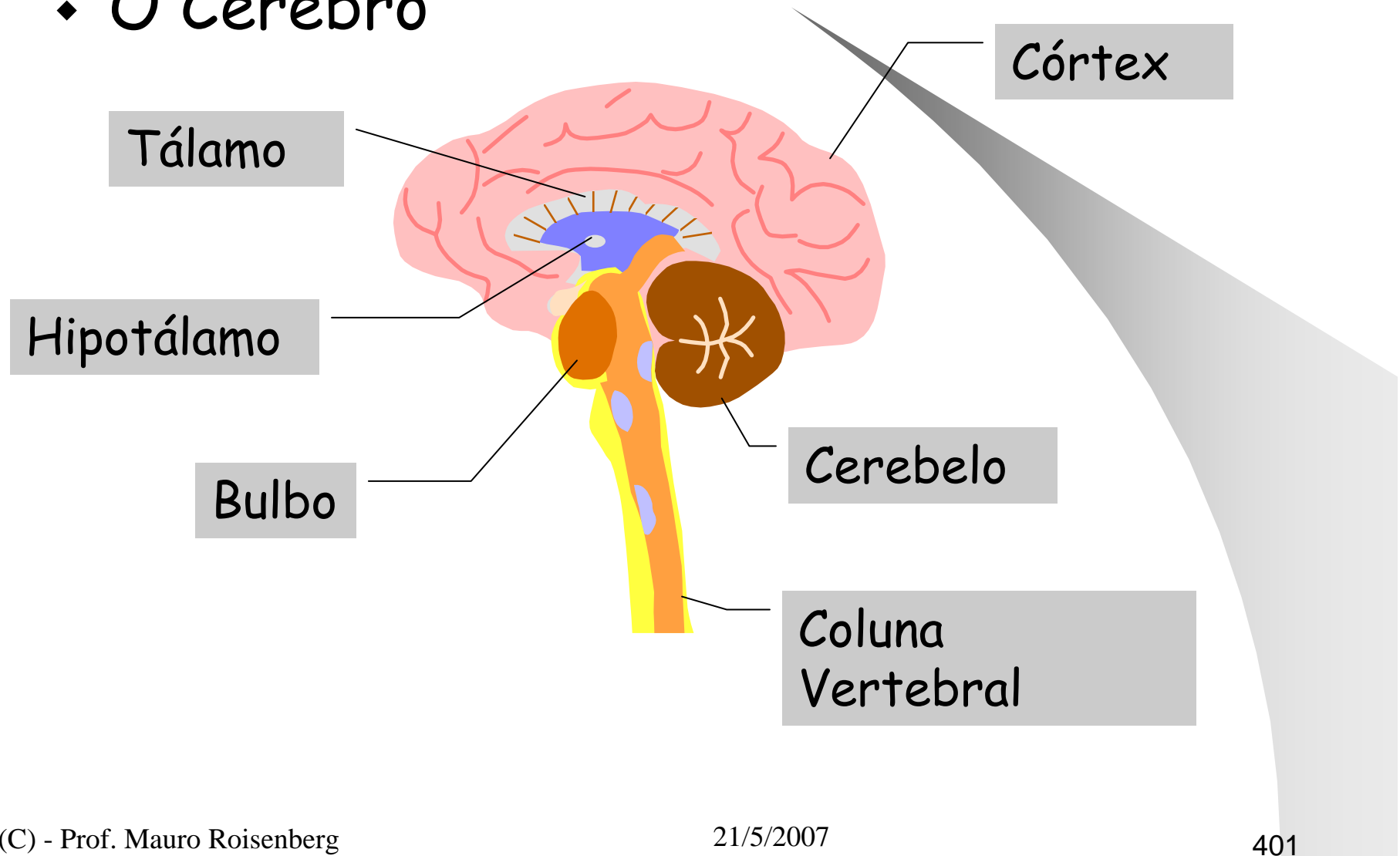
- Massa: 1-2kg no adulto - 2% do peso
20% do peso do recém-nascido.
- Usa 20% do oxigênio, 25% da glucose, 15% do fluxo de sangue.

♦ O Córtex

- O tamanho do córtex separa os humanos das outras espécies.
(5cm^2 no rato, 500cm^2 no chimpanzé e 2000cm^2 no homem)
- 3×10^{10} neurônios no córtex humano.
- 10^3 a 10^4 sinapses por neurônio.

Fundamentos Biológicos

- ♦ O Cérebro

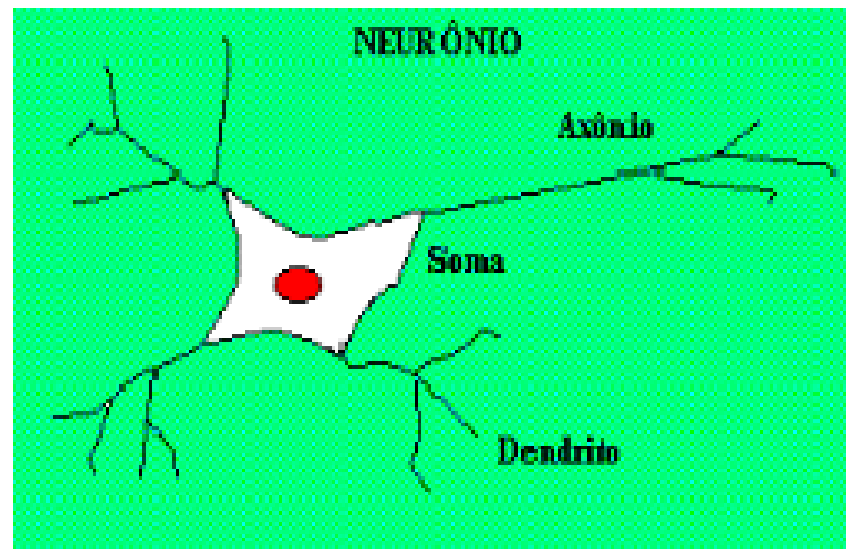


Fundamentos Biológicos

- ♦ Funciona de forma inteiramente diferente dos computadores convencionais
 - Neurônios são 100 mil a 1 milhão de vezes mais lentos que portas lógicas de silício
 - Lentidão compensada por grande número de neurônios massivamente conectados
- ♦ Para certas operações, muito mais rápido que computadores convencionais
 - Visão, audição, controle, previsão, etc
- ♦ Adaptável
- ♦ Tolerante a falhas
- ♦ Redundância
 - 10 a 100 bilhões de neurônios, cada um conectado a até 10.000 outros neurônios

Fundamentos Biológicos

- ♦ Esquema Simplificado de um Neurônio
 - Existem neurônios com vários formatos e tamanhos diferentes



Fundamentos Biológicos

♦ O Neurônio

- Um neurônio típico é composto por um corpo celular, um axônio tubular e várias ramificações arbóreas conhecidas como dendritos.
- Os dendritos formam uma malha de filamentos finíssimos ao redor do neurônio.
- O axônio é essencialmente um tubo longo e fino que ao final se divide em ramos que terminam em pequenos bulbos que quase tocam os dendritos dos outros neurônios.
- O pequeno espaço entre o fim do bulbo e o dendrito é conhecido como sinapse, através da qual as informações se propagam.

Fundamentos Biológicos

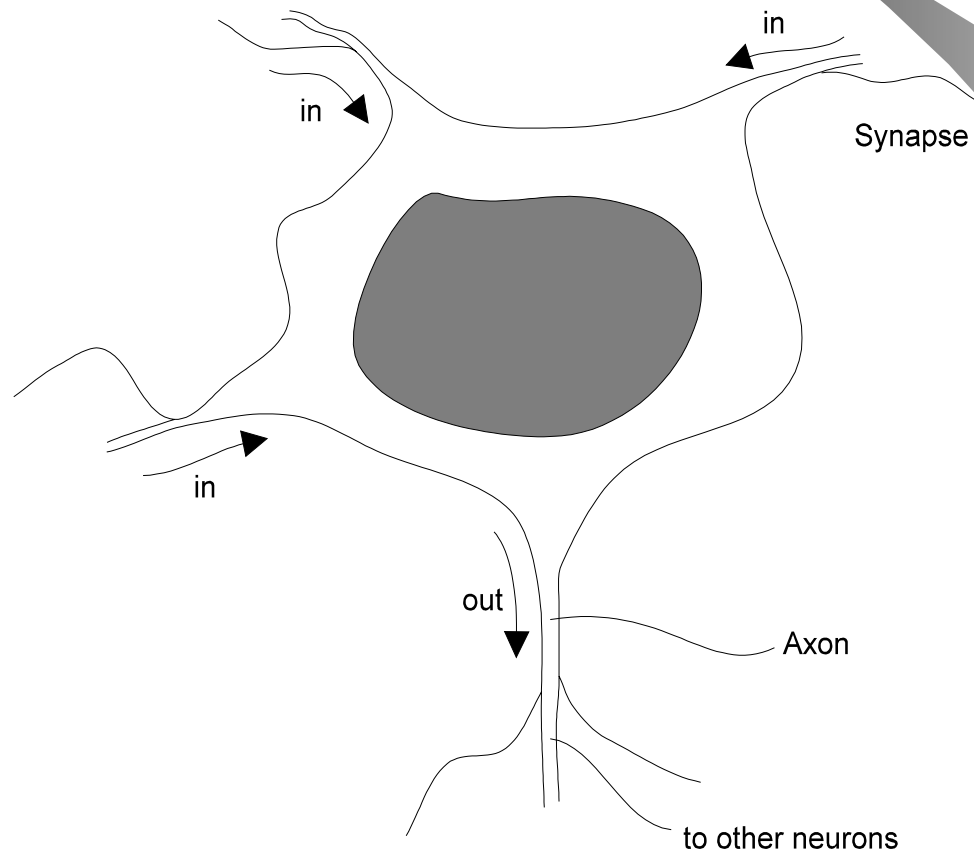
- ♦ O Funcionamento do Neurônio - O Potencial de Ação
 - Concentrações diferentes de Na^+ e K^+ dentro e fora das células provocam diferença de potencial.
 - Estimulação elétrica, química, calor, etc. pode perturbar a membrana do neurônio alterando este potencial.
 - Após um certo tempo as coisas voltam ao normal devido ao mecanismo de transporte ativo. No entanto a onda de variação de tensão se propaga.
 - Na região junto à sinapse o potencial de ação libera neurotransmissores, provocando uma perturbação na membrana do neurônio seguinte, e o fenômeno continua.

Fundamentos Biológicos

- ♦ O Funcionamento do Neurônio - A Sinapse
 - Sinapse: é a ligação entre a terminação axônica e os dendritos e que permite a propagação dos impulsos nervosos de uma célula a outra. As sinapses podem ser excitatórias ou inibitórias.
 - As sinapses excitatórias cujos neuro-excitadores são os íons sódio permitem a passagem da informação entre os neurônios e as sinapses inibitórias, cujos neuro-bloqueadores são os íons potássio, bloqueiam a atividade da célula, impedindo ou dificultando a passagem da informação.

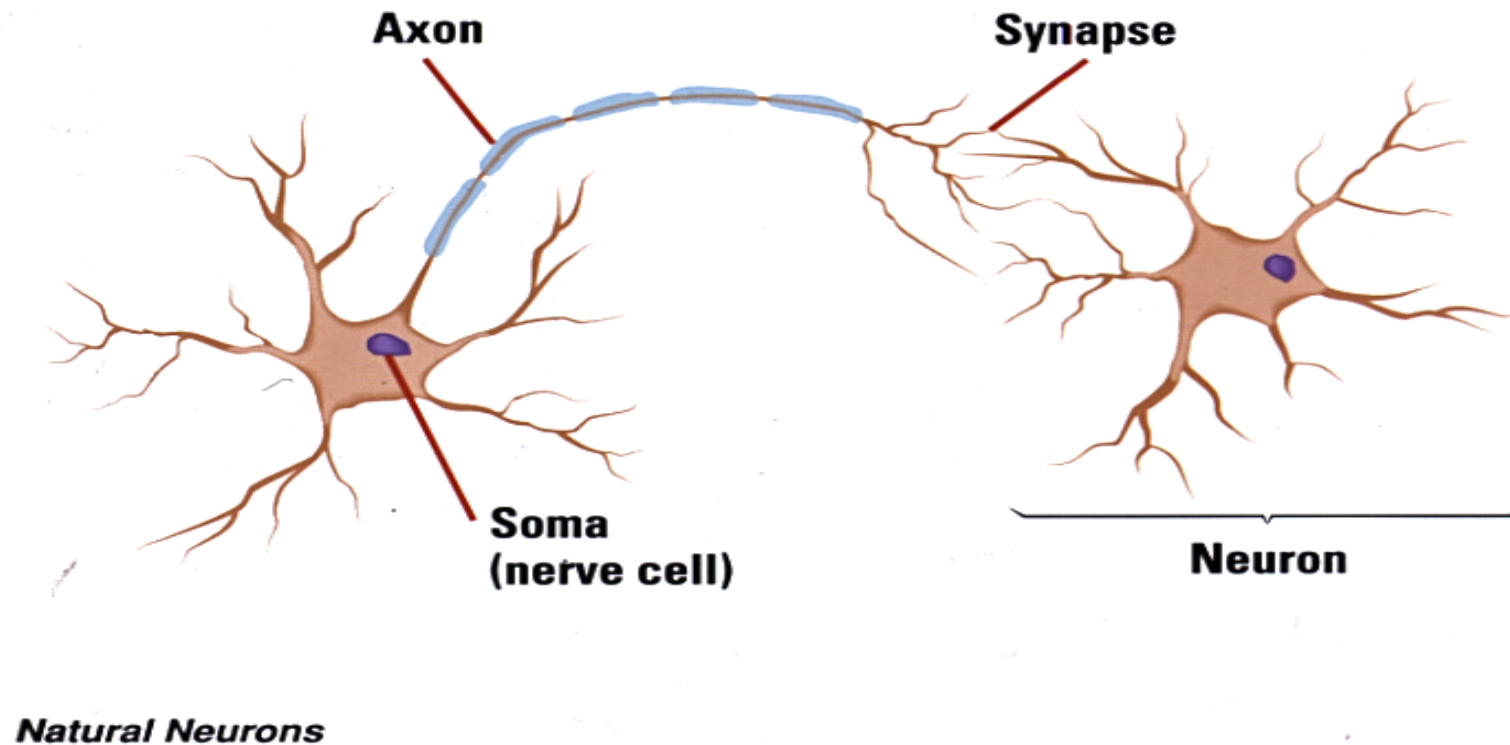
Fundamentos Biológicos

♦ O Funcionamento do Neurônio



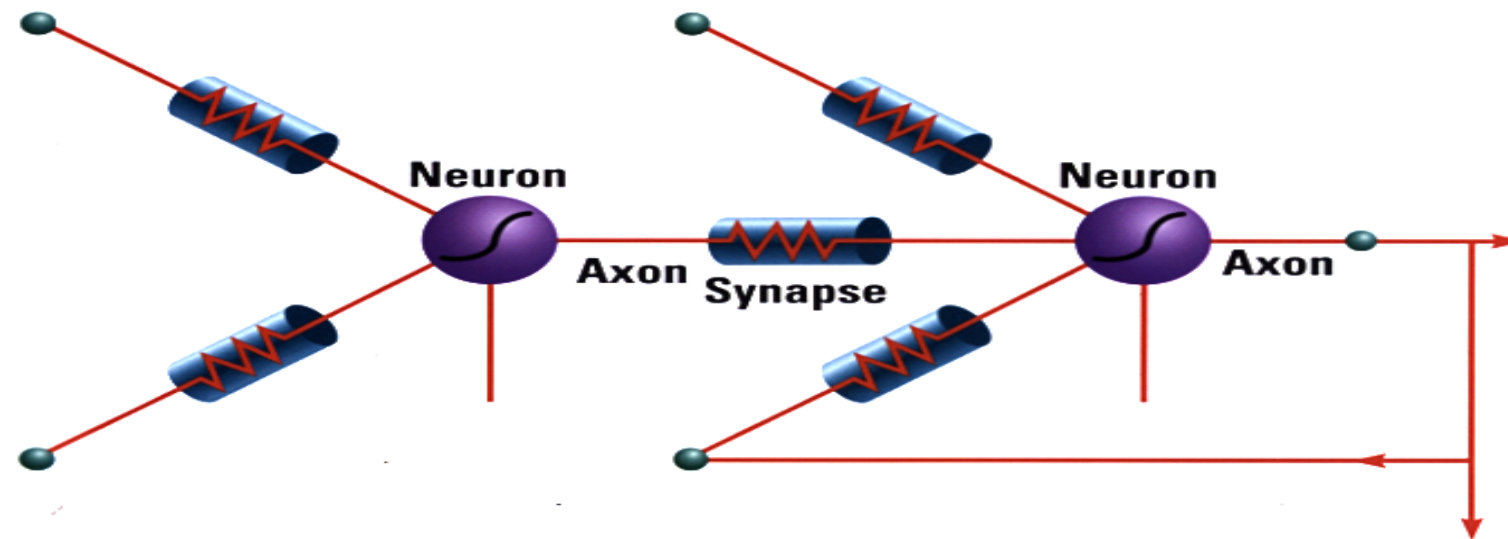
Fundamentos Biológicos

♦ O Funcionamento do Neurônio



Fundamentos Biológicos

♦ O Funcionamento do Neurônio



Artificial Neurons

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Estrutura Matricial

- Matrizes são geralmente representadas como arranjos retangulares de números escalares.
- A matriz A , $m \times n$, possui m linhas e n colunas. A notação A_{ij} é utilizada para referenciar o número da i -ésima linha, j -ésima coluna de A .

$$A = \begin{bmatrix} 4 & 2 & -5 \\ 1 & 0 & -8 \end{bmatrix} \quad A_{13} = -5$$

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Uma matriz $n \times 1$ é geralmente chamada de VETOR.
- Uma matriz com o mesmo número de linhas e colunas é chamada de MATRIZ QUADRADA.
- Uma matriz é NULA se possui todos os elementos igual a 0.
- Uma matriz é DIAGONAL se for quadrada e possuir os elementos da diagonal principal diferentes de 0 e os restantes iguais a 0.
- Uma matriz diagonal é chamada ESCALAR se todos os elementos da diagonal forem iguais.
- Uma matriz é chamada IDENTIDADE ou UNIDADE se for escalar com elementos da diagonal igual a 1.

Uma Breve Revisão de Álgebra Linear

- ♦ Definições Iniciais

$$V = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 4 & -5 \\ 1 & -8 \end{bmatrix}$$

$$Z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 4 & 0 \\ 0 & -8 \end{bmatrix}$$

$$E = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- A TRANSPOSTA de uma matriz A $m \times n$ é denotada por A^T . A^T é uma matriz $n \times m$ cujos elementos são:

$$(A^T)_{ij} = A_{ji}$$

- A transposta de um vetor coluna é chamada de VETOR LINHA.

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 2 & 1 \\ 3 & 5 \\ 4 & 6 \end{bmatrix}$$

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Soma de Matrizes

- Somar duas matrizes A e B resulta em uma matriz cujos elementos são a soma dos correspondentes elementos de A e B : Se $C = A + B$, então $C_{ij} = A_{ij} + B_{ij}$

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 4 & -3 & 0 \\ 2 & 1 & 5 \end{bmatrix}$$

$$C = A + B = \begin{bmatrix} 6 & 0 & 4 \\ 3 & 6 & 11 \end{bmatrix}$$

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Multiplicação de Matrizes

- Multiplicar uma matriz A $m \times n$ por uma matriz B $n \times p$ resulta em uma matriz C $m \times p$ cujos elementos são $C = AB$, então

$$C_{ik} = \sum_{j=1}^n A_{ij} B_{jk}$$
$$A = \begin{bmatrix} 2 & 3 \\ 4 & 1 \\ 5 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 0 & 1 & -2 \\ 1 & 2 & 3 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 11 & 6 & 11 & 11 \\ 17 & 2 & 7 & -3 \\ 22 & 4 & 11 & 0 \end{bmatrix}$$

$$C=AB = \begin{bmatrix} 2.4+3.1 & 2.0+3.2 & 2.1+3.3 & 2.-1+3.5 \\ 4.4+1.1 & 4.0+1.2 & 4.1+1.3 & 4.-2+1.5 \\ 5.4+2.1 & 5.0+2.2 & 5.1+2.3 & 5.-2+2.5 \end{bmatrix}$$

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Multiplicação de Matrizes

- Dada uma matriz quadrada A $n \times n$, verifica-se a seguinte propriedade: $A.I = I.A = A$
- A matriz quadrada A $n \times n$ é chamada INVERSÍVEL se existir uma matriz denotada por A^{-1} que satisfaz: $A.A^{-1} = A^{-1}.A = I$
- Se A^{-1} existir, ela é chamada matriz INVERSA. Se A^{-1} não existir, A é chamada matriz SINGULAR.

Uma Breve Revisão de Álgebra Linear

♦ Definições Iniciais

- Determinante de uma Matriz

- Determinante de uma matriz quadrada é um número associado à matriz dada pela definição recorrente seguinte:

dada uma matriz quadrada A $n \times n$, chama-se DETERMINANTE de A e indicamos $|A|$ ou $\det A$, ao número dado por:

a) Se $n=1$ então $|A| = a^{11}$

b) Se $n>1$ então $|A| = \sum_{j=1}^n (-1)^{1+j} a^{1j} \cdot |A^{1,j}|$

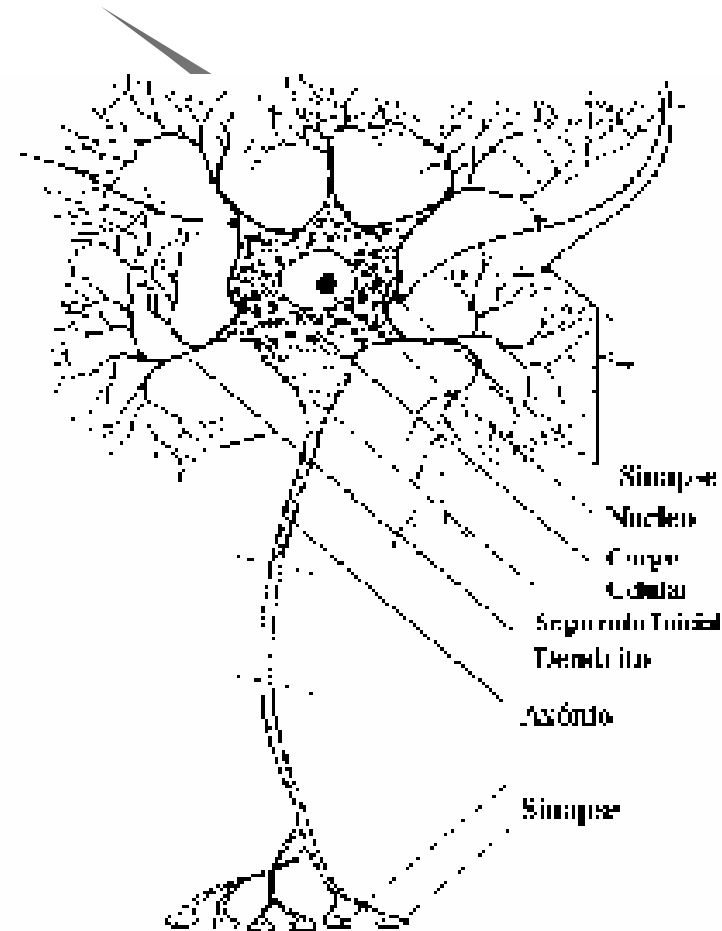
onde $A^{1,j}$ é a submatriz obtida da matriz A eliminando a linha 1 e a coluna j

- $\det A = 0$ se e somente se A não é inversível.

Redes Neurais Artificiais

Origens

- ♦ Neurônio
 - Modelo Simplificado
 - Modelo Simulado
 - Características Básicas
 - Adaptação
 - Representação de Conhecimentos baseada em conexões

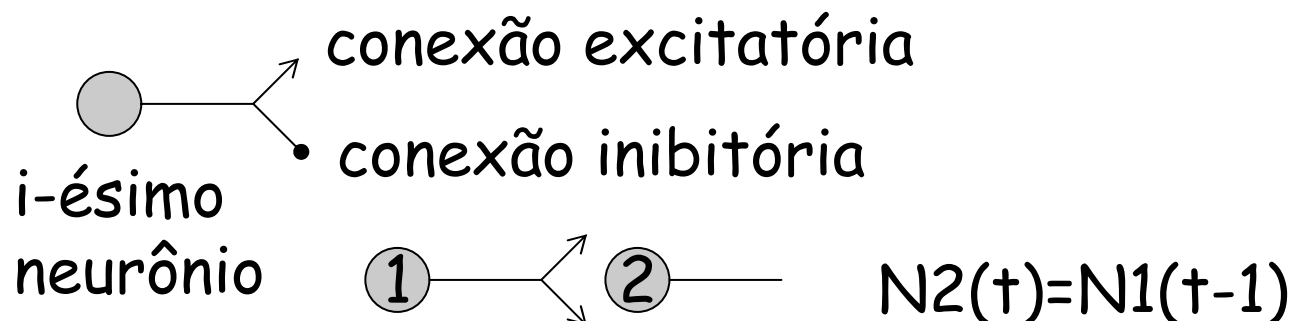


Circuitos Neurais e Computação

- ♦ O Modelo de McCulloch & Pitts (1943)
 - O Cérebro como um Sistema Computacional
 - 5 Suposições Básicas
 - A atividade de um neurônio é um processo tudo ou nada.
 - Um certo número fixo (>1) de entradas devem ser excitadas dentro de um período de adição latente para excitar um neurônio.
 - O único atraso significativo é o atraso sináptico.
 - A atividade de qualquer sinapse inibitória previne absolutamente a excitação do neurônio.
 - A estrutura das interconexões não muda com o tempo.

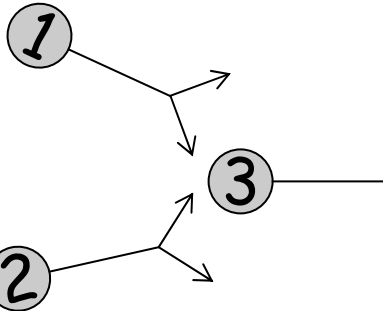
O Modelo de McCulloch & Pitts

- Comportamento da Rede Neural pode ser expresso por predicados.
- Notação:
 - $N_i(t)$: Asserção que o i -ésimo neurônio dispara no tempo t .
 - $\neg N_i(t)$: Asserção que o i -ésimo neurônio NÃO DISPARA no tempo t .
- Circuitos encontrados no SNC

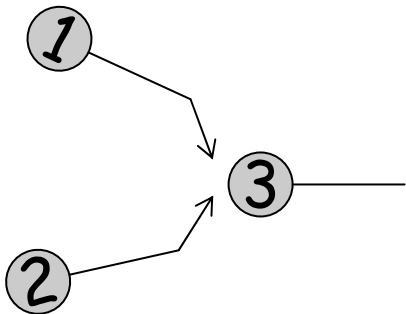


O Modelo de McCulloch & Pitts

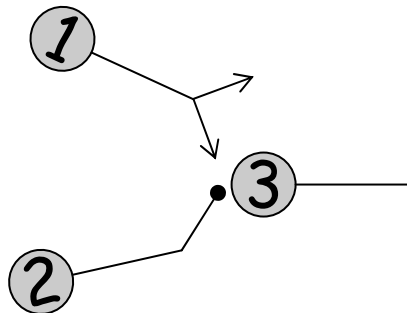
- Circuitos encontrados no SNC



$$N3(t) = N1(t-1) \text{ ou } N2(t-1) \text{ DISJUNÇÃO}$$



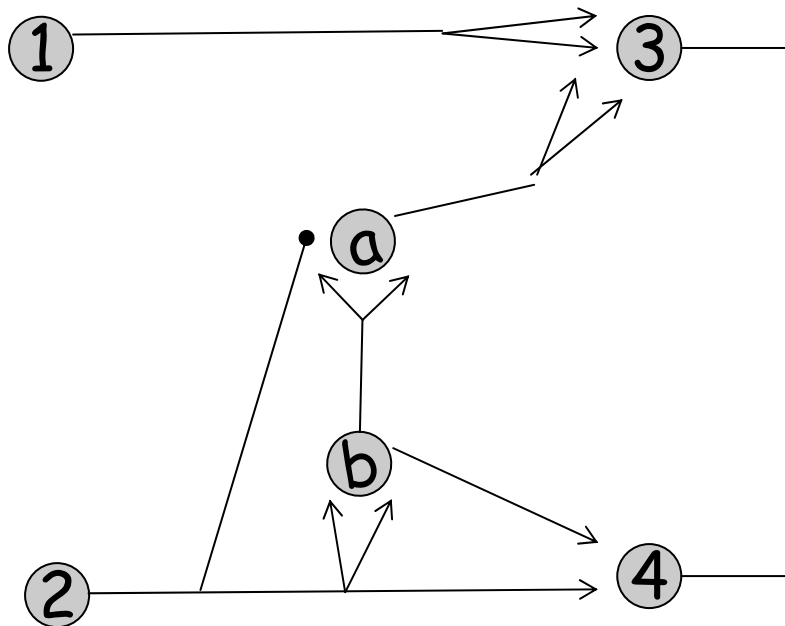
$$N3(t) = N1(t-1) \text{ e } N2(t-1) \text{ CONJUNÇÃO}$$



$$N3(t) = N1(t-1) \text{ e } \neg N2(t-1)$$

O Modelo de McCulloch & Pitts

- Exemplo



$$N3(t) = N1(t-1) \text{ ou } Na(t-1)$$

$$Na(t-1) = Nb(t-2) \text{ e } \neg N2(t-2)$$

$$Nb(t-2) = N2(t-3)$$

$$N3(t) = N1(t-1) \text{ ou } (N2(t-3) \text{ e } \neg N2(t-2))$$

$$N4(t) = N2(t-1) \text{ e } Nb(t-1)$$

$$N4(t) = N2(t-1) \text{ e } N2(t-2)$$

O Modelo de McCulloch& Pitts

- Conseqüências

+

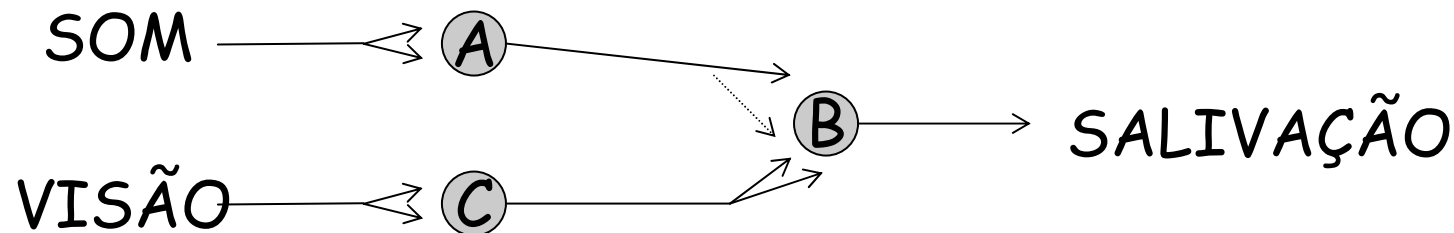
- Combinação dos neurônios implementa qualquer função lógica - Rede Neural como Computador Digital.

-

- Não explica como são formadas as topologias das Redes Neurais.
- Não explica como acontece o aprendizado.
- Rede Neural só funciona corretamente se todos os elementos funcionarem corretamente.

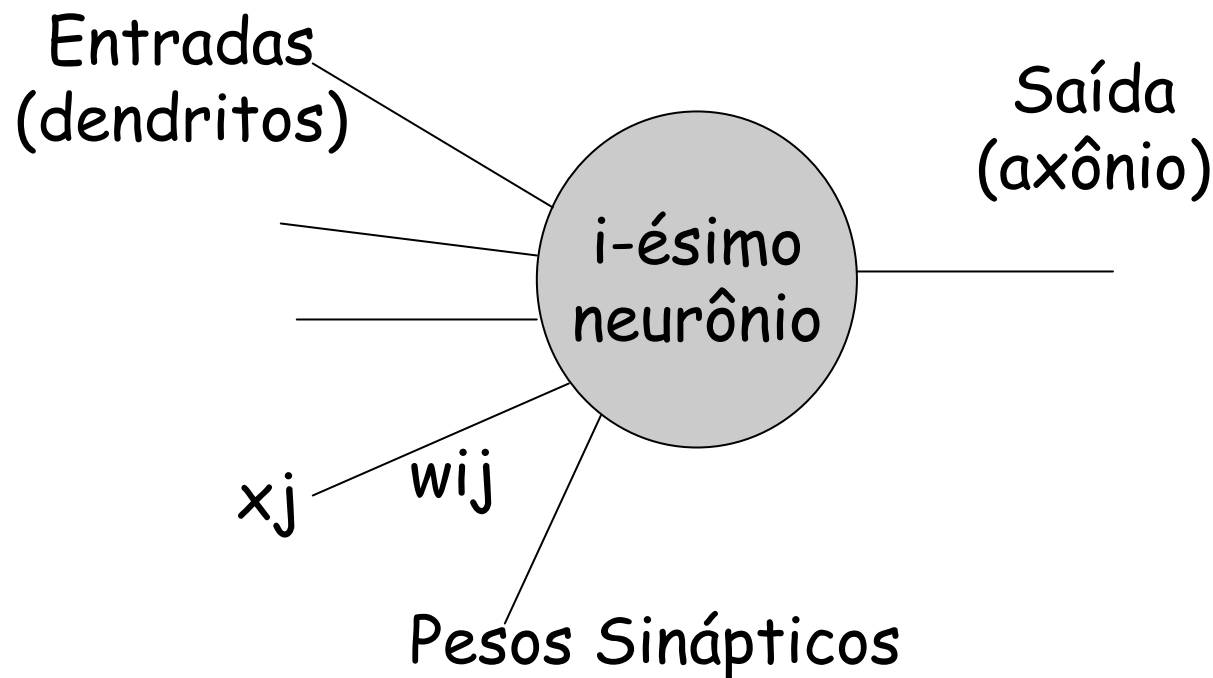
Uma forma de aprendizado no Modelo de McCulloch & Pitts

- ♦ O Aprendizado de Hebb (1949)
 - Base para todas as outras regras de aprendizado
 - "Quando um axônio de um neurônio A está próximo o suficiente para excitar um neurônio B, e repetidamente ou persistentemente toma parte do disparo de B, então, ocorre um certo processo de crescimento ou mudança metabólica em uma das 2 células, de forma que a eficiência de A em contribuir para o disparo de B é aumentado ("força do contato sináptico)."
 - Experimento de Pavlov



Modelos de Neurônios Artificiais

- ♦ Neurônio Artificial, Nó ou Processing Element-PE
- ♦ Um Primeiro Modelo



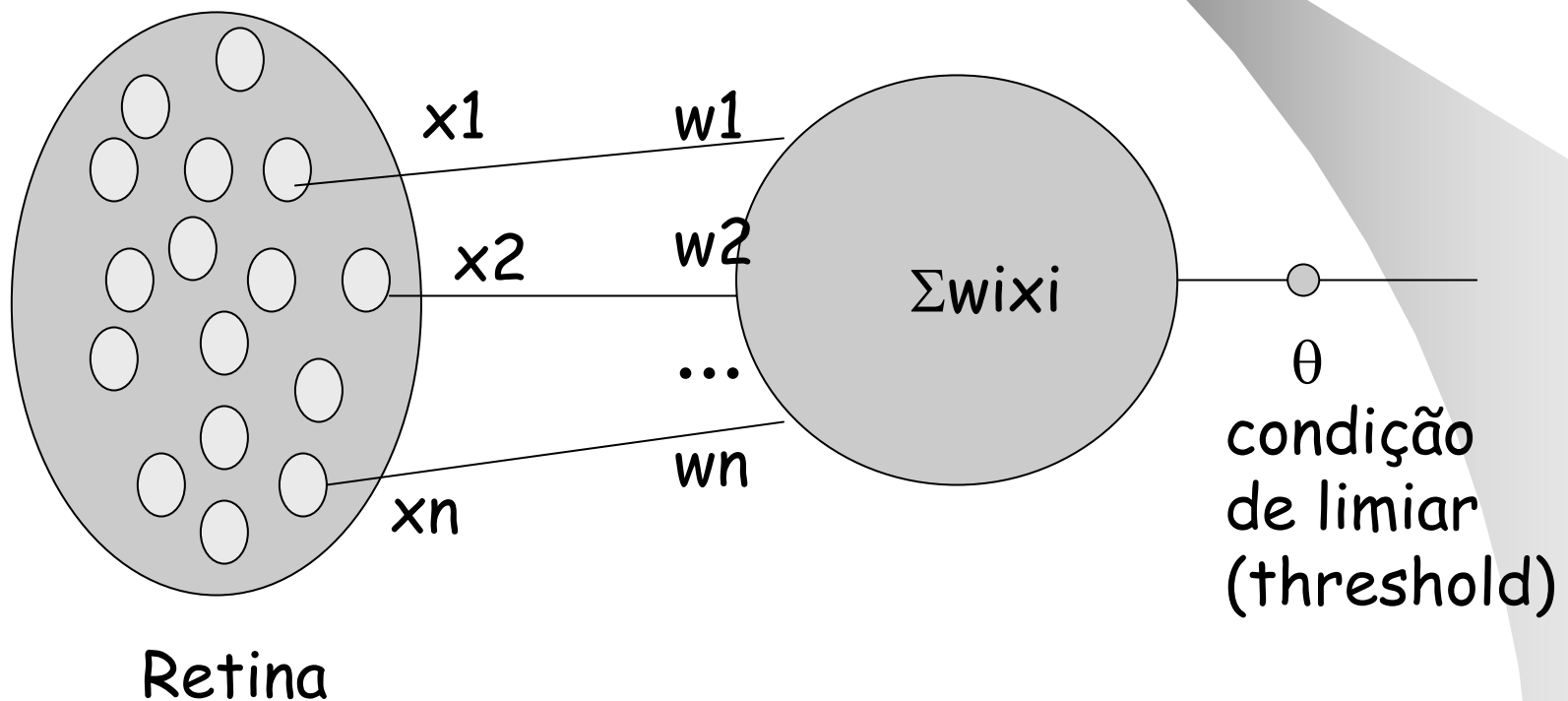
$$net_i = \sum_j w_{ij} \cdot x_j$$

Modelos de Neurônios Artificiais

- ♦ O PERCEPTRON : Frank Rosenblatt (1958)
 - "A conectividade desenvolvida nas redes biológicas contém um grande número aleatório de elementos".
 - No início, o Perceptron não é capaz de distinguir padrões e portanto ele é genérico.
 - Pode ser treinado.
 - Com o tempo foi-se notando que a capacidade de separabilidade era dependente de "certas condições de contorno" dos padrões de entrada.

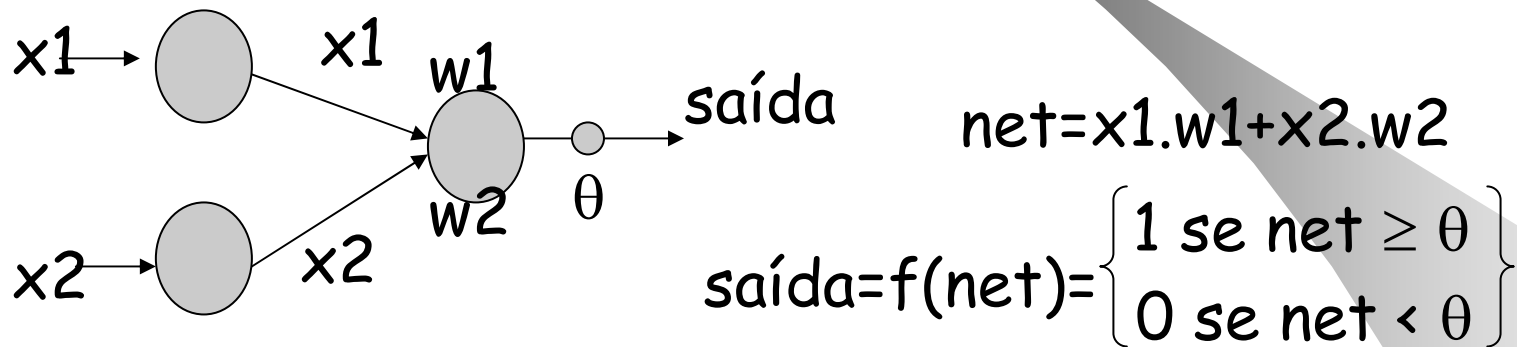
Modelos de Neurônios Artificiais

- ♦ O PERCEPTRON : Frank Rosenblatt (1958)



Modelos de Neurônios Artificiais

- ♦ O PERCEPTRON : Frank Rosenblatt (1958)

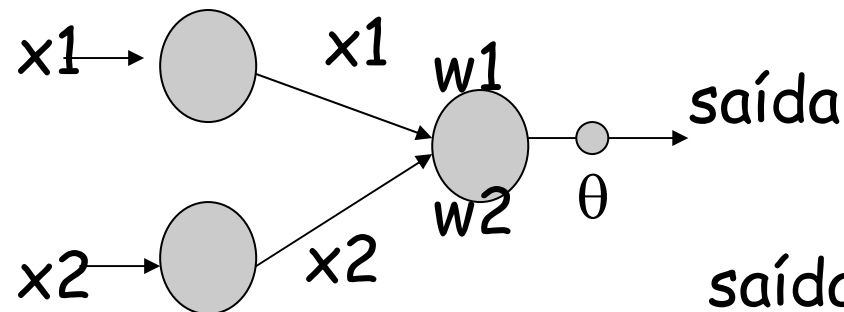


EQUAÇÃO FUNDAMENTAL DO PERCEPTRON

$w1.x1 + w2.x2 = \theta$ ← EQUAÇÃO DE UMA RETA

Modelos de Neurônio Artificiais

• O PERCEPTRON : Exemplos

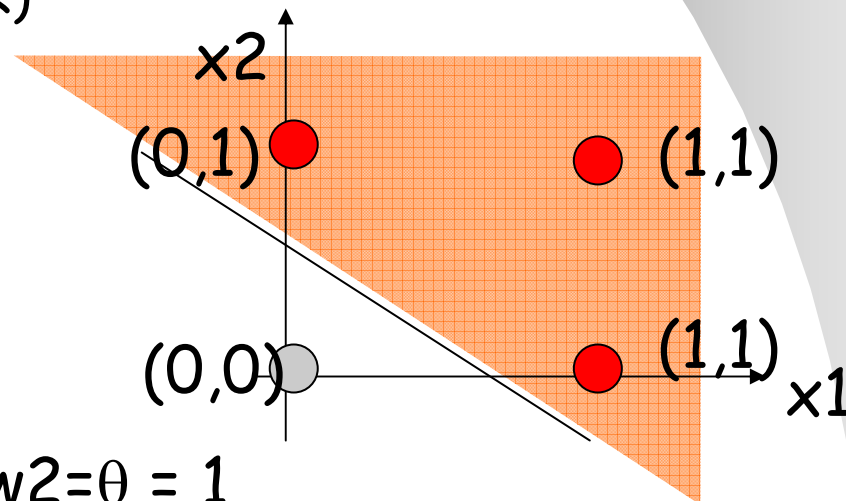


$$\text{net} = x1.w1 + x2.w2$$

$$\text{saída} = f(\text{net}) = \begin{cases} 1 & \text{se } \text{net} \geq \theta \\ 0 & \text{se } \text{net} < \theta \end{cases}$$

- Determinar $w1$, $w2$ e θ para que o Perceptron aprenda a função OU Lógico (OR)

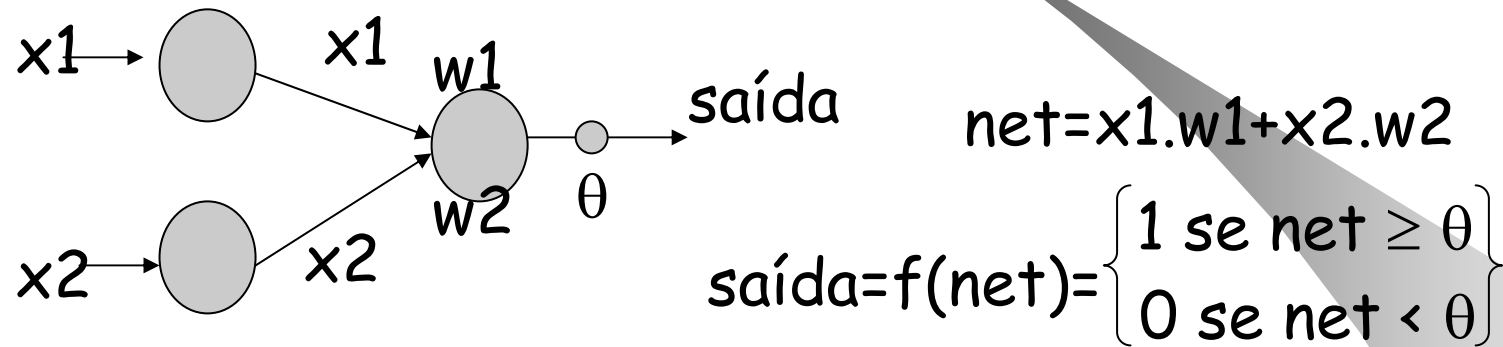
$x1$	$x2$	OR
0	0	0
0	1	1
1	0	1
1	1	1



Possível solução: $w1=w2=\theta = 1$

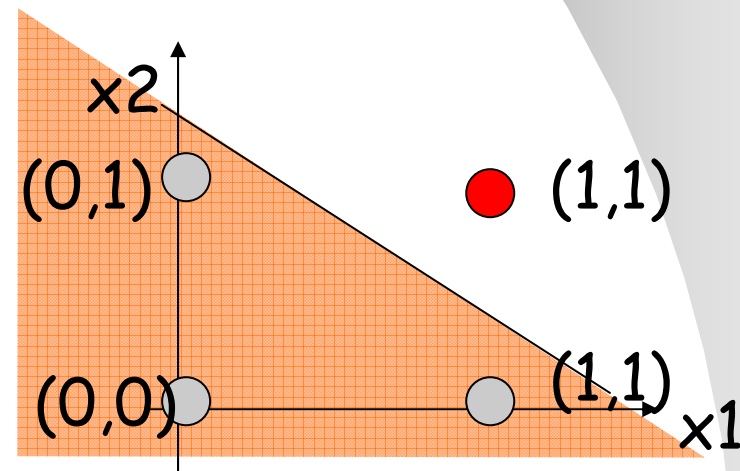
Modelos de Neurônios Artificiais

♦ O PERCEPTRON : Exemplos



- Determinar $w1$, $w2$ e θ para que o Perceptron aprenda a função E Lógico (AND)

x1	x2	OR
0	0	0
0	1	0
1	0	0
1	1	1



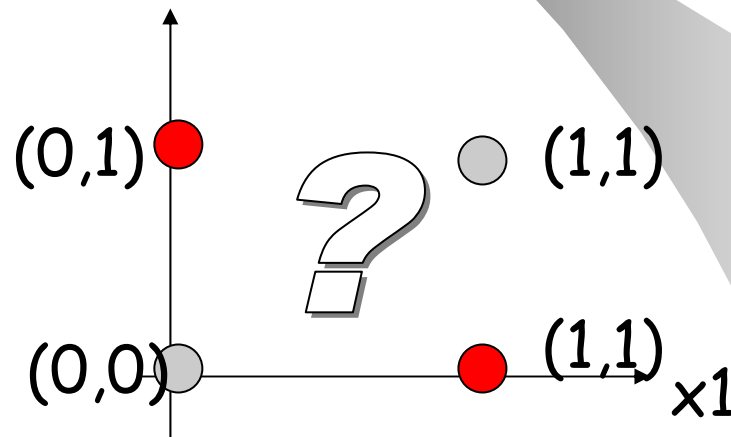
Possível solução: $w1=w2=1$ $\theta = 2$

Modelos de Neurônios Artificiais

♦ O PERCEPTRON : Exemplos

- Determinar w_1 , w_2 e θ para que o Perceptron aprenda a função OU-EXCLUSIVO Lógico (XOR)

x_1	x_2	OR
0	0	0
0	1	1
1	0	1
1	1	0



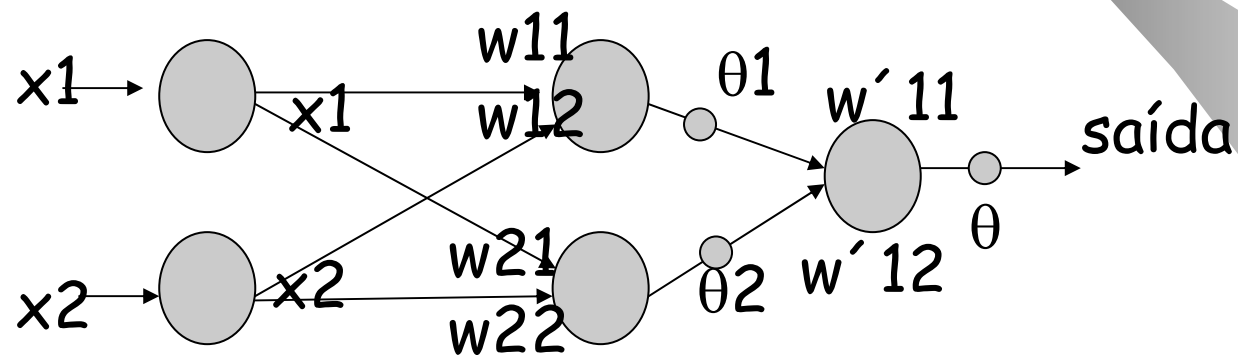
CONCLUSÃO: O PERCEPTRON É CAPAZ DE DISTINGUIR APENAS PADRÕES LINEARMENTE SEPARÁVEIS!!!

Modelos de Neurônios Artificiais

- ♦ O PERCEPTRON : Minsky e Papert (1969)
 - CONCLUSÃO: O PERCEPTRON É CAPAZ DE DISTINGUIR APENAS PADRÕES LINEARMENTE SEPARÁVEIS!!!
 - Isto causou um "trauma" na comunidade científica e levou ao corte de verbas para as pesquisas em Redes Neurais Artificiais.
 - "Se colocarmos mais uma camada de neurônios podemos resolver esta limitação. Mas como achar os pesos?"
 - Na época não se sabia como.

Modelos de Neurônios Artificiais

- ♦ O PERCEPTRON : Minsky e Papert (1969)



$$w_{11}=w_{12}=w_{21}=w_{22}=1$$

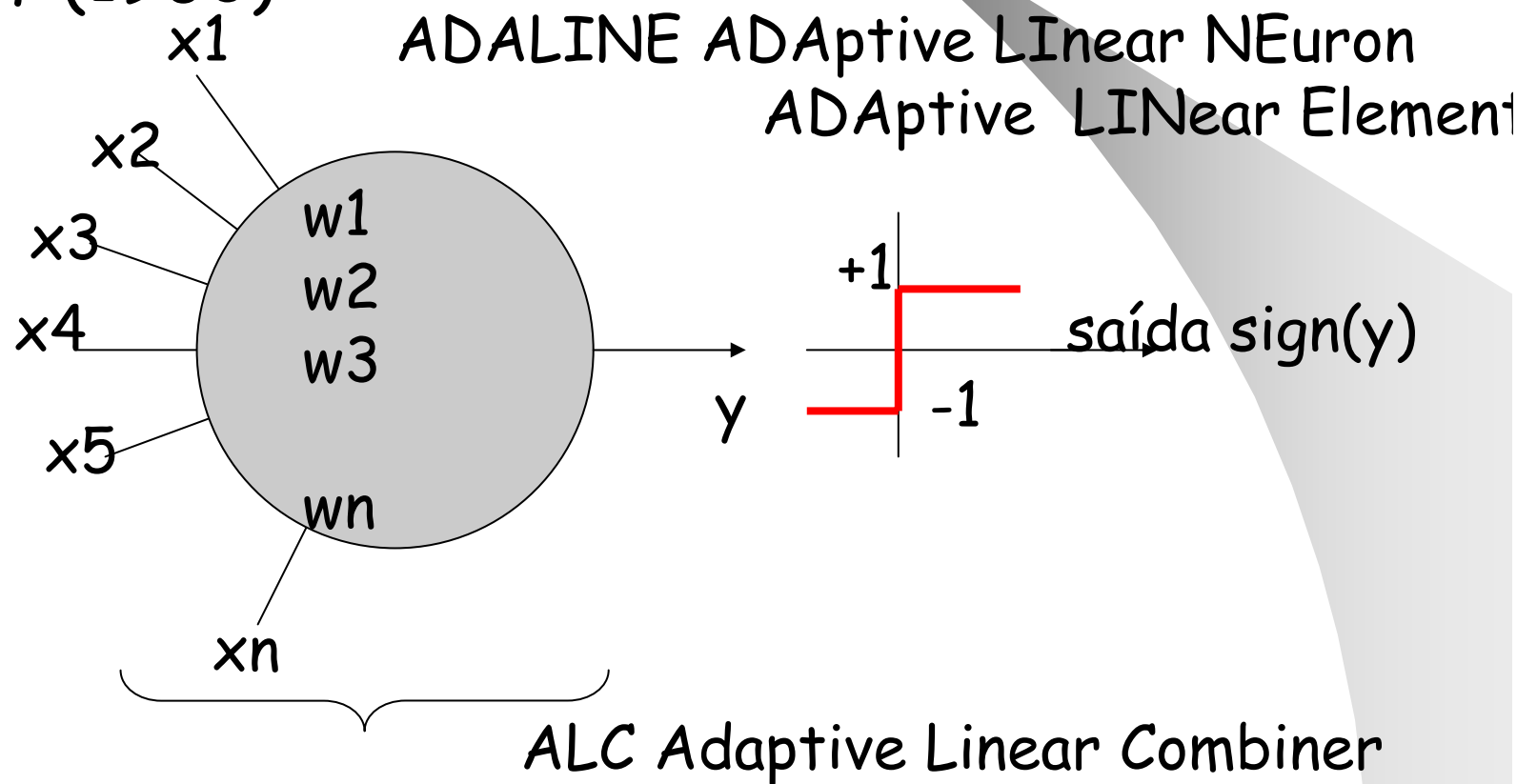
$$\theta_1=0,4$$

$$\theta_2=1,2$$

$$\theta = 0.5$$

Modelos de Neurônios Artificiais

- O ALGORITMO DE APRENDIZADO DO PERCEPTRON : O ADALINE - Widrow e Hoff (1960)



Modelos de Neurônios Artificiais

- ♦ O ADALINE - Widrow e Hoff (1960)

Formulação Vetorial

$$\bullet \quad W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} \quad X = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \quad y = W^T \cdot X$$

Suponha que temos um conjunto de vetores de entrada $\{X_1, X_2, \dots, X_L\}$, cada um com seu valor de saída correto (desejado) $\{d_1, d_2, \dots, d_L\}$. O Método de Aprendizado procura achar os pesos de forma a minimizar a diferença entre a saída desejada e a saída obtida com o vetor de entrada.

Modelos de Neurônios Artificiais

- ♦ Como Minimizar o Erro Médio Quadrático

- Exemplo

- Função OR: $k=4$

x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

$\underbrace{\quad\quad\quad}_{X_k} \quad\quad\quad \uparrow \quad d_k$

$$X_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad d_1 = 0$$

$$X_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad d_2 = 1$$

$$X_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad d_3 = 1$$

$$X_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad d_4 = 1$$

Modelos de Neurônios Artificiais

- ♦ Como Minimizar o Erro Médio Quadrático
 - Erro Médio Quadrático
 - Erro para o k-ésimo vetor de entrada
 $E_k = d_k - y_k$ desejado - obtido
 - Erro Médio Quadrático

$$\langle E_k^2 \rangle = \frac{1}{L} \sum_{k=1}^L E_k^2$$

$$\text{então } \langle E_k^2 \rangle = \langle (d_k - y_k)^2 \rangle$$

$$\text{mas } y_k = W^T X_k$$

$$\text{logo } \langle E_k^2 \rangle = \langle (d_k - W^T X_k)^2 \rangle =$$

$$\langle d_k^2 \rangle + W^T \langle X_k X_k^T \rangle W - 2 \langle d_k X_k^T \rangle W$$

Modelos de Neurônios Artificiais

- Como Minimizar o Erro Médio Quadrático
- Erro Médio Quadrático

$$\langle E_k^2 \rangle = \langle d_k^2 \rangle + W^T \langle X_k X_k^T \rangle W - 2 \langle d_k X_k^T \rangle W$$

Equação de uma Parábola

Minimização do Error Médio Quadrático significa encontrar o FUNDO DA PARÁBOLA.

$$\frac{\partial \langle E_k^2 \rangle}{\partial W} = 0$$

$$2 \langle X_k X_k^T \rangle W - 2 \langle d_k X_k \rangle = 0$$

Modelos de Neurônios Artificiais

- ♦ Exercício: Calculemos os valores de W para o problema do OU.

Modelos de Neurônios Artificiais

- ♦ O ADALINE - Widrow e Hoff (1960)
- ♦ A Regra de Hebb
 - Idealizada por Hebb, a idéia básica é que se duas unidades são ativadas simultaneamente, suas interconexões tendem a se fortalecer. Se i recebe o sinal de saída de j , o peso W_{ij} é modificado de acordo com :

$$\Delta W_{ij} = \lambda a_i a_j$$

- onde λ (lambda) é uma constante de proporcionalidade representando a taxa de aprendizado e a_i e a_j são ativações (ou saídas) das unidades i e j respectivamente. Alguns autores representam alternativamente a matriz W_{ij} por W_{ji} .

Modelos de Neurônios Artificiais

- ♦ O ADALINE - Widrow e Hoff (1960)
- ♦ A Regra Delta - Erro Médio Quadrático Mínimo (LMS) - Widrow-Hoff
 - É uma variante da Regra de Hebb, introduzida por Widrow-Hoff. A diferença quanto a de Hebb é que possui uma saída desejada d_j . Assim, o peso será proporcional à saída.
 - Sendo a_j e a_i os níveis de ativações das unidades j e i respectivamente, a variação dos pesos é:

$$\Delta W_{ij} = (d_j - a_j) a_i$$

Modelos de Neurônio Artificiais

- ♦ A Regra Delta - Erro Médio Quadrático Mínimo (LMS) - Widrow-Hoff
 - Determinação dos Conjunto de Pesos W , pelo Método da Descida Mais Íngreme (Steepest Descente)
 - Método Iterativo
 - Escreve-se W como uma função do "tempo".
 - Vetor de pesos iniciais $W(0)$
 - Vetor de pesos no "passo" ou "tempo" t $W(t)$

Modelos de Neurônios Artificiais

- ♦ Determinação dos Conjunto de Pesos W , pelo Método da Descida Mais Íngreme (Steepest Descent)

1-Começar especificando valores aleatórios para os pesos.

2-Aplicar um vetor de entrada X_k .

3-Determinar o erro $E_k(t)$ utilizando $W(t)$.

$$E_k^2(t) = (d_k - W^T(t)X_k)^2$$

4-Supor que $E_k^2(t)$ é uma aproximação razoável para o Erro Médio Quadrático $\langle E_k^2(t) \rangle$

Modelos de Neurônios Artificiais

- ♦ Determinação dos Conjunto de Pesos W , pelo Método da Descida Mais Íngreme (Steepest Descent)

5-Calcular o gradiente do erro, isto é a "direção" em que derivada é maior.

$$E_k^2(t) = (d_k - W^T(t)X_k)^2$$

$$\nabla E_k^2(t) = 2(d_k - W^T(t)X_k) \frac{\partial E_k}{\partial W} =$$

$$2E_k(t).(-X_k) = -2E_k X_k$$

Modelos de Neurônios Artificiais

- ♦ Determinação dos Conjunto de Pesos W , pelo Método da Descida Mais Íngreme (Steepest Descent)

6-Atualizar o Vetor de Pesos.

$$W(t+1) = W(t) - \mu \nabla E_k^2(t)$$

$$W(t+1) = W(t) + 2\mu E_k X_k$$

com

$$E_k = d_k - W^T(t) X_k$$

Modelos de Neurônios Artificiais

- ♦ Determinação dos Conjunto de Pesos W , pelo Método da Descida Mais Íngreme (Steepest Descent)
 - 7-Repetir os passos de 2 a 6 até o erro alcançar um valor suficientemente pequeno.

Modelos de Neurônios Artificiais

- ♦ Exercício: Calculemos os valores de W para o problema do OU.

O Período "Negro"

- ♦ Minsky e Papert - 1969 - Livro "Perceptrons". O Perceptron (ou o Adaline) é incapaz de classificar corretamente padrões não linearmente separáveis.
- ♦ A maioria dos problemas são não linearmente separáveis.
- ♦ Apesar do descrédito gerado sobre a área da neurocomputação, entre 1969 e 1982 os estudos neste campo continuaram, ainda que englobadas em outras linhas de pesquisa, como processamento adaptativo de sinais, reconhecimento de padrões, modelagem biológica, etc. Este trabalho, ainda que silencioso, construiu as bases necessárias para que o desenvolvimento das redes neurais pudesse continuar de forma consistente.

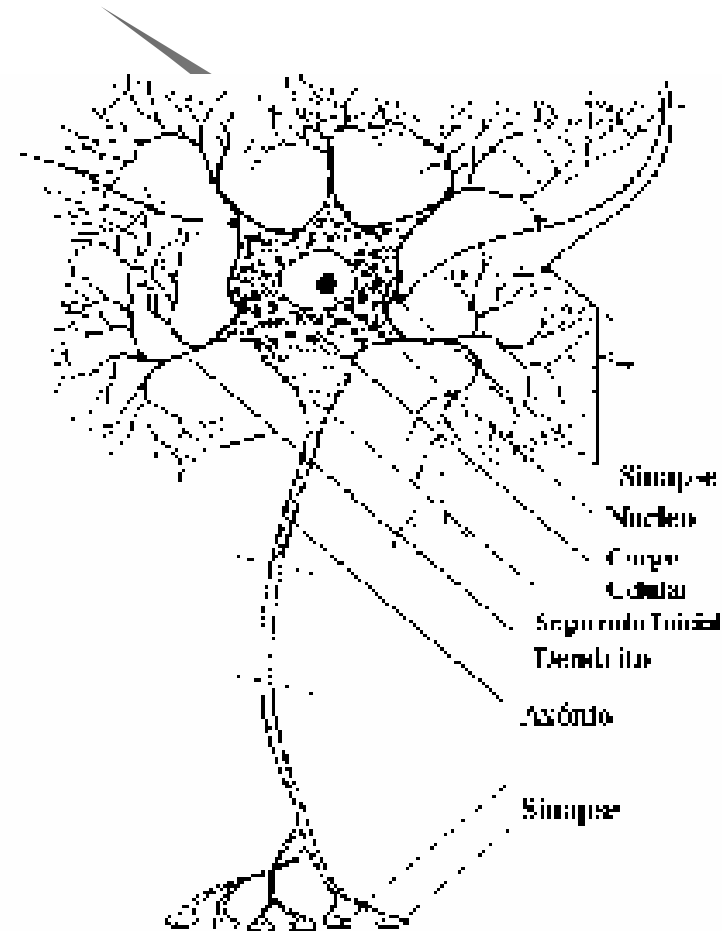
O Renascimento

- ♦ Em 1974, Paul Werbos conseguiu o maior progresso em termos de redes neurais desde o perceptron de Rosenblatt: ele lançou as bases do algoritmo de retro-propagação ("backpropagation"), que permitiu que redes neurais com múltiplas camadas apresentassem capacidade de aprendizado. Em 1982, David Parker desenvolveu um método similar, de forma aparentemente independente. Contudo, a potencialidade deste método tardou a ser reconhecida.
- ♦ Os primeiros resultados da retomada do desenvolvimento sobre redes neurais foram publicados em 1986 e 1987, através dos trabalhos do grupo PDP (Parallel and Distributed Processing), onde ficou consagrada a técnica de treinamento por backpropagation.

Redes Neurais Artificiais

Os Tempos Modernos

- ♦ Elementos Básicos de um Neurônio Artificial
 - Uma Abordagem Unificada.
 - Terminologia Básica.



Elementos Básicos de um Neurônio Artificial

- ♦ **Modelo de Neurônio Artificial**

- **As Entradas**

- As entradas de um neurônio podem ser as saídas de outros neurônios, entradas externas, um bias ou qualquer combinação destes elementos.

- **A Combinação das Entradas - O "Net"**

- O somatório de todas estas entradas, multiplicadas por suas respectivas forças de conexão sináptica (os pesos), dá origem ao chamado "net" de um neurônio.

$$net_i(t) = \sum_{j=1}^n w_{ij} u_j(t)$$

w_{ij} é um número real que representa a conexão sináptica da entrada do $i^{ésimo}$ neurônio com a saída do $j^{ésimo}$ neurônio.

A conexão sináptica é conhecida como excitatória se $w_{ij} > 0$ ou inibitória caso $w_{ij} < 0$

Elementos Básicos de um Neurônio Artificial

- ♦ **Modelo de Neurônio Artificial**

- Após a determinação do net_i , o valor de saída do neurônio é produzido através da função de saída λ .
- A Função de Saída λ
 - Essencialmente, qualquer função contínua e monotonicamente crescente tal que $x \in \mathbb{R}$ e $y(x) \in [-1,1]$ pode ser utilizada como função de saída na modelagem neural.
 - Existem, no entanto, uma série de funções mais comumente utilizadas como funções de saída em neurônios. Estas funções são:
 - A Função Linear
 - A Função Sigmoidal ou Logística
 - A Função Tangente Hiperbólica

Elementos Básicos de um Neurônio Artificial

- ♦ **Modelo de Neurônio Artificial**

- A Função de Saída λ

- A Função Linear

$$y(x) = ax$$

- A Função Sigmoidal ou Logística - Função UNIPOLAR mais utilizada.

$$y(x) = \frac{1}{1 + e^{-kx}}$$

onde k é um fator de escala positivo.

Elementos Básicos de um Neurônio Artificial

- ♦ **Modelo de Neurônio Artificial**

- A Função de Saída λ

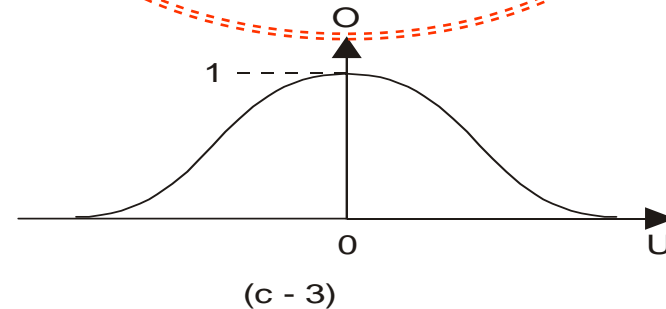
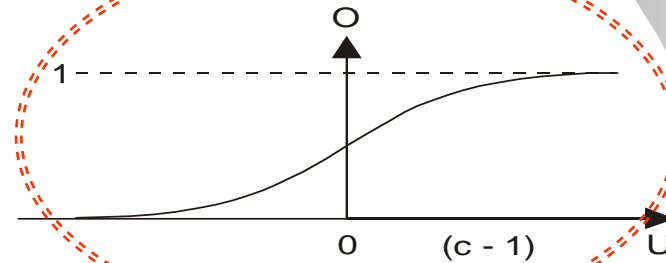
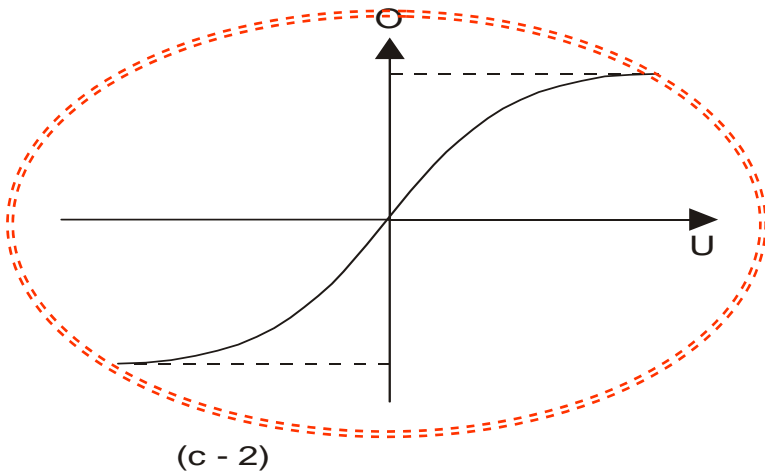
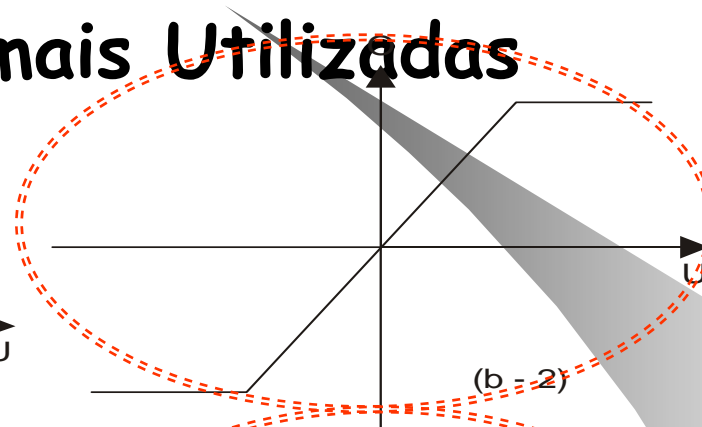
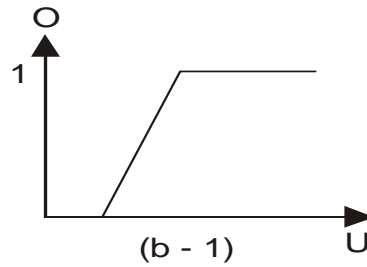
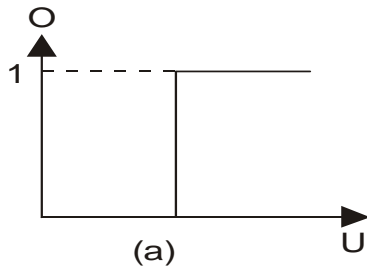
- A Função Tangente Hiperbólica - A Função BIPOLAR mais utilizada.

$$y(x) = \tanh(kx) = \frac{1 - e^{-kx}}{1 + e^{-kx}}$$

onde k é um fator de escala positivo.

Elementos Básicos de um Neurônio Artificial

♦ Funções de Saída mais Utilizadas



Elementos Básicos de um Neurônio Artificial

- Confusão na Nomenclatura - CUIDADO!!!

- Na literatura muitas vezes só é apresentado o neurônio estático e portanto muitas vezes se confunde a função de ativação com a função de saída. Também é comum encontrarmos o termo "função de transferência".

Topologias Das Redes Neurais

Como os Neurônio se Conectam

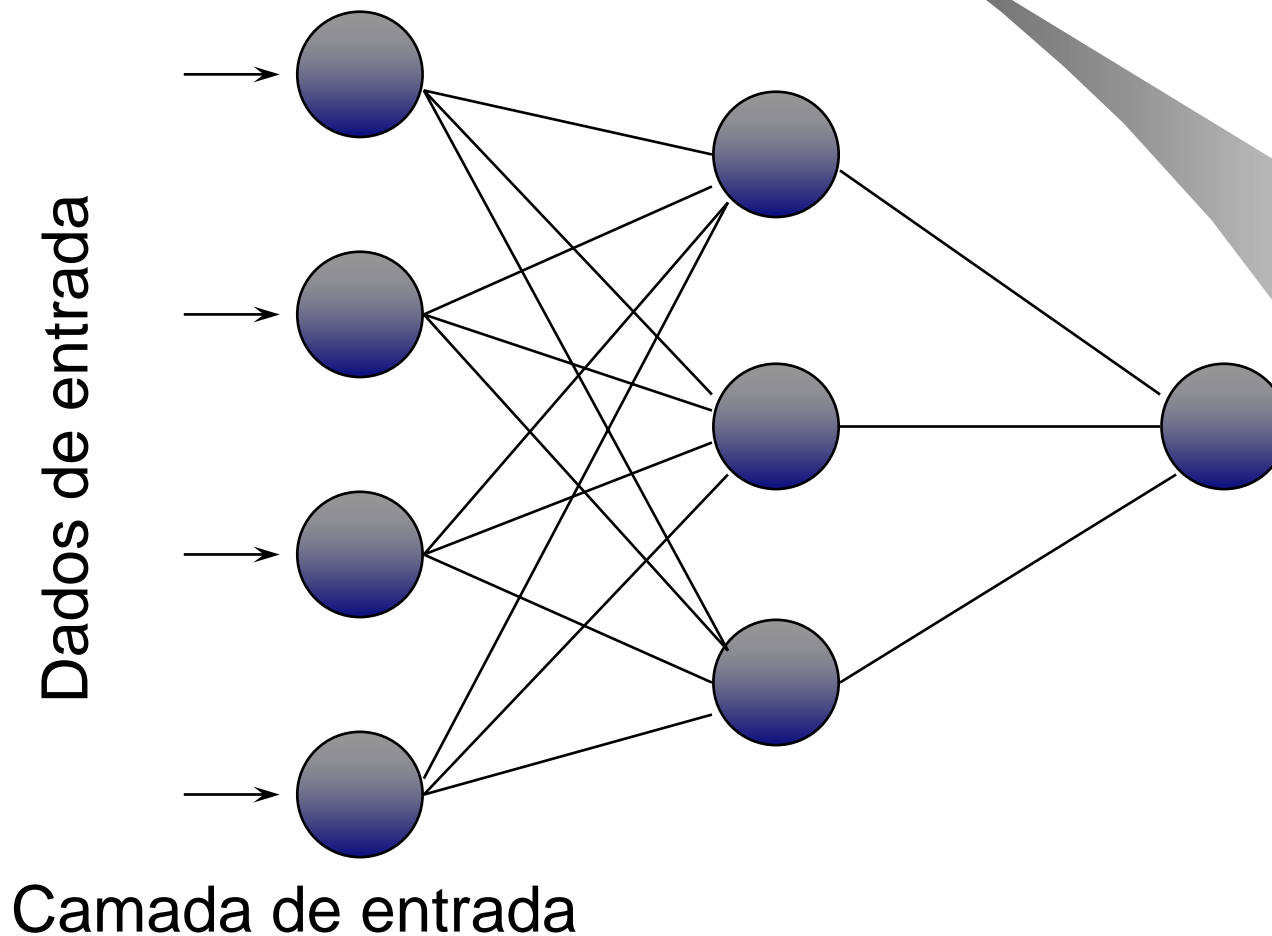
- ♦ Depende da forma como os Neurônios se conectam para formar uma "Rede" de neurônios.
- ♦ Redes Diretas - "Feedforward" -
 - Não existem ciclos.
 - Conexões para a "frente".
- ♦ Redes Recorrentes - "Feedback"
 - Existem ciclos.
 - Conexões para trás e/ou para os lados.

Topologia das Redes Neurais

- ♦ Redes Neurais Diretas - Feedforward
 - As redes diretas são aquelas cujo grafo não tem ciclos.
 - Frequentemente é comum representar estas redes em camadas e neste caso são chamadas redes de camadas.

Topologia das Redes Neurais

- ♦ Redes Neurais Diretas - Feedforward



Topologia das Redes Neurais

- ♦ Redes Neurais Diretas

- Neurônios que recebem sinais de excitação do ambiente são chamados de camada de entrada ou primeira camada.
- Neurônios que tem sua saída como saída da rede pertencem a camada de saída ou última camada.
- Neurônios que recebem sinais apenas de outros neurônios e enviam suas saídas para outros neurônios, pertencem a camadas intermediárias ou escondidas (hidden).

Aprendizado de Redes Neurais

- ♦ Um neurônio é considerado ser um elemento adaptativo. Seus pesos sinápticos são modificáveis dependendo do algoritmo de aprendizado.
- ♦ Por exemplo, alguns, dependendo do sinal de entrada que recebem, tem seus valores de saída associados a uma resposta diante de um aprendizado supervisionado por uma espécie de "professor".
- ♦ Em alguns casos o sinal do "professor" não está disponível e não há informação de erro que possa ser utilizada para correção dos pesos sinápticos, assim o neurônio modificará seus pesos baseado somente no sinal de entrada e/ou saída, sendo o caso do aprendizado não-supervisionado.

Aprendizado de Redes Neurais

♦ Aprendizado Supervisionado

- Neste caso, o "professor" indica explicitamente um comportamento bom ou ruim.
- Por exemplo, seja o caso de reconhecimento de caracteres e para simplificar seja reconhecer entre um A e um X.
- Escolhe-se uma rede direta, com dois neurônios na camada de saída, uma ou várias camadas internas e um conjunto de neurônios na camada de entrada capaz de representar com a precisão desejada a letra em questão.
- Apresentam-se estas letras sucessivamente a uma retina artificial constituída de uma matriz de elementos foto-sensíveis, cada um ligado a um neurônio da rede neural artificial direta.

Aprendizado de Redes Neurais

- ♦ **Aprendizado Supervisionado**
 - Observa-se qual dos dois neurônios de saída está mais excitado. Se for o que se convencionou representar a letra que for apresentada nada deve ser corrigido, caso contrário modifica-se os valores das conexões sinápticas no sentido de fazer a saída se aproximar da desejada.
 - Foi exatamente isto que Rosenblatt fez com o seu Perceptron. Como a cada exemplo apresentado uma correção é introduzida depois de observar a saída da rede, este é um caso de aprendizado supervisionado.

Aprendizado de Redes Neurais

- ♦ Aprendizado Não-Supervisionado
 - É aquele que para fazer modificações nos valores das conexões sinápticas não usa as informações sobre a resposta da rede, isto é se a resposta está correta ou não.
 - Usa-se por outro lado um esquema, tal que, para exemplos de coisas semelhantes, a rede responda de modo semelhante.

Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation
 - Estrutura da Rede:
 - Rede Direta Multi-camada com Neurônios Estáticos.
 - Modo de Treinamento:
 - Supervisionado.
 - Solução para superar o problema do aprendizado da classificação de padrões não-linearmente separáveis:
 - Utilização de uma camada intermediária de neurônios, chamada Camada Intermediária (ou Escondida - "Hidden Layer"), de modo a poder implementar superfícies de decisão mais complexas.

Aprendizado de Redes Neurais

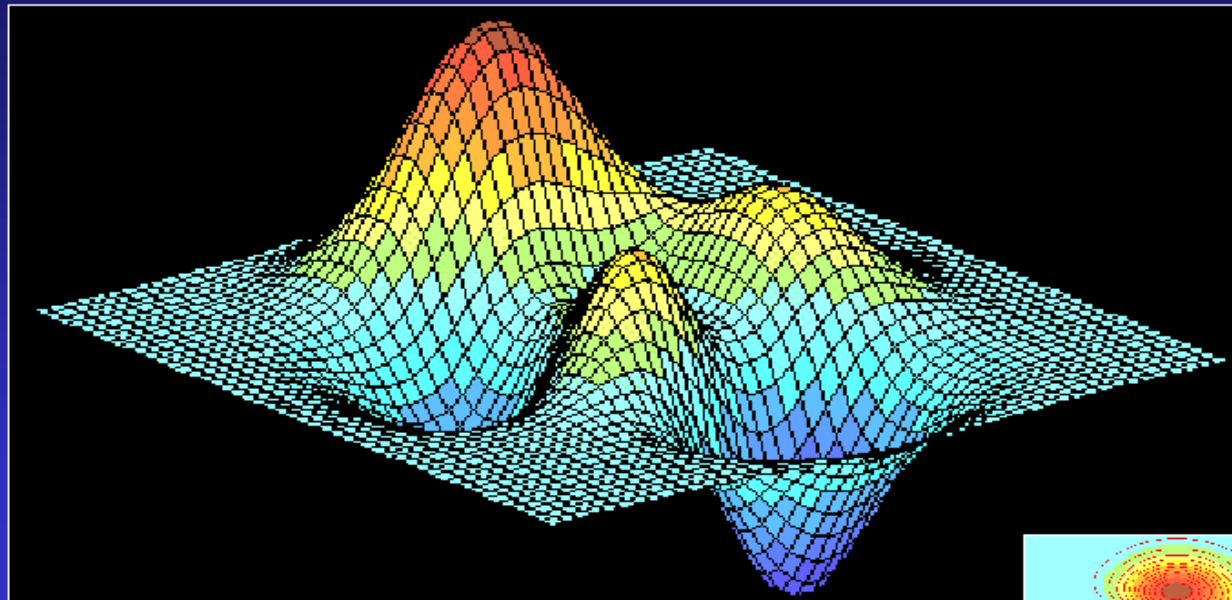
- ♦ O Aprendizado Backpropagation
 - Desvantagem em utilizar esta camada escondida:
 - O aprendizado se torna muito mais difícil.

A característica principal da camada escondida é que seus elementos se organizam de tal forma que cada elemento aprenda a reconhecer características diferentes do espaço de entrada, assim, o algoritmo de treinamento deve decidir que características devem ser extraídas do conjunto de treinamento.
 - Nos anos 80, um algoritmo chamado Retro-propagação ou Backpropagation, veio fazer renascer o interesse geral pelas redes neurais.

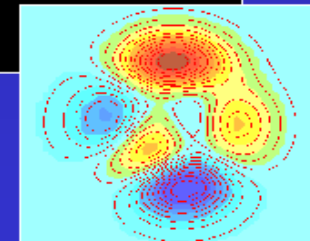
Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation

Um exemplo de Superfície Adaptativa



Vista superior →

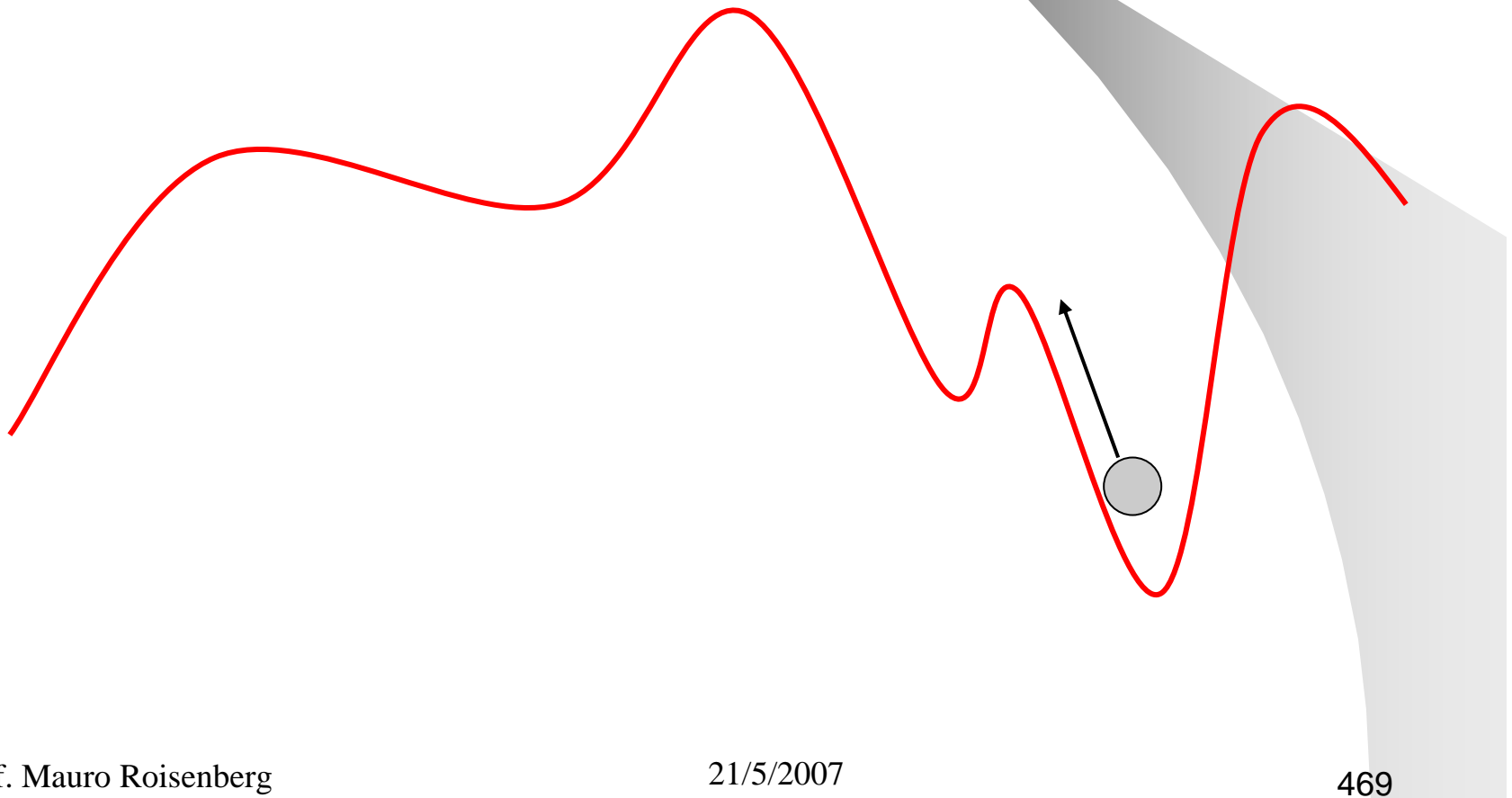


Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation
 - Utiliza o mesmo princípio da Regra Delta
 - a minimização de uma função custo, no caso, a soma dos erros médios quadráticos sobre um conjunto de treinamento, utilizando a técnica de busca do gradiente-descendente.
 - Também é chamado muitas vezes de Regra Delta Generalizada ("Generalized Delta-Rule").
 - A modificação principal em relação a Regra Delta foi a utilização de funções contínuas e suaves como função de saída dos neurônios ao invés da função de limiar lógico.
 - Como as funções de saída passaram a ser deriváveis, isto permitiu a utilização da busca do gradiente-descendente também para os elementos das camadas intermediárias.

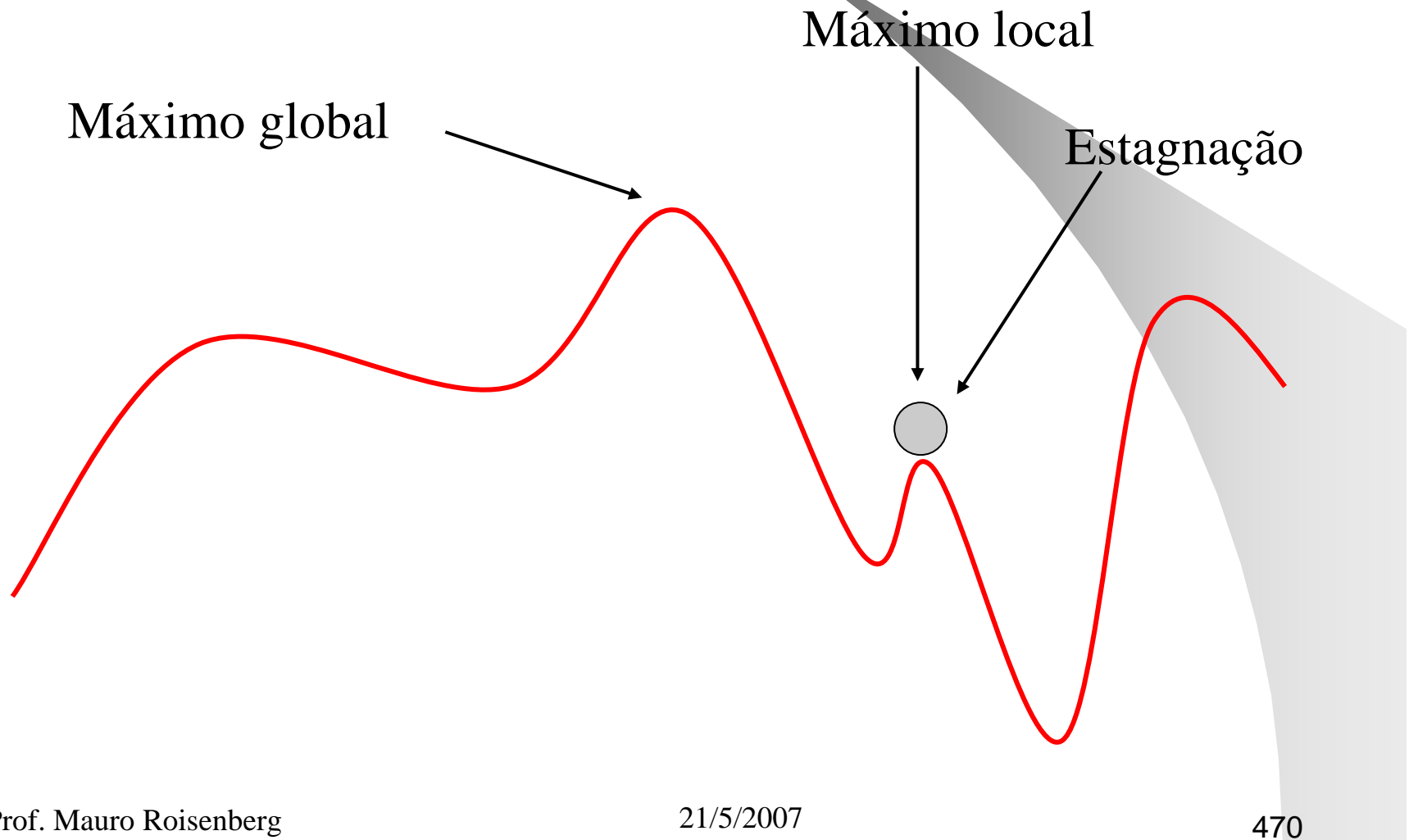
Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation



Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation



Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation
 - O algoritmo Backpropagation é hoje em dia a técnica de aprendizado supervisionado mais utilizada para redes neurais unidirecionais multi-camadas com neurônios estáticos.
 - Basicamente, a rede aprende um conjunto pré-definido de pares de exemplos de entrada/saída em ciclos de propagação/adaptação.
 - Depois que um padrão de entrada foi aplicado como um estímulo aos elementos da primeira camada da rede, ele é propagado por cada uma das outras camadas até que a saída seja gerada. Este padrão de saída é então comparado com a saída desejada e um sinal de erro é calculado para cada elemento de saída.

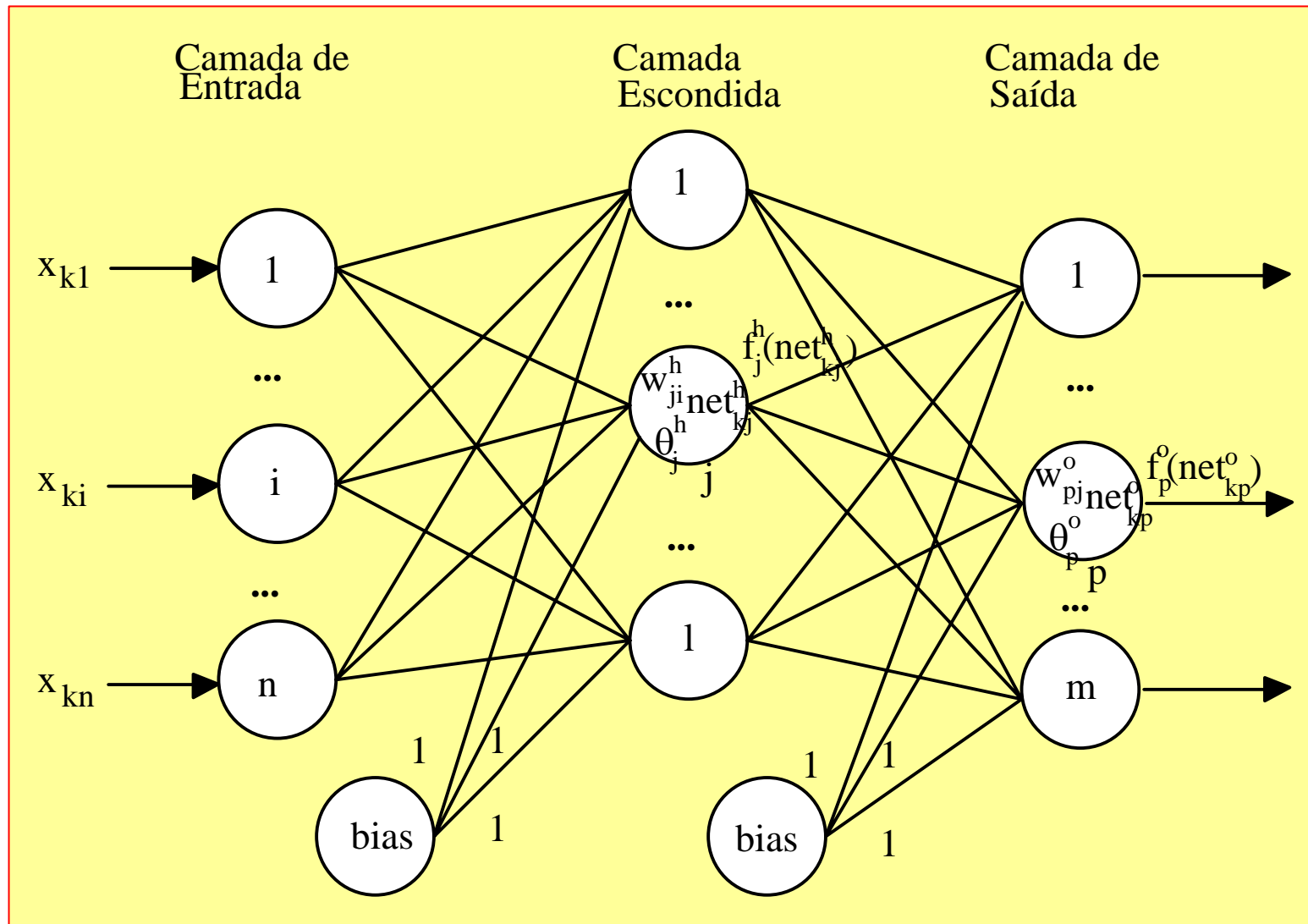
Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation
 - O sinal de erro é então retro-propagado da camada de saída para cada elemento da camada intermediária anterior que contribui diretamente para a formação da saída.
 - Cada elemento da camada intermediária recebe apenas uma porção do sinal de erro total, proporcional apenas à contribuição relativa de cada elemento na formação da saída original.

Aprendizado de Redes Neurais

- ♦ O Aprendizado Backpropagation
 - Este processo se repete, camada por camada, até que cada elemento da rede receba um sinal de erro que descreva sua contribuição relativa para o erro total.
 - Baseado no sinal de erro recebido, os pesos das conexões são então atualizados para cada elemento de modo a fazer a rede convergir para um estado que permita a codificação de todos os padrões do conjunto de treinamento.

Algoritmo Backpropagation



Algoritmo Backpropagation

♦ Princípios Básicos

- Suponhamos que tenhamos um conjunto de P pares de vetores $(\mathbf{X}_1, \mathbf{Y}_1), (\mathbf{X}_2, \mathbf{Y}_2), \dots, (\mathbf{X}_P, \mathbf{Y}_P)$, no nosso conjunto de treinamento e que são exemplos de um mapeamento funcional definido como:

$$Y = \theta(X) : X \in \mathcal{R}^n, Y \in \mathcal{R}^m$$

- Desejamos treinar a rede de modo que ela consiga aprender uma aproximação da forma:

$$O = Y' = \theta'(X) : X \in \mathcal{R}^n, Y' \in \mathcal{R}^m$$

Algoritmo Backpropagation

♦ Etapas

- 1. Aplicar um vetor de entrada do conjunto de treinamento e propagar até a saída

- Um vetor de entrada $\mathbf{X}_k = [x_{k1} \ x_{k2} \ \dots \ x_{kn}]^T$ do conjunto de treinamento é apresentado à camada de entrada da rede. Os elementos de entrada distribuem os valores para os elementos da camada escondida. O valor do net para o $j^{\text{ésimo}}$ elemento da camada escondida vale:

$$net_{kj}^h = \sum_{i=1}^n w_{ji}^h x_{ki} + \theta_j^h$$

- onde w_{ji} é o peso da conexão entre o $i^{\text{ésimo}}$ elemento da camada de entrada e o $j^{\text{ésimo}}$ elemento da camada escondida h .

Algoritmo Backpropagation

♦ Etapas

- 1. Aplicar um vetor de entrada do conjunto de treinamento e propagar até a saída
 - Como os neurônios são estáticos, o valor de saída para um neurônio da cada escondida vale:

$$i_{kj} = f_j^h(net_{kj}^h)$$

Algoritmo Backpropagation

♦ Etapas

- 1. Aplicar um vetor de entrada do conjunto de treinamento e propagar até a saída

- Do mesmo modo, as equações para os neurônios da camada de saída são:

$$net_{kp}^o = \sum_{j=1}^l w_{pj}^o i_{kj} + \theta_p^o$$
$$o_{kp} = f_p^o(net_{kp}^o)$$

Algoritmo Backpropagation

♦ Etapas

- 2. Calcular o erro entre a saída calculada pela rede e a saída desejada no conjunto de treinamento.

- Definimos o erro para um único neurônio p na camada de saída para um vetor de entrada k como sendo:

$$E_{kp} = (y_{kp} - o_{kp})$$

- E o erro a ser minimizado pelo algoritmo para todos os neurônios da camada de saída como:

$$E_k = \frac{1}{2} \sum_{p=1}^m E_{kp}^2$$

$$E_k = \frac{1}{2} \sum_{p=1}^m (y_{kp} - o_{kp})^2$$

Algoritmo Backpropagation

♦ Etapas

- 3. Determinar em que direção a mudança de peso deverá ocorrer.
 - Para determinar a direção da modificação dos pesos, calculamos o negativo do gradiente de E_k , ∇E_k , com relação aos pesos w_{pj}

$$\frac{\partial E_k}{\partial w_{pj}^o} = -(y_{kp} - o_{kp}) \frac{\partial f_p^o}{\partial (net_{kp}^o)} \frac{\partial (net_{kp}^o)}{\partial w_{pj}^o}$$

- Podemos escrever a derivada de f_p^o como:
 $f_p^{o'}(net_{kp}^o)$

- e o último termo da equação como:
$$\frac{\partial (net_{kp}^o)}{\partial w_{pj}^o} = \left(\frac{\partial}{\partial w_{pj}^o} \sum_{j=1}^l w_{pj}^o i_{kj} + \theta_p^o \right) = i_{kj}$$

Algoritmo Backpropagation

♦ Etapas

- 3. Determinar em que direção a mudança de peso deverá ocorrer.

- Combinando as equações, temos para o negativo do gradiente:

$$-\frac{\partial E_k}{\partial w_{pj}^o} = (y_{kp} - o_{kp}) f_p^{o'}(net_{kp}^o) i_{kj}$$

- 4. Determinar o valor da mudança de cada peso.

- A atualização dos pesos dos neurônios da camada de saída se faz por:

$$w_{pj}^o(t+1) = w_{pj}^o(t) + \Delta_k w_{pj}^o(t)$$

$$\Delta_k w_{pj}^o = \eta (y_{kp} - o_{kp}) f_p^{o'}(net_{kp}^o) i_{kj}$$

- onde η é a TAXA DE APRENDIZADO.

Algoritmo Backpropagation

- ♦ Etapas

- 4. Determinar o valor da mudança de cada peso.
 - AS FUNÇÕES DE SAÍDA -para que possamos implementar a busca do gradiente-descendente, é necessário que f_p^o seja diferenciável.
 - as funções usualmente utilizadas são:
 - a função linear:
 - a função logística ou sigmoidal:

$$f_p^o(net_{jp}^o) = net_{jp}^o$$

- a função tangente hiperbólica:

$$f_p^o(net_{jp}^o) = \frac{1}{1 + e^{-net_{jp}^o}}$$

$$f_p^o(net_{jp}^o) = \tanh(net_{jp}^o) = \frac{1 - e^{-net_{jp}^o}}{1 + e^{-net_{jp}^o}}$$

Algoritmo Backpropagation

- ♦ Etapas

- 4. Determinar o valor da mudança de cada peso.

- AS FUNÇÕES DE SAÍDA

- Estas funções são bastante populares pois as suas derivadas podem ser calculadas de maneira simples, sem a necessidade de cálculos complexos.

- para a função linear:

$$f_p^{o'} = 1$$

- para a função logística ou sigmoidal:

$$f_p^{o'} = f_p^o * (1 - f_p^o)$$

- para a função tangente hiperbólica:

$$f_p^{o'} = \frac{1}{2} * (1 - f_p^{o2})$$

Algoritmo Backpropagation

♦ Etapas

- 5. Repetir os procedimentos para os pesos da Camada Intermediária.
 - Desejamos repetir para a camada escondida os mesmos tipos de cálculos que realizamos para a camada de saída.
 - O problema aparece quando tentamos determinar uma medida para o erro das saídas dos neurônios da camada escondida.
 - Sabemos qual é a saída destes neurônios calculada pela rede, porém não sabemos a priori qual deveria ser a saída correta para estes elementos. Intuitivamente, o erro total, E_k , deve de alguma forma estar relacionado com o valor de saída dos neurônios da camada escondida.

Algoritmo Backpropagation

♦ Etapas

- 5. Repetir os procedimentos para os pesos da Camada Intermediária.
 - Voltando a equação do Erro temos:

$$\begin{aligned} E_k &= \frac{1}{2} \sum_p (y_{kp} - o_{kp})^2 \\ &= \frac{1}{2} \sum_p (y_{kp} - f_p^o(\text{net}_{kp}^o))^2 \\ &= \frac{1}{2} \sum_p (y_{kp} - f_p^o(\sum_j w_{pj}^o i_{kj} + \theta_p^o))^2 \end{aligned}$$

Algoritmo Backpropagation

♦ Etapas

- 5. Repetir os procedimentos para os pesos da Camada Intermediária.
 - Sabendo que i_{pj} depende dos pesos da camada escondida, podemos utilizar este fato para calcular o gradiente de E_k com respeito aos pesos da camada escondida.

$$\begin{aligned}\frac{\partial E_k}{\partial w_{ji}^h} &= \frac{1}{2} \sum_p \frac{\partial}{\partial w_{ji}^h} (y_{kp} - o_{kp})^2 \\ &= - \sum_p (y_{kp} - o_{kp}) \frac{\partial o_{kp}}{\partial (net_{kp}^o)} \frac{\partial (net_{kp}^o)}{\partial i_{kj}} \frac{\partial i_{kj}}{\partial (net_{kj}^h)} \frac{\partial (net_{kj}^h)}{\partial w_{ji}^h}\end{aligned}$$

Algoritmo Backpropagation

♦ Etapas

- 5. Repetir os procedimentos para os pesos da Camada Intermediária.
 - Cada um dos fatores da equação pode ser calculado explicitamente das equações anteriores, assim como foi feito para o gradiente da camada de saída.
 - O resultado fica:

$$\frac{\partial E_k}{\partial w_{ji}^h} = - \sum_p (y_{kp} - o_{kp}) f_p^{o'}(net_{kp}^o) w_{pj}^o f_j^{h'}(net_{kj}^h) x_{ki}$$

Algoritmo Backpropagation

♦ Etapas

- 5. Repetir os procedimentos para os pesos da Camada Intermediária.
 - Por fim, assim como no caso da camada de saída, atualizamos os pesos da camada escondida proporcionalmente ao valor negativo da equação.

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \Delta_k w_{ji}^h(t)$$

- onde:

$$\Delta_k w_{ji}^h = \eta f_j^{h'}(net_{kj}^h) x_{ki} \sum_p (y_{kp} - o_{kp}) f_p^{o'}(net_{kp}^o) w_{pj}^o$$

- η novamente é a taxa de aprendizado.

Algoritmo Backpropagation

♦ Etapas

- 6. Voltar ao passo 1, escolhendo um novo vetor de entrada do conjunto de treinamento e repetir os passos de 1 a 5, somando o erro.
- 7. Após todos os vetores de entrada do conjunto de treinamento terem sido apresentados (uma "época"), calcular o erro médio quadrático.

Se for aceitável parar,
senão voltar ao passo 1.

Algoritmo Backpropagation

- ♦ O que são mínimos locais?
 - São pequenos "buracos" na superfície de erro, mas não são na realidade a "solução" (fundo do poço) do problema.
- ♦ Como escapar de mínimos locais?
 - Podemos escapar de mínimos locais usando na atualização dos pesos um termo proporcional a última direção de alteração do peso. (Alteração do Peso no passo anterior do algoritmo Backpropagation) - idéia de inércia ou um "empurrão" para sair de buracos.

$$w_{pj}^o(t+1) = w_{pj}^o(t) + \Delta_k w_{pj}^o(t) + \alpha \Delta_k w_{pj}^o(t-1)$$

- onde α é o parâmetro conhecido como "momento".

Algoritmo Backpropagation

- ♦ Algumas dicas práticas
 - Inicialização dos valores dos pesos
 - valores aleatórios entre -1 e +1.
 - Valor da taxa de aprendizado η
 - valores pequenos 0.01 e 0.1
 - Se a função de saída for sigmoidal, escalar os valores de saída.
 - As saídas nunca atingem exatamente 0 ou 1. Usar valores como 0.1 e 0.9 para representar o menor e o maior valor de saída.
 - Se a função de saída for tangente hiperbólica, escalar os valores de saída.
 - As saídas nunca atingem exatamente -1 ou 1. Usar valores como -0.9 e 0.9 para representar o menor e o maior valor de saída.

Algoritmo Backpropagation

- ♦ Algumas dicas práticas
 - Valor do parâmetro de momento?
 - Valores "grandes" entre 0.8 e 0.9.
 - O que acontece se usarmos taxas de aprendizado muito grandes?
 - Converge rápido, mas pode não chegar no valor de erro mínimo, pois fica "saltando" de um lado para outro na superfície de erros.
 - Quantos neurônios na camada intermediária?
 - Quantas camadas intermediárias?

Redes Neurais Artificiais

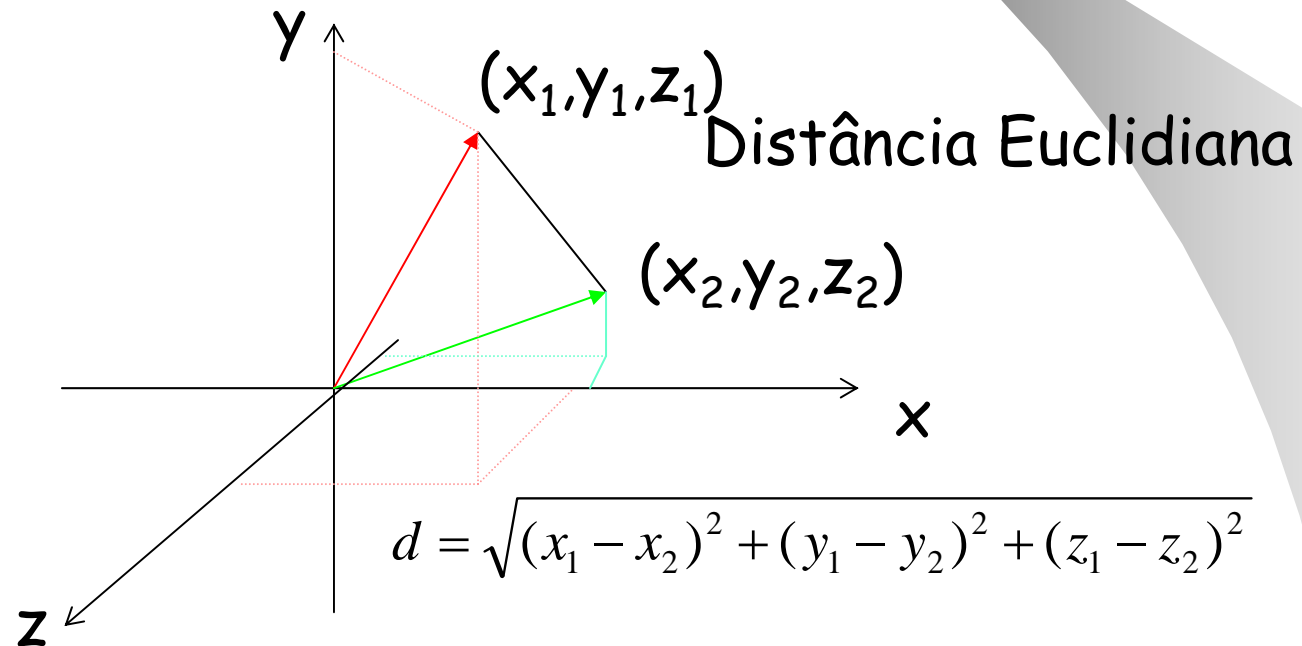
Memórias Associativas

- ♦ Conceitos Básicos - Memória Associativa
 - É um conceito "intuitivo".
 - Parece ser uma das funções primárias do cérebro.
 - Facilmente ASSOCIAMOS a face de um amigo com seu nome, ou um nome a um número de telefone.
 - Também serve para reconstituir padrões "corrompidos" ou incompletos.
 - Se olharmos uma foto com os lábios da Natasha Kinsky, logo recompomos todo o seu rosto.
 - Se vemos um amigo que normalmente não usa óculos, com eles, ainda assim, reconhecemos a face como sendo da pessoa em questão.
 - Recuperação de informação pelo "conteúdo"

Redes Neurais Artificiais

Memórias Associativas

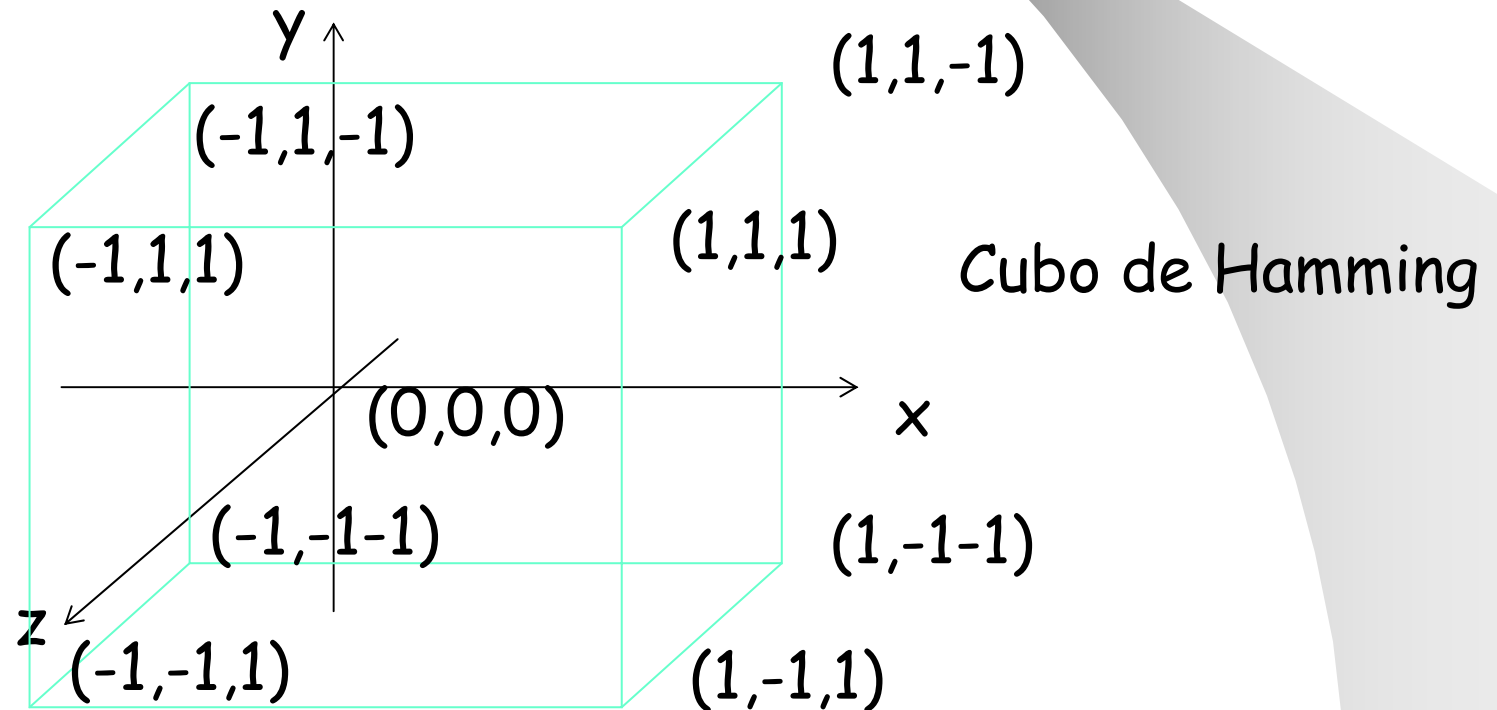
- ♦ Definições Iniciais
 - Algumas Medidas de Distância
 - Distância no Espaço Euclidiano



Redes Neurais Artificiais

Memórias Associativas

- ♦ Definições Iniciais
 - Algumas Medidas de Distância
 - Distância no Espaço de Hamming



Redes Neurais Artificiais

Memórias Associativas

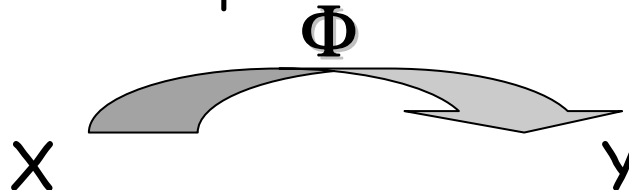
- ♦ Definições Iniciais
 - Distância de Hamming X Distância Euclidiana
 - Dados dois pontos $(x_i, y_i) \in \{-1, +1\}$
 - Distância Euclidiana
 - $(x_1 - x_2)^2 \in \{0, 4\}$
 - Distância Euclidiana: $d = \sqrt{4(\# \text{ "desencontros" })}$
 - Distância de Hamming
 - Distância de Hamming: $h = \# \text{ "desencontros" }$

$$d = 2\sqrt{h}$$

Redes Neurais Artificiais

Memórias Associativas

- ♦ Memórias Associativas - Definição Formal
 - Associadores Lineares
 - Suponha que tenhamos L pares de vetores, $\{(X_1, Y_1), (X_2, Y_2), \dots, (X_L, Y_L)\}$, com $X_i \in \mathbb{R}^n$, e $Y_i \in \mathbb{R}^m$.
 - Chamamos a estes vetores **exemplares**.
 - O que desejamos com os Associadores Lineares é fazer um "mapeamento" Φ de X para Y .



- Podemos distinguir então 3 tipos de **MEMÓRIAS ASSOCIATIVAS**.

Redes Neurais Artificiais

Memórias Associativas

- Memórias Heteroassociativas

- Implementa um mapeamento Φ de X para Y tal que $\Phi(X_i) = Y_i$. Se X for o mais próximo (menor distância de Hamming) de X_i do que qualquer X_j , $j=1,2,\dots,L$, então $\Phi(X) = Y_i$.

- Memórias Associativas Interpolativas

- Implementa um mapeamento Φ de X para Y tal que $\Phi(X_i) = Y_i$. Mas se o vetor de entrada X diferir de um exemplar X_i por um vetor D , de forma que $X = X_i + D$, então a saída da memória também difere dos exemplares de saída por um vetor E , ou seja:

$$\Phi(X) = \Phi(X_i + D) = Y_i + E.$$

Redes Neurais Artificiais

Memórias Associativas

- Memórias Autoassociativas
 - Assume que $X_i = Y_i$ e implementa um mapeamento Φ de X para X tal que $\Phi(X_i) = X_i$. Se X for o mais próximo (menor distância de Hamming) de X_i do que qualquer X_j , $j=1,2,\dots,L$, então $\Phi(X) = X_i$.

Redes Neurais Artificiais

Memórias Associativas

- ♦ Implementação Matemática

- Construir uma memória associativa não é difícil se introduzirmos a restrição de que os vetores exemplares devam ser "ortonormais" entre si (vetores dos vértices de um Cubo de Hamming).

- $X_i^T \cdot X_j = \delta_{ij}$, onde $\delta_{ij} = 1$ se $i=j$, e $\delta_{ij}=0$ se $i \neq j$.

$$\Phi(X) = (Y_1 X_1^T + Y_2 X_2^T + \dots + Y_L X_L^T) \cdot X$$

- Exemplo

$$\Phi(X_2) = (Y_1 X_1^T + Y_2 X_2^T + \dots + Y_L X_L^T) \cdot X_2$$

$$\Phi(X_2) = Y_1 X_1^T X_2 + Y_2 X_2^T X_2 + \dots + Y_L X_L^T X_2$$

$$\Phi(X_2) = Y_1 \delta_{12} + Y_2 \delta_{22} + \dots + Y_L \delta_{L2}$$

$$\delta_{ij}=0 \quad i \neq j, \quad \delta_{ij}=1 \quad i=j$$

$$\Phi(X_2) = Y_2$$

Redes Neurais Artificiais

Memórias Associativas

- ♦ Implementação por Redes Neurais (Elementos Processadores Distribuídos)

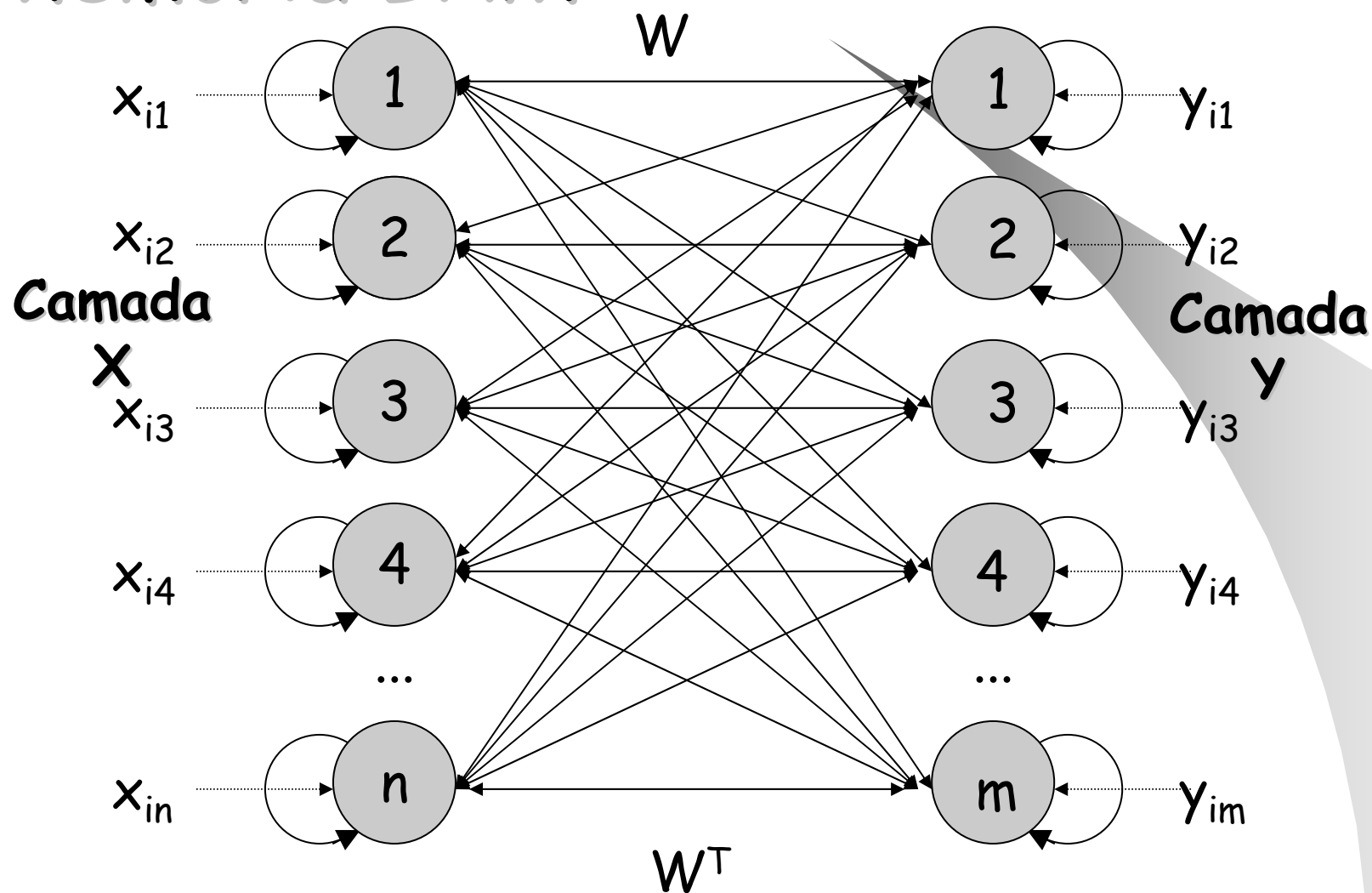
A MEMÓRIA BAM

(Bidirectional Associative Memory)

- Consiste de duas camadas de neurônios que estão completamente conectados entre as camadas. Cada neurônio está conectados a si mesmo.
- O peso das conexões é determinado a-priori, baseado nos vetores de "treinamento".

Redes Neurais Artificiais

Memória BAM



Redes Neurais Artificiais

Memória BAM

- ♦ "Treinamento" da BAM

- A matriz de pesos W é construída utilizando o modelo de Associador Linear.

- Dados L pares de vetores que constituem o conjunto de exemplares que desejamos armazenar,

$$W = Y_1 X_1^T + Y_2 X_2^T + Y_3 X_3^T + \dots + Y_L X_L^T$$

fornece os pesos das conexões da camada X para a camada Y .

$$W^T$$

fornece os pesos das conexões da camada Y para a camada X .

Redes Neurais Artificiais

Memória BAM

- ♦ Processamento da BAM
 - Cálculo das Matrizes de Pesos
 - Recuperação da Informação
 1. Aplicar um par de vetores iniciais (X_0, Y_0) .
 2. Propagar a informação de X para Y (multiplicar X_0 pela matriz W) e atualizar Y .
 3. Propagar a informação atualizada de Y para X (multiplicar Y pela matriz W^T) e atualizar X .
 4. Repetir os passos 2 e 3 até não haver mudança nos valores dos neurônios.
 - Dado X_0 o resultado será X_i com a menor distância de Hamming de X_0 .

Redes Neurais Artificiais

Memória BAM

- ♦ Algumas Considerações
 - É este algoritmo que fornece à BAM suas características bi-direcionais.
 - Os termos "entrada" e "saída" dependem da direção atual de propagação.
 - Após algumas iterações a rede irá estabilizar em um estado estável.
 - Não sobrecarregar demais a memória com muita informação, ou ela irá estabilizar em um "estado espúrio" (crosstalk).
 - O Crosstalk ocorre quando os padrões exemplares estão muito "próximos" uns dos outros.

Redes Neurais Artificiais

Memória BAM

- ♦ Matemática da BAM
 - Os neurônios calculam o net, como neurônios "normais":

cálculo do net para o neurônio i da camada Y .

$$net_i^Y = \sum_{j=1}^n w_{ij} x_j$$

cálculo do net para a camada Y .

$$net^Y = W.X$$

Redes Neurais Artificiais

Memória BAM

- ♦ Matemática da BAM

- Os neurônios calculam o net, como neurônios "normais":

cálculo do net para o neurônio j da camada X .

$$net_j^X = \sum_{i=1}^m w_{ji} y_i$$

cálculo do net para a camada X .

$$net^X = W^T . Y$$

Redes Neurais Artificiais

Memória BAM

- ♦ Matemática da BAM

- O valor da saída para cada neurônio depende do valor do net e do valor da saída "anterior".

Novo valor para a saída y no instante $t + 1$

está relacionada com o valor de y no instante t por

$$y_i(t+1) = \begin{cases} +1 & \text{se } net_i^y > 0 \\ y_i(t) & \text{se } net_i^y = 0 \\ -1 & \text{se } net_i^y < 0 \end{cases}$$

Novo valor para a saída x no instante $t + 1$

está relacionada com o valor de x no instante t por

$$x_i(t+1) = \begin{cases} +1 & \text{se } net_i^x > 0 \\ x_i(t) & \text{se } net_i^x = 0 \\ -1 & \text{se } net_i^x < 0 \end{cases}$$

Redes Neurais Artificiais

Memória BAM

- ♦ Alguns exemplos

$$X_1 = [1, -1, -1, 1, -1, 1, 1, -1, -1, 1]^T \text{ e } Y_1 = [1, -1, -1, -1, -1, 1]^T$$

$$X_2 = [1, 1, 1, -1, -1, -1, 1, 1, -1, -1]^T \text{ e } Y_2 = [1, 1, 1, 1, -1, -1]^T$$

$$W = Y_1 X_1^T + Y_2 X_2^T$$

$$W = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$

Redes Neurais Artificiais

Memória BAM

♦ Exemplo - continuação

- Tomemos $X_0 = [-1, -1, -1, 1, -1, 1, 1, -1, -1, 1]^T$
- $h(X_0, X_1) = 1$ e $h(X_0, X_2) = 7$ (distância de Hamming)
- Fazemos Y_0 igual a um dos vetores exemplares (ou usamos um vetor bipolar aleatório) $Y_0 = Y_2 = [1, 1, 1, 1, -1, -1]^T$.
- Propagamos de X para Y , já que a "chave" é X_0 .
- $net^Y = W.X_0 = [4, -12, -12, -12, 4, 12]^T$
- Calculamos o novo vetor Y , $Y_{new} = [1, -1, -1, -1, 1, 1]^T$
- Propagamos agora de Y para X .
- $net^X = [4, -8, -8, 8, -4, 8, 4, -8, -4, 8]^T$
- Calculamos o novo vetor X , $X_{new} = [1, -1, -1, 1, -1, 1, 1, -1, -1, 1]^T$
- Novas propagações não alterarão o resultado e portanto consideramos a memória estabilizada e com o padrão X_1 recuperado.

Redes Neurais Artificiais

Memória BAM

♦ Exemplo 2

- Tomemos $X_0 = [-1, 1, 1, -1, 1, 1, 1, -1, 1, -1]^T$ e $Y_0 = [-1, 1, -1, 1, -1, -1]^T$
- $h(X_0, X_1) = 7$ e $h(X_0, X_2) = 5$ (distância de Hamming)
- $h(Y_0, Y_1) = 4$ e $h(Y_0, Y_2) = 2$
- Propagamos de X para Y , já que a "chave" é X_0 .
- $net^Y = W.X_0 = [-4, 4, 4, 4, -4]^T$
- Calculamos o novo vetor Y , $Y_{new} = [-1, 1, 1, 1, 1, -1]^T$
- Propagamos agora de Y para X .
- $net^X = [-4, 8, 8, -8, 4, -8, -4, 8, 4, -8]^T$
- Calculamos o novo vetor X , $X_{new} = [-1, 1, 1, -1, 1, -1, -1, 1, 1, -1]^T$
- Novas propagações não alterarão o resultado e portanto consideramos a memória estabilizada.
- **Qual foi o padrão recuperado?**

Redes Neurais Artificiais

Memória BAM

- ♦ Exemplo 2 - continuação
 - O padrão recuperado foi o "complemento" de X_1 .
 - Propriedade básica da BAM: Se armazenamos um padrão (X,Y) , automaticamente armazenamos o complemento do padrão.
 - Não é comum uma criança dizer "apaga" a luz quando na verdade quer acender a luz?

Computação Evolucionária (ou Evolutiva)

- ♦ Métodos de Resolução de Problemas
 - **Métodos Fortes:** são concebidos para resolverem problemas genéricos, mas foram desenvolvidos para operarem em um mundo específico, onde impera linearidade, continuidade, diferenciabilidade e/ou estacionariedade.
 - **Métodos Específicos:** são concebidos para resolverem problemas específicos em mundos específicos.

Computação Evolucionária (ou Evolutiva)

- ♦ Métodos de Resolução de Problemas
 - **Métodos Fracos:** são concebidos para resolverem problemas genéricos em mundos genéricos. Operam em mundos não-lineares e não-estacionários, embora não garantam eficiência total na obtenção da solução. No entanto, geralmente garantem a obtenção de uma "boa aproximação" para a solução, em um tempo que aumenta a uma taxa menor que exponencial com o aumento do "tamanho" do problema.

Computação Evolucionária (ou Evolutiva)

- ♦ Independente da aplicação, métodos fracos (soluções baseadas em computação evolutiva) devem ser considerados se e somente se métodos fortes (soluções clássicas) e métodos específicos (soluções dedicadas) não existem, não se aplicam, ou falham quando aplicados.
- ♦ **conclusão:** soluções baseadas em computação evolutiva devem ser consideradas como o último recurso (reforçado ainda mais pela atual imaturidade desta área de pesquisa).
- ♦ **alento:** subtraindo-se os problemas tratáveis pelos métodos fortes e métodos específicos, o campo de aplicação para técnicas de computação evolutiva continua sendo extremamente vasto (nem tão vasto assim se considerarmos as restrições impostas pelo seu nível de maturidade).

Computação Evolucionária (ou Evolutiva)

"...Se **variações úteis** para qualquer organismo devam ocorrer para que ele venha a existir, certamente indivíduos assim caracterizados terão a **melhor chance** de serem preservados na luta por sobrevivência; e do forte princípio de **hereditariedade**, eles tenderão a produzir gerações com características similares. Este princípio de **preservação**, eu batizei, para ser sucinto, de **Seleção Natural**."

(Darwin, 1859)

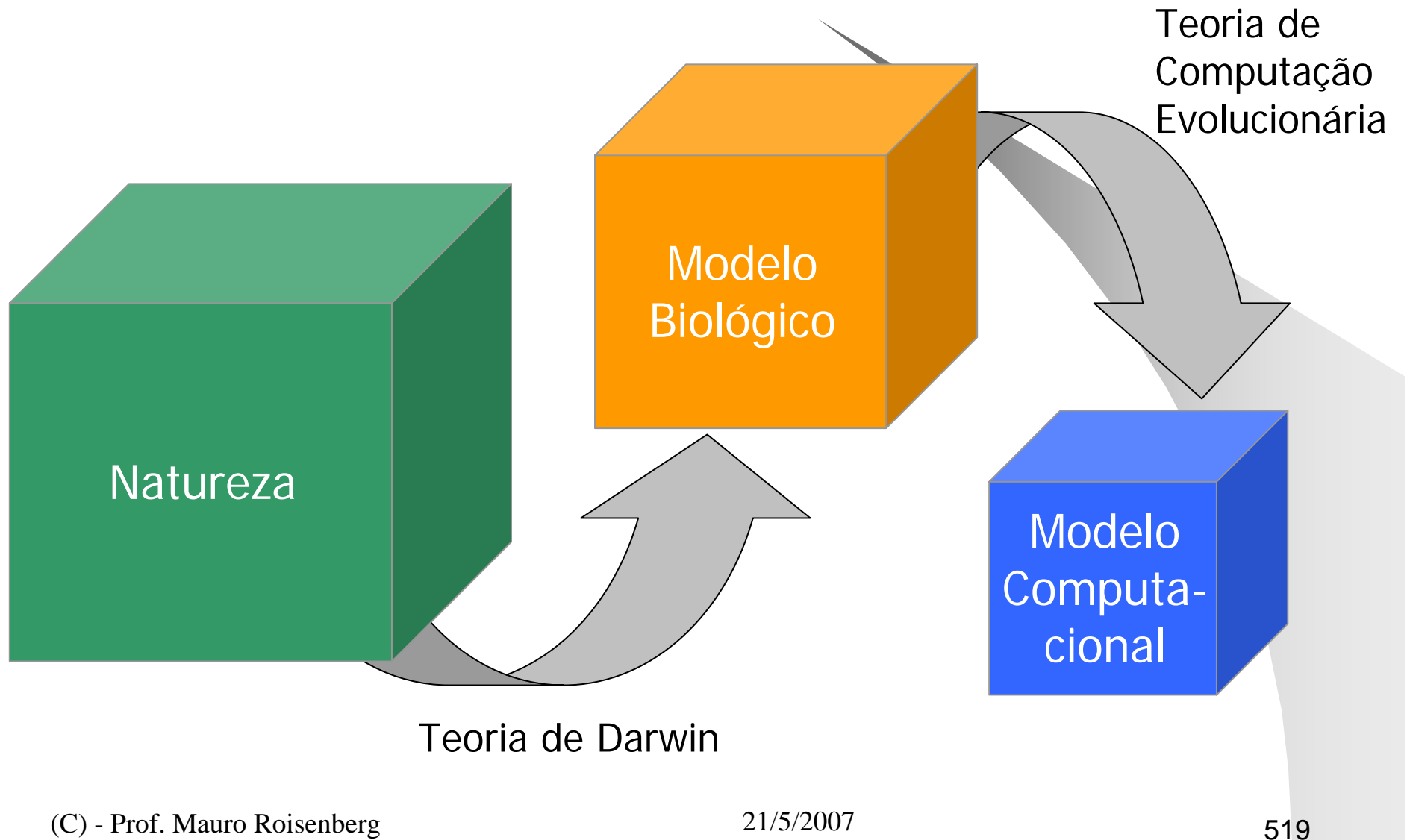
Computação Evolucionária (ou Evolutiva)

- ♦ Inspirada nos processos subjacentes à evolução:
 - Moldar uma população de indivíduos através da sobrevivência de seus membros mais ajustados.
 - Pressões seletivas não surgem apenas do ambiente externo, mas também de interações entre membros de uma população.

Computação Evolucionária (ou Evolutiva)

- ♦ Algoritmos genéticos, sistemas classificadores, estratégias evolutivas, programação genética e programação evolutiva
 - A computação evolutiva pode desempenhar os seguintes papéis básicos:
 - 1. ferramenta adaptativa para a solução de problemas;
 - 2. modelo computacional de processos evolutivos naturais.
- ♦ **Sistemas Naturais:** metáfora diretora, fonte de inspiração.

Computação Evolucionária (ou Evolutiva)



Fundamentos dos AGs

- ♦ Os algoritmos genéticos e outras analogias evolucionárias formais produzem soluções para problemas com capacidade crescente, operando sobre populações de soluções candidatas para o problema.
- ♦ **Uma Definição**
 - Algoritmos Genéticos (Ags) são métodos computacionais de busca baseados nos mecanismos de evolução natural e na genética. Em Ags, uma população de possíveis soluções para o problema em questão evolui de acordo com operadores probabilísticos concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua.

Fundamentos dos AGs

- ♦ Foram desenvolvidos por John Holland, University of Michigan (1970's) com o objetivo de:
 - Entender os processos adaptativos dos sistemas naturais
 - Projetar sistemas artificiais (software) que possuíssem a robustez dos sistemas naturais
- ♦ Provêem uma técnica eficiente e efetiva para otimização e aplicações de aprendizado de máquina
- ♦ Largamente utilizados atualmente em negócios e problemas científicos e de engenharia

Componentes de um AG

Um problema a resolver, e ...

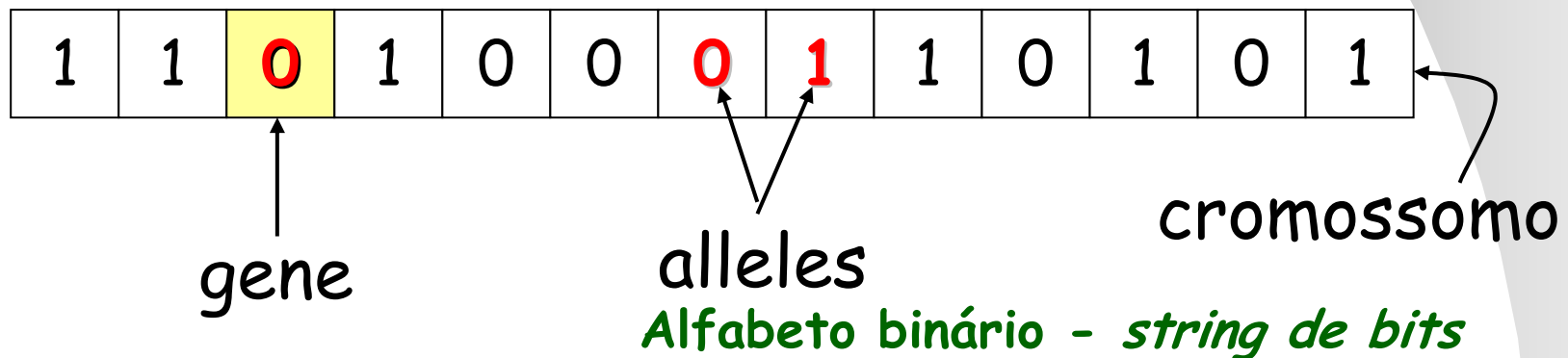
- ♦ Técnica de codificação *(gene, cromossomo)*
- ♦ Procedimento de Inicialização *(genesis)*
- ♦ Função de Avaliação *(meio ambiente)*
- ♦ Seleção de pais *(reprodução)*
- ♦ Operadores Genéticos *(mutação, recombinação)*
- ♦ Ajuste de Parâmetros *(prática e arte)*

Características Primárias

- ♦ AGs operam numa população (conjunto) de pontos, e não a partir de um ponto isolado.
- ♦ AGs operam num espaço de soluções codificadas, e não no espaço de busca diretamente.
- ♦ AGs necessitam somente de informação sobre o valor de uma função objetivo para cada membro da população, e não requerem derivadas ou qualquer outro tipo de conhecimento.
- ♦ AGs usam transições probabilísticas, e não regras determinísticas.

Representação Cromossômica

- Para poder aplicar AGs deve-se primeiramente representar cada possível solução x no espaço de busca como uma sequência de símbolos s gerados a partir de um dado alfabeto finito A .
- Geralmente usa-se o alfabeto binário $A=\{0,1\}$, mas no caso geral tanto o método de representação quanto o alfabeto dependem de cada problema.
- Cada sequência s corresponde a um cromossomo.
- Cada elemento de s é equivalente a um gene.



Representação Cromossômica

Outras representações:

- ♦ com números reais: (43.2 -33.1 ... 0.0 89.2)
- ♦ *Strings* simbólicas: ©⌘☼❖☆☯ ...
- ♦ Permutações de elementos (E11 E3 E7 ... E1 E15)
- ♦ Listas de regras (R1 R2 R3 ... R22 R23)
- ♦ Programas (para programação genética)
- ♦ Qualquer estrutura de dados

Representação Cromossômica

- ♦ Na maior parte dos AGs assume-se que cada indivíduo seja constituído de um único cromossomo.
- ♦ A grande maioria dos AGs propostos na literatura usam uma população de número fixo de indivíduos, com cromossomos também de tamanho constante.

Representação Cromossômica

- ♦ Para cada problema, encontrar uma representação cromossômica conveniente é sempre o primeiro passo.
- ♦ Usa-se um vetor binário para representar cada ponto no espaço de busca.
- ♦ Como há um número infinito de pontos no intervalo de interesse, a dimensão deste vetor ou sequência binária depende da precisão requerida para o problema.

Representação Cromossômica

- ♦ Assumamos, como exemplo, uma precisão de 4 casas com um espaço de busca entre os números -2 até +2.
- ♦ Tal precisão significa que o processo de busca deve distinguir pelo menos $4 \times 10.000 = 40.000$ pontos.
- ♦ Como resultado, seqüências binárias (cromossomos) de 16 bits são necessários, uma vez que $40.000 < 2_{16} = 65.536$.
- ♦ Divide-se assim, o intervalo de busca em 65.536 partes iguais.

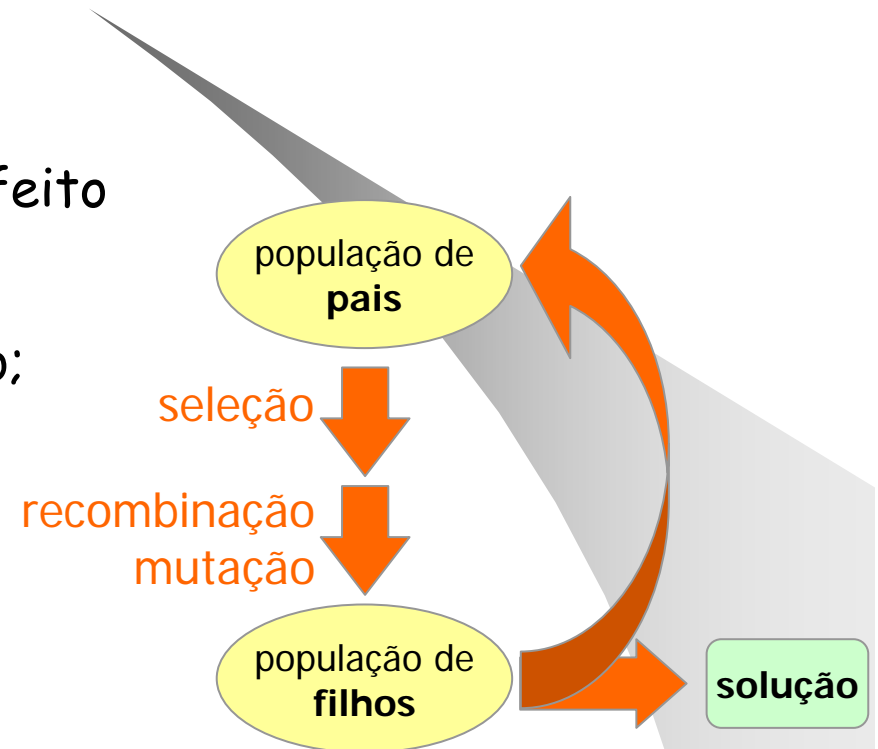
Representação Cromossômica

- Assim, cada possível solução x será representada por uma seqüência $s=[b_{15}b_{14}...b_2b_1b_0]$ onde cada $b \in \{0,1\}$.
- Primeiro converte-se o valor do cromossomo para um valor em base 10.
- Em seguida, o número é mapeado de volta ao espaço de busca de acordo com:

$$x = -x_{\min} + \frac{x_{\max} - x_{\min}}{2^{16} - 1} x$$

Fluxo Básico

```
{  
  inicializar população;  
  avaliar população;  
  while  
    Criterio_de_Termino_Nao_Satisfeito  
  {  
    selecionar pais para reprodução;  
    realizar recombinação e mutação;  
    avaliar população;  
  }  
}
```



- ♦ Implementações específicas do algoritmo particularizam esta estrutura de diferentes maneiras.

Fluxo Básico

♦ Inicialização

- Geralmente a população inicial de N indivíduos é gerada aleatoriamente ou através de algum processo heurístico.
- É importante que a população inicial cubra a maior parte possível do espaço de busca.

Fluxo Básico

♦ Avaliação e Adequabilidade

- Afs necessitam da informação do valor de uma função objetivo para cada membro da população, que deve ser um valor não negativo.
- Nos casos mais simples, usa-se o próprio valor da função que se quer maximizar.
- A função objetivo dá, para cada indivíduo, uma medida de quão bem adaptado ao ambiente ele está.
- A avaliação de cada indivíduo resulta num valor denominado de fitness.
- Quanto maior o fitness, maiores são as chances do indivíduo sobreviver e se reproduzir.

Fluxo Básico

♦ Seleção

- Emula os processos de reprodução assexuada e seleção natural.
- Em geral, gera-se uma população temporária de N indivíduos extraídos com probabilidade proporcional ao fitness relativo de cada indivíduo na população.
- A probabilidade de seleção de um cromossomo S é dada por:

$$P_{sel}(s) = \frac{f(s)}{\sum_{j=1}^{|P|} f(s_j)}$$

Fluxo Básico

♦ Exemplo: Seleção dos pais

<i>Chromosome</i>	<i>Chromosome Value</i>	<i>Evaluation (fitness)</i>	<i>% of Total</i>
1	10111	201	51
2	11001	136	35
3	11110	20	5
4	00010	34	9
Total:		391	100

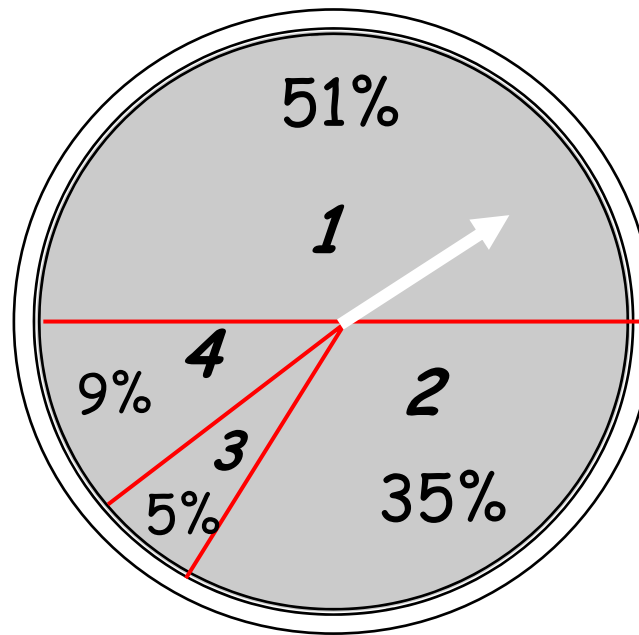
Fluxo Básico

♦ Seleção

- Neste processo, indivíduos com baixo fitness (adequabilidade) terão alta probabilidade de desaparecerem da população (serem extintos).
- Indivíduos adequados terão grandes chances de sobreviverem.
- Como a seleção é probabilística, membros fracos recebem menores probabilidades, mas não são diretamente eliminados.
- É importante que alguns candidato menos aptos sobrevivam, pois eles podem ainda conter algum componente essencial de uma solução.

Fluxo Básico

- ♦ Exemplo: Seleção dos pais
 - A seleção dos pais é realizada usando-se uma roleta com tamanhos de setores de acordo com os valores obtidos na avaliação dos cromossomos.

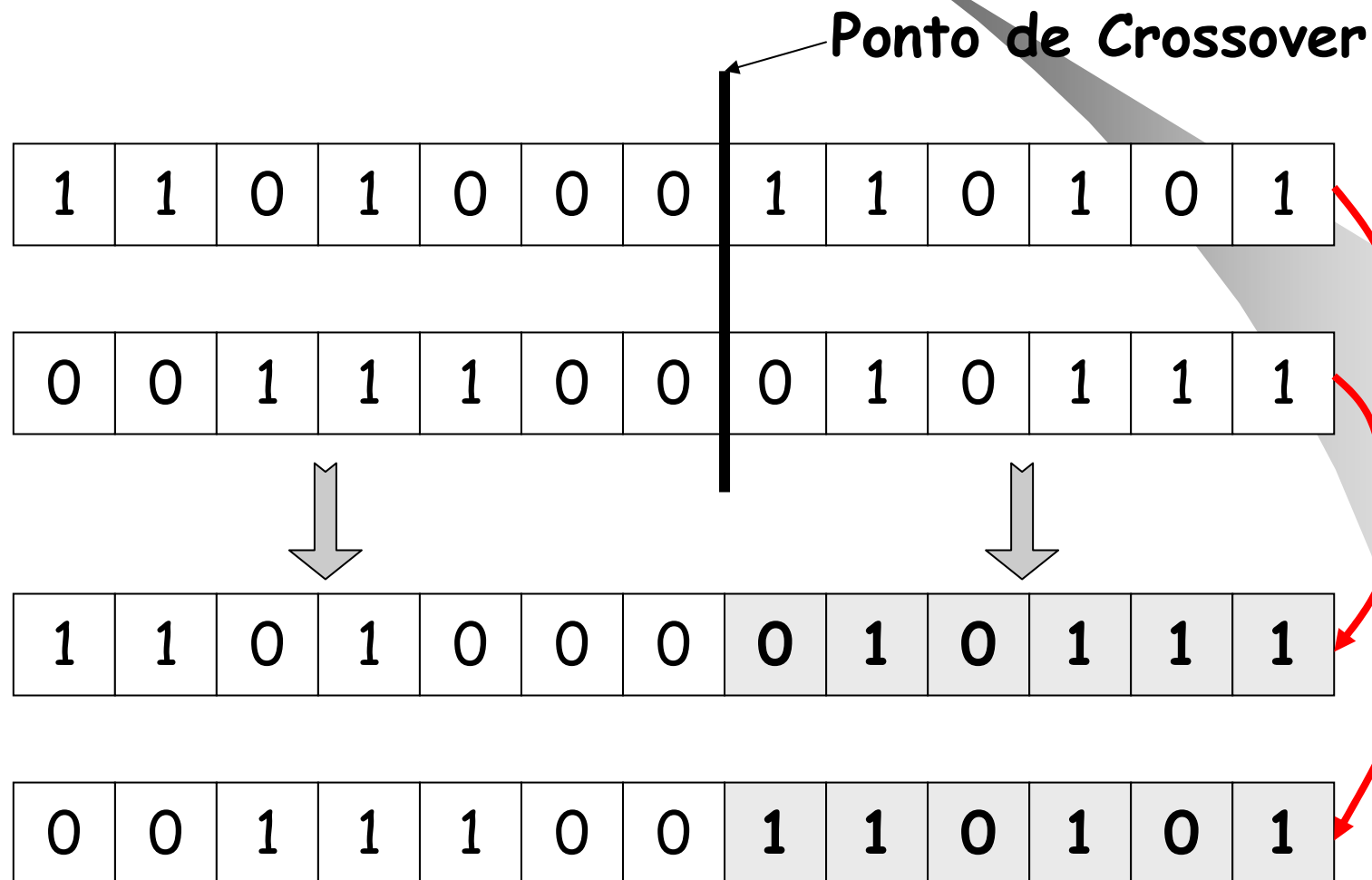


Fluxo Básico

- ♦ **Recombinação (crossover)**
 - É um processo sexuadao - ou seja, envolve mais de um indivíduo.
 - Emula o fenômeno de "crossover", a troca de fragmentos entre pares de cromossomos.
 - Na forma mais simples, é um processo aleatório que ocorre com probabilidade fixa p_{rec} que deve ser especificada pelo usuário.
 - O ponto de divisão pode ser ajustado aleatoriamente ou mudado durante o processo de solução.

Fluxo Básico

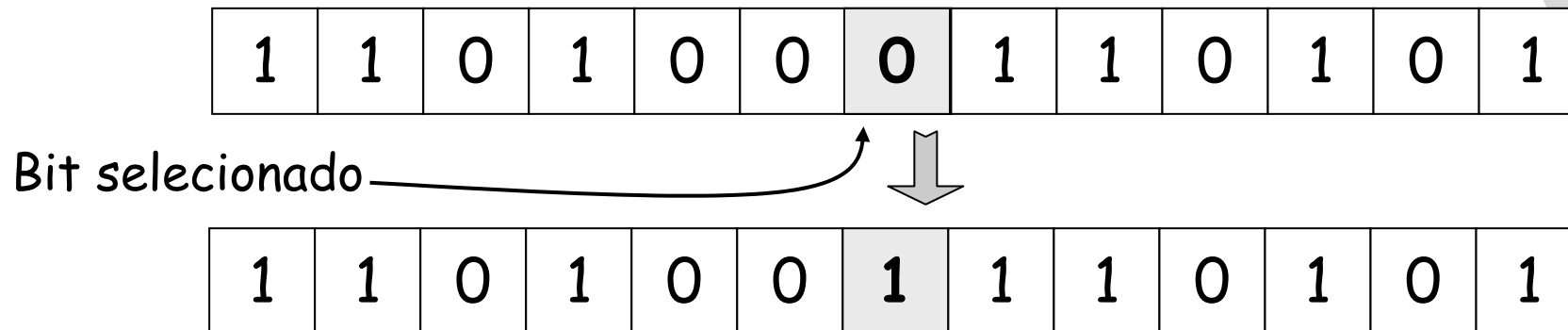
- Recombinação (crossover)



Fluxo Básico

- ♦ Mutação

- O processo de mutação é equivalente à busca aleatória.
- Seleciona-se uma posição num cromossomo e muda-se o valor do gene correspondente aleatoriamente para um outro alelo possível.
- O processo é geralmente controlado por um parâmetro fixo p_{mut} que indica a probabilidade de um gene sofrer mutação.



Fluxo Básico

♦ Mutação

- É um processo importante, pois a população inicial pode excluir um componente essencial da solução.
- O processo de reprodução, após uma série de iterações, tende a um equilíbrio estático. A mutação tende quebrar a estacionariedade, incorporando aspectos de criatividade no AG.
- Se a mutação não gerar um bom resultado, a probabilidade de reprodução será bastante pequena, porém, sendo bom poderá trazer mudanças radicais no processo de busca da solução.
- O mecanismo de mutação pode levar a resultados, que possivelmente não eram esperados.

Fluxo Básico

♦ Condições de Término

- Como normalmente estamos tratando de problemas de otimização, o ideal seria que o algoritmo terminasse assim que o ponto ótimo fosse descoberto.
- Pode haver situações onde todos ou o maior número possível de pontos ótimos sejam desejados.
- Na prática raramente se pode afirmar se um dado ponto ótimo corresponde a um ótimo global.

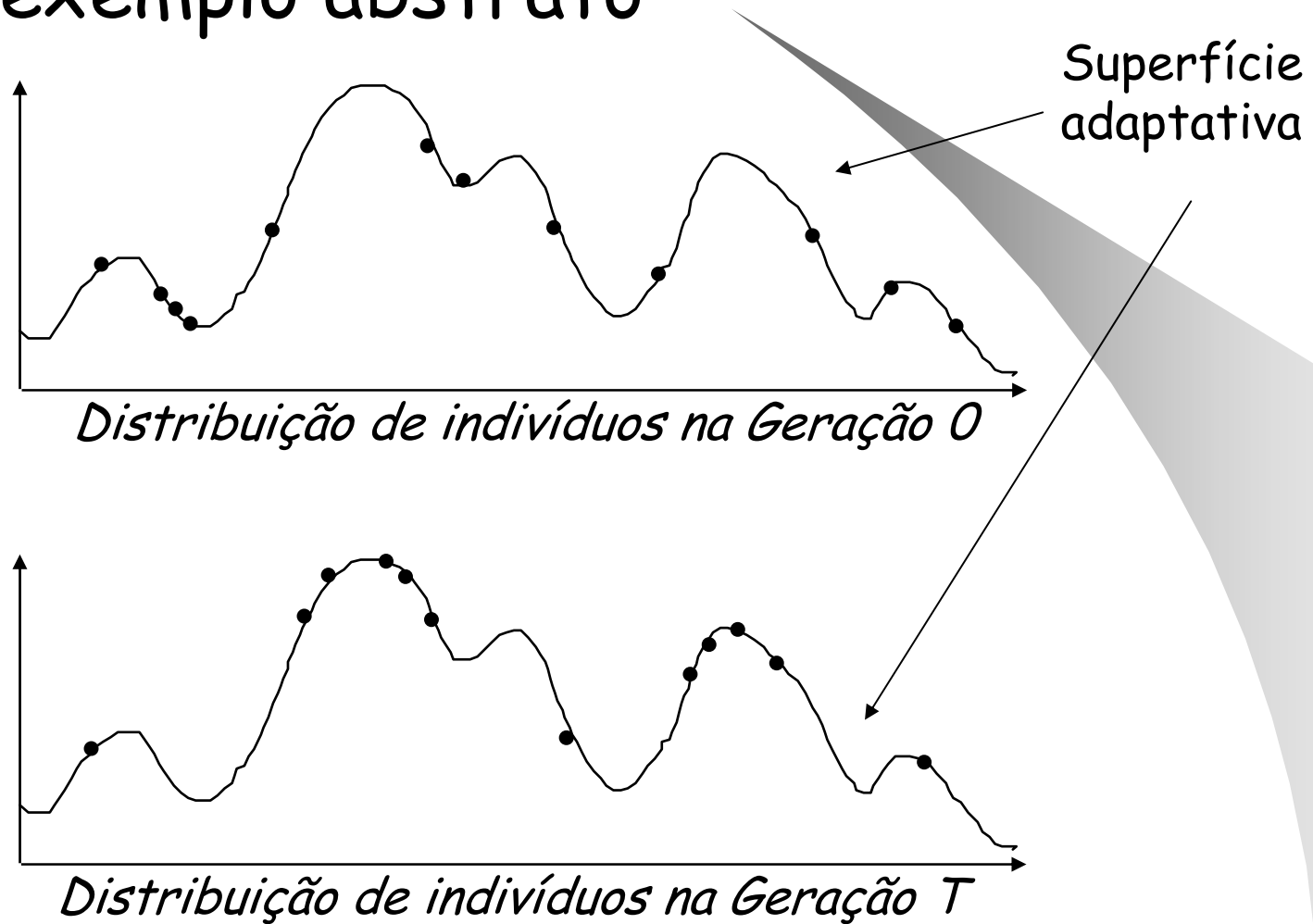
Fluxo Básico

- ♦ Condições de Término

- Normalmente usa-se o critério de número máximo de gerações ou um tempo limite de processamento para parar um AG.
- Outro critério usa a idéia de estagnação, ou, seja, para-se o algoritmo quando não se observa melhoria da população depois de várias gerações.

Fluxo Básico

- ♦ Um exemplo abstrato



Exemplo

- Imagine uma população inicial de 4 indivíduos, cada um representado por uma cadeia de 10 bits.
- A função objetivo a maximizar é o número de bits em 1 em uma cadeia.
- Para normalizar a função objetivo a valores entre 0 e 1 divide-se por 10 o número de bits em 1 de cada indivíduo.

População P1:

String (indivíduos)	Valor de Aptidão
0000011100	0.3
1000011111	0.6
0110101011	0.6
1111111011	0.9

Exemplo

- Na população P1 os quatro indivíduos possuem valor de aptidão 0.3, 0.6, 0.6 e 0.9.
- Teoricamente o mecanismo de seleção proporcional deveria alocar 12%, 25%, 25% e 38% de descendentes para cada indivíduo.
- Neste caso a alocação final de decendentes é 0, 1, 1 e 2 respectivamente.

População P2: Após a Seleção

String (indivíduos)	Valor de Aptidão
1000011111	0.6
0110101011	0.6
1111111011	0.9
1111111011	0.9

Exemplo

- A seguir as 4 cadeias são pareadas randomicamente para cruzamento.
- As cadeias 1 e 4 formam um par e as cadeias 2 e 3 outro par.
- Com uma taxa de cruzamento de 0.5, apenas as cadeias 1 e 4 são selecionadas para cruzamento, enquanto as cadeias 2 e 3 são deixadas intactas.
- O ponto de cruzamento cai entre o quinto e o sexto bit das cadeias.
- Os bits 1 a 5 das cadeias são trocados.

População P2: Após a Seleção

String (indivíduos)	Valor de Aptidão
1000011111	0.6
0110101011	0.6
1111111011	0.9
1111111011	0.9

População P3: Após o Cruzamento

String (indivíduos)	Valor de Aptidão
10000 11011	0.5
0110101011	0.6
1111111011	0.9
11111 11111	1.0

Exemplo

- A operação de mutação na população P3 pode ser visto na população P4, no bit 10 do indivíduo 4.
- Apenas um bit de um total de 40 foi trocado, representando uma taxa de mutação de 0.025.
- A população P4 representa a próxima geração.

População P4: Após a Mutação

String (indivíduos)	Valor de Aptidão
1000011011	0.5
0110101011	0.6
1111111011	0.9
0111111111	0.9

Exemplo

- Este exemplo tem caráter apenas ilustrativo.
- Um *AG* típico usa uma população de 30 a 200 indivíduos, taxas de cruzamento de 0.5 a 0.9 e taxas de mutação de 0.001 a 0.05.

Algoritmo Genético

- Implementam mecanismos de evolução natural incluindo *cruzamento, mutação e aptidão para sobrevivência*.
- Trabalha com populações de indivíduos.
- Os indivíduos são submetidos a um processo de evolução.

Programação Evolutiva

- Concebida como uma estratégia de otimização similar aos AG's.
- Proposta original trata de predição de comportamento de máquinas de estado finitos, porém se adapta a qualquer estrutura de problema.
- Enfatiza o relacionamento *comportamental* entre progenitores e sua descendência, ao invés de tentar emular operadores genéticos específicos observados na natureza.

Programação Evolutiva

- Não implementam cruzamentos. Ao invés disso, confiam na aptidão para sobrevivência e mutação.
- Cada indivíduo gera descendentes através de mutação, e a seguir a (melhor) metade da população ascendente e a (melhor) metade da população descendente são reunidas para formar a nova geração.
- Não se exige que a população permaneça constante, ou que cada ascendente gere apenas um descendente.

Algoritmos

ALGORITMO GENÉTICO

```
{  
  inicializar população (P);  
  avaliar população;  
  while Criterio_Nao_Satisfeito  
  {  
    selecionar pais para reprodução;  
    realizar recombinação;  
    realizar mutação;  
    avaliar população;  
    P = sobreviventes;  
  }  
}
```

PROGRAMAÇÃO EVOLUTIVA

```
{  
  inicializar população (P);  
  avaliar população;  
  while Criterio_Nao_Satisfeito  
  {  
    realizar mutação;  
    avaliar população;  
    P = sobreviventes;  
  }  
}
```

Estratégias Evolutivas

- Concebidos para solucionar problemas técnicos de otimização.
- Por muito tempo, foi empregado quase que exclusivamente em engenharia civil, como uma alternativa às soluções convencionais. Normalmente não há uma função objetiva fechada para os PTO's e portanto nenhum método aplicável de otimização além da intuição do engenheiro.
- Em uma Estratégia de Evolução dupla (1+1), cada progenitor produz um decendente por geração através da aplicação de mutações normalmente distribuídas, onde pequenos passos ocorrem mais provavelmente que grandes saltos, até que o decendente exibe um desempenho superior ao progenitor e assume o seu lugar.
- Ênfase na auto-adaptação. O papel da recombinação é aceito, mas como operador secundário.

Estratégias Evolutivas

- $(\mu + 1)$ - uma população de indivíduos (μ) se recombina de maneira randômica para formar um descendente, o qual, após sofrer mutação, substitui (se for o caso) o pior elemento da população.
- Estratégia Soma: $(\mu + \lambda)$ - os ancestrais e os descendentes convivem.
- Estratégia Vírgula: (μ, λ) - os ancestrais "morrem", deixando apenas os descendentes "vivos".

Estratégias Evolutivas

- A abordagem é adequada a uma vasta gama de problemas de otimização, pois não necessita de muitas informações sobre o problema.
- É capaz de resolver problemas multidimensionais, multimodais e não lineares sujeitos a restrições lineares ou não lineares.

Estratégias Evolutivas

Algoritmo Básico:

- 1. Uma dada população consiste de μ indivíduos. Cada um é caracterizado pelo seu genótipo, consistindo de n genes, que determina de modo não ambíguo a aptidão para a sobrevivência.
- 2. Cada indivíduo da população produz (λ / μ) descendentes, na média, de modo que um total de indivíduos novos são gerados. O genótipo dos descendentes difere ligeiramente dos genótipos de seus ancestrais.
- 3. Apenas os μ melhores indivíduos dos λ gerados permanecem vivos, tornando-se os ancestrais na próxima geração.

Programação Genética

- Koza (1992) sugeriu que um programa de computador bem-sucedido poderia evoluir através de aplicações sucessivas de operadores genéticos.
- Na programação genética, as estruturas que são adaptadas são segmentos de programas organizados hierarquicamente.
- O algoritmo de aprendizagem mantém uma população de programas.
- A aptidão é medida pela capacidade de resolver um conjunto de tarefas e os programas são modificados aplicando recombinação e mutação de subárvores.

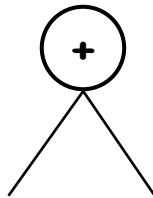
Programação Genética

- População inicial: pedaços de programas.
 - Adequados para um domínio de problema podem ser:
 - Operações aritméticas padrão, outras operações de programação e funções matemáticas, bem como funções lógicas e específicas do domínio.
- A produção de novos programas advém da aplicação de operadores genéticos.
- Qualquer programa que se sair bem, sobreviverá para ajudar a produzir os filhos da próxima geração.

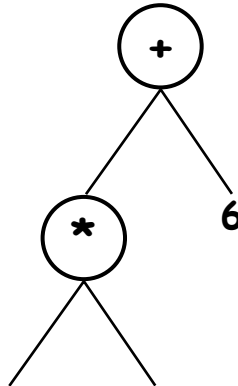
Programação Genética

- **Necessita-se:**
 - **F:** conjunto de Funções $\rightarrow +, *, -, /, \text{sen}(x), \text{cos}(x)$ ou operações de matrizes.
 - **T:** conjunto de Valores terminais.
- População de "programas" iniciais:

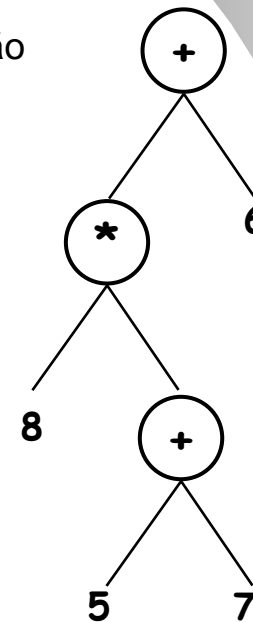
1a. Seleção



2a. Seleção

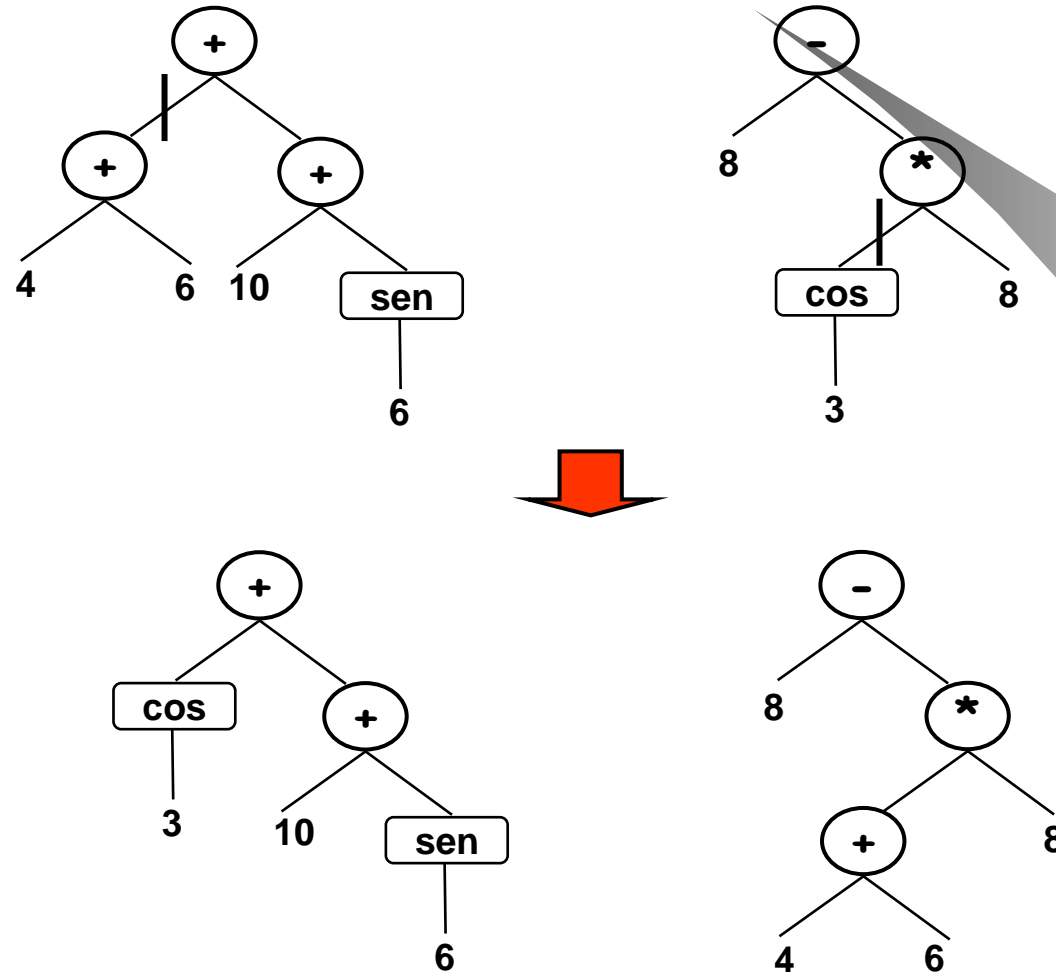


3a. Seleção



Programação Genética

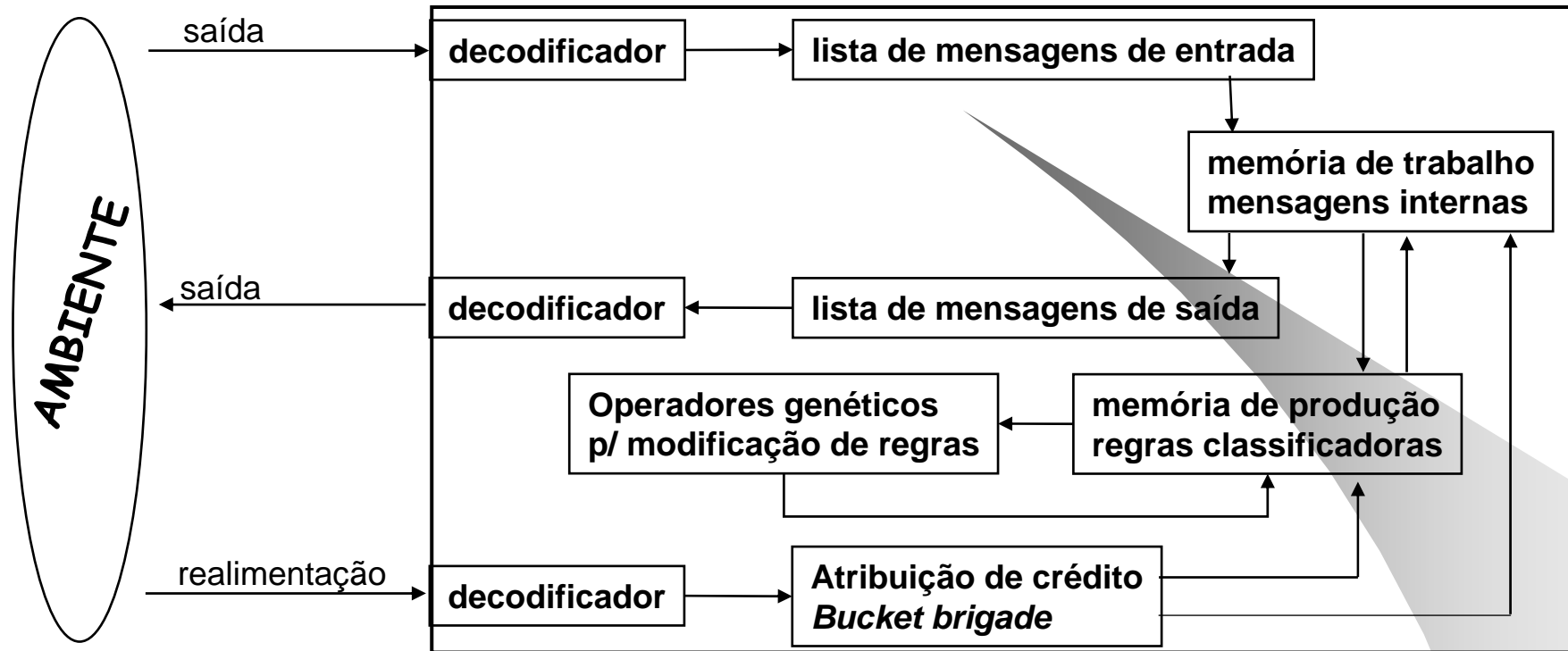
Recombinação:



Sistemas Classificadores

- Propostos por Holland (1986), como sistemas capazes de perceber e classificar os acontecimentos e reagir a eles.
- Aplica aprendizagem genética a regras de um sistema de produção.
- Para construção de tais sistemas é necessário:
 1. Um ambiente;
 2. Regras de produção → classificadores;
 3. Memória de Trabalho;
 4. Sensores de Entrada → decodificadores;
 5. Saídas → atuadores;
- A base de conhecimento gerada é tratada através de regras SE-ENTÃO.

Sistemas Classificadores



- ♦ Algoritmo *bucket brigade*: atribui crédito ou culpa a regras:
 - Para ações bem ou mal sucedidas
- ♦ Regras de produção: população de classificadores:
 - Cada classificador tem uma medida de aptidão;

Sistemas Classificadores

- ♦ Implementam uma forma de aprendizagem por reforço.
- ♦ Podem aprender de duas formas:
 1. Sistema de recompensa que ajusta as medidas de aptidão das regras classificadoras, recompensando disparos de regras bem-sucedidos e penalizando erros.
 2. Modificando as próprias regras usando operadores genéticos como recombinação e mutação.
 - Regras mais bem-sucedidas sobrevivem e se combinam para formar novos classificadores;
 - Classificadores com regras malsucedidas desapareçam.

Sistemas Classificadores - Exemplo

- ♦ Conjunto de objetos classificado por seis atributos (condições c_1, c_2, \dots, c_6).
- ♦ Cada atributo pode ter cinco valores $\{1, 2, \dots, 5\}$.
- ♦ Objetos podem ser de quatro classes possíveis
 - $(c_1 c_2 c_3 c_4 c_5 c_6) \rightarrow A_i$, onde $i = 1, 2, 3, 4$.

Sistemas Classificadores - Exemplo

Conjunto de Classificadores condição → ação a ser “aprendido”

Condição		Ação (Classificação)	No. Regras
(1 # # # 1 #)	→	A1	1
(2 # # 3 # #)	→	A1	2
(# # 4 3 # #)	→	A2	3
(1 # # # # #)	→	A2	4
(# # 4 3 # #)	→	A3	5
etc.

- ♦ Desempenho ao aprender a classificação A1:

(1 # # # 1 #) → (1 0 0 0)

(1 # # # # #) → (0 1 0 0)

(# # 4 3 # #) → (0 1 1 0) (atributos dão suporte às regras A2 e A3)

Sistemas Classificadores - Exemplo

- ♦ 1. População aleatória. (padrão de força, s entre 0 e 1)

(# # # 2 1 #) \rightarrow 1 $s = 0,6$

(# # 3 # # 5) \rightarrow 0 $s = 0,5$

(2 1 # # # #) \rightarrow 1 $s = 0,4$

(# 4 # # # 2) \rightarrow 0 $s = 0,23$

Onde: 1 levou a uma classificação correta e 0 a uma errada

- ♦ 2. Entrada do ambiente: (1 4 3 2 1 5)

- Casa com as regras 1 e 2:

- 0,5: não importa | 1: casamentos exatos.

- Regra 1: $((4*0,5 + 2*1) * 0,6) / 6 = 0,4$

- Regra 2: $((4*0,5 + 2*1) * 0,5) / 6 = 0,33$

Sistemas Classificadores - Exemplo

- ♦ 3. Primeira regra é selecionada (maior lance).
 - Comunica sua ação (1), indicando que este padrão é um exemplo de A1.
- ♦ 4. Medida de aptidão é aumentada para um valor entre o atual e 1,0. (se a ação fosse errada seria diminuído).
- ♦ 5. Aplicação de operadores genéticos para criar a próxima geração de regras.

R1. $(\# \# \# 2 \mid 1 \#) \rightarrow 1$ $s = 0,6$

R2. $(\# \# 3 \# \mid \# 5) \rightarrow 0$ $s = 0,5$

Descendentes:

$$(\# \# 3 \# \mid 1 \#) \rightarrow 0 \quad s = 0,53 \quad (1/3 * 0,6) + (2/3 * 0,5) = 0,53$$

$(\# \# \# 2 \mid \# 5) \rightarrow 1 \quad s = 0,57 \quad (1/3 * 0,5) + (2/3 * 0,6) = 0,57$

Software - Algoritmo Genético

- ♦ Evolver - Palisade Corp (www.palisade.com)
 - Otimizador Genético Add-in para Excel
 - Emprega modelo tradicional de AG
 - Permite especificar:
 - Operadores Genéticos
 - Método de Solução
 - Parâmetros (população, taxas, condição de parada, etc)
 - Possui biblioteca de funções (C++ e VB).