

Universidade de Aveiro

Inteligência Artificial (LEI, LECI)

Tópicos de IA:
Representação do Conhecimento

Ano lectivo 2024/2025

Regente: Luís Seabra Lopes

Tópicos de Inteligência Artificial

- Agentes
 - Noção de agente
 - Objectivo da Inteligência Artificial
 - Agentes reactivos e deliberativos
 - Propriedades do mundo de um agente
 - Arquitecturas de agentes
- Representação do conhecimento
- Técnicas de resolução de problemas

Representação do conhecimento

- Redes semânticas
 - Redes semânticas genéricas
 - Sistemas de “frames”
 - Herança e raciocínio não-monotônico
 - Relação com diagramas UML
 - Exemplo para aulas práticas
- Lógica proposicional e lógica de primeira ordem
- Linguagem KIF
- Engenharia do conhecimento
- Ontologia geral
- Redes de Bayes

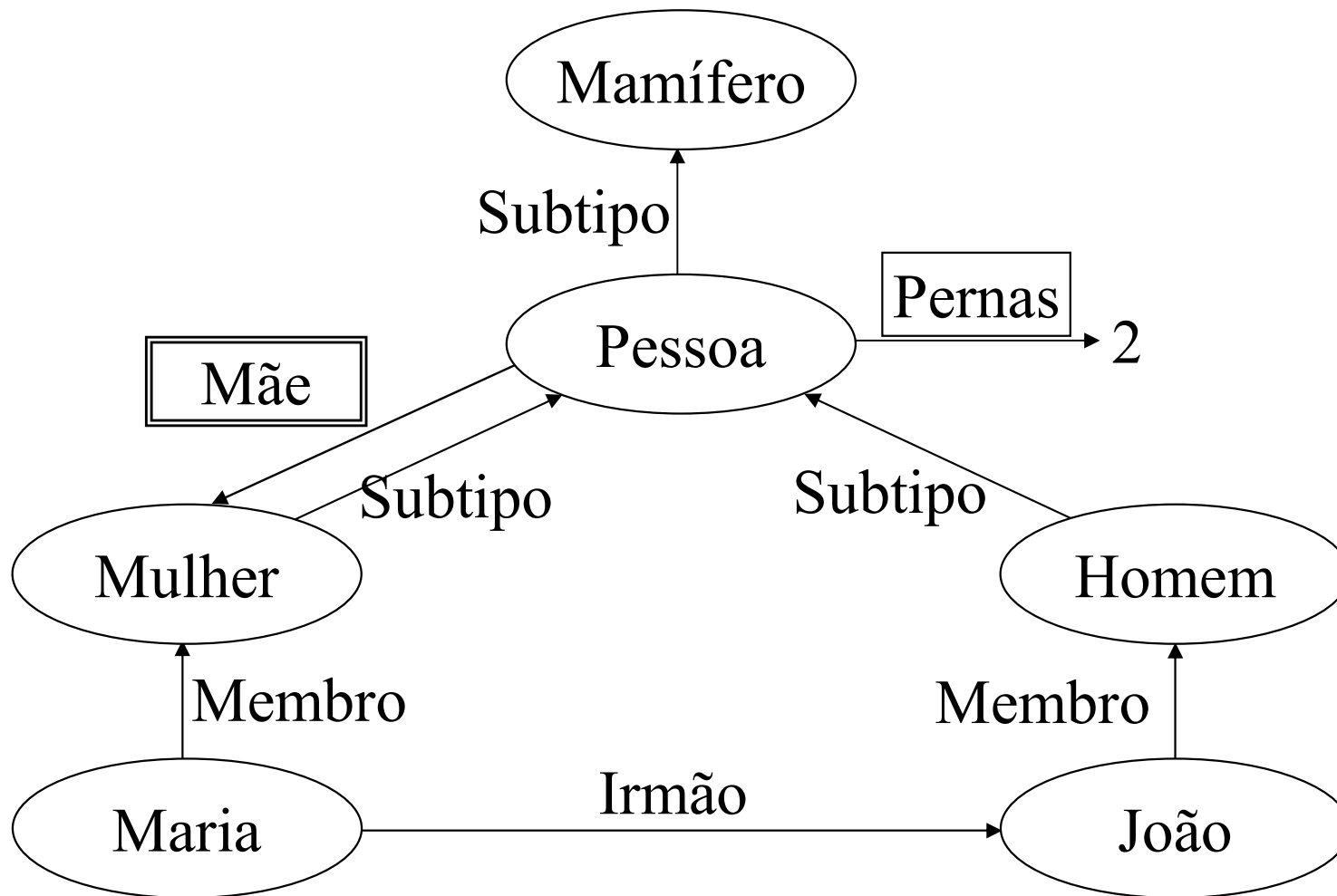
Redes Semânticas

- Redes semânticas são representações gráficas do conhecimento
- Têm a vantagem da legibilidade
- As redes semânticas podem ser tão expressivas quanto a lógica de primeira ordem

Redes Semânticas – tipos de relações

- *Sub-tipo* (ou *sub-conjunto* ou ainda *sub-classe*):
 - $A \subset B$
- *Membro* (ou *instância*):
 - $A \in B$
- Relação objecto-objecto:
 - $R(A,B)$
- Relação conjunto-objecto:
 - $\forall x \ x \in A \Rightarrow R(x,B)$
- Relação conjunto-conjunto:
 - $\forall x \ x \in A \Rightarrow \exists y \ (y \in B \wedge R(x,y))$

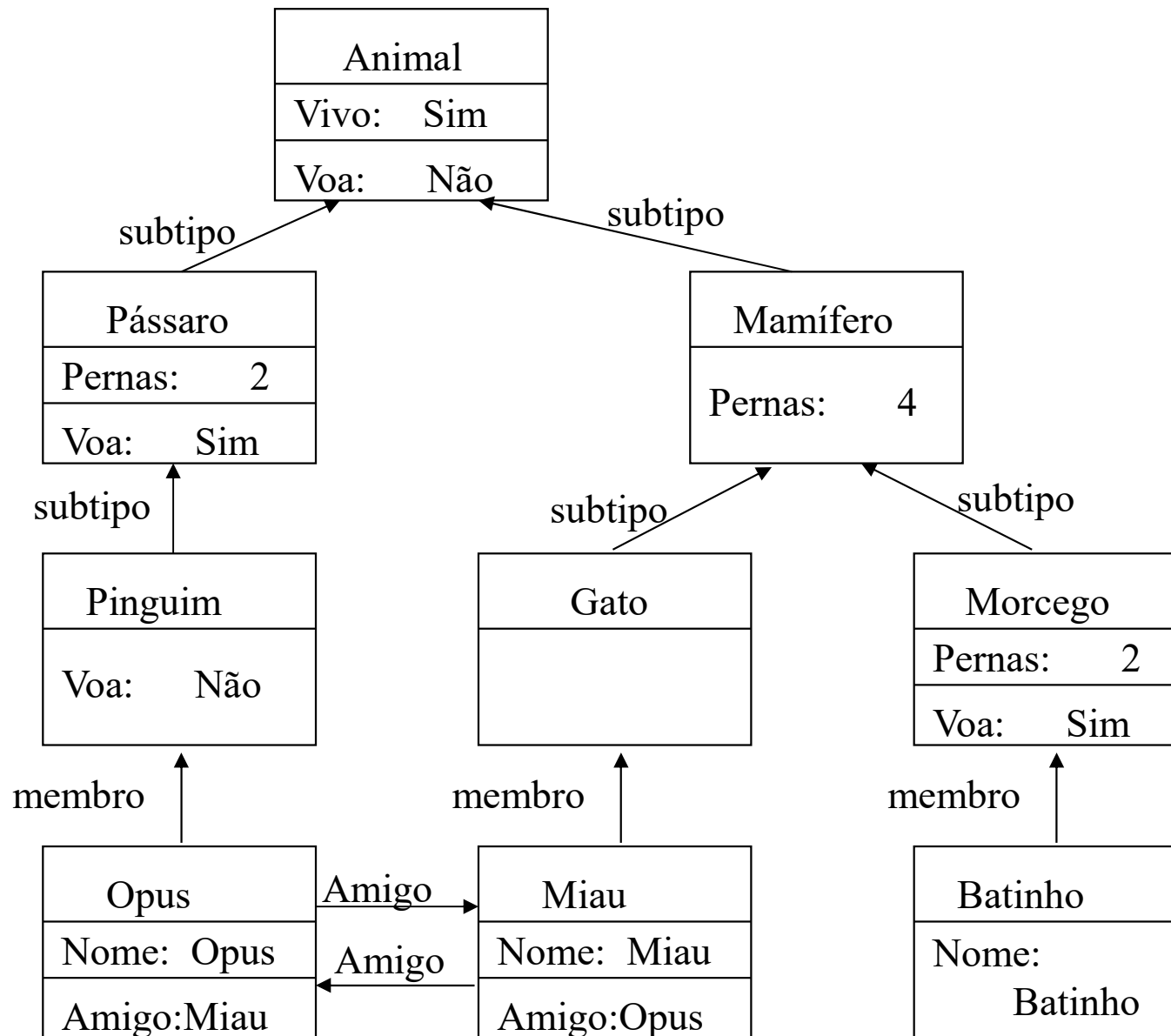
Redes semânticas – exemplo



Redes Semânticas - herança

- As relações de *sub-tipo* e *membro* permitem a herança de propriedades:
 - O sub-tipo herda todas as propriedades dos tipos mais abstractos dos quais descende
 - A instância herda todas as propriedades do tipo a que pertence
- A inferência pode ser vista como o seguimento das ligações entre entidades com vista à herança de propriedades
- Pode implementar-se raciocínio não monotónico através do estabelecimento de valores por defeito e o correspondente cancelamento da herança. (ver exemplo)

Redes Semânticas - exemplo



Redes Semânticas – Métodos e Demónios

- Normalmente, por razões computacionais, usam-se redes semânticas bastante menos expressivas do que a lógica de primeira ordem
- Deixa-se de lado:
 - Negação
 - Disjunção
 - Quantificação
- Em contra-partida, nomeadamente nos chamados sistemas de *frames*, usam-se métodos e demónios:
 - *Métodos* têm uma semântica similar à da programação orientada por objectos.
 - *Demónios* são procedimentos cuja execução é disparada automaticamente quando certas operações de leitura ou escrita são efectuadas.

GOLOG – Um gestor de objectos em Prolog

- O GOLOG é um gestor de objectos cuja semântica é próxima das *frames* (Seabra Lopes, 1995)
- Algumas primitivas:
 - new_frame(Frame)
 - new_slot(Slot)
 - new_value(Frame,Slot,Value)
 - new_relation(Rel,Trans,Restrictions,Inv)
 - Trans ::= transitive | intransitive
 - Restrictions ::= all | none | inclusion(Slots) | exclusion(Slots)
 - call_method(Frame,Method,ParamList)
 - new_demon(Frame,Slot,Proc,Access,When,Effect).
 - Access ::= if_read | if_write | if_delete | if_execute
 - When ::= before | after
 - Effect ::= alter_value | side_effect

UML / Diagramas de Classes - 1

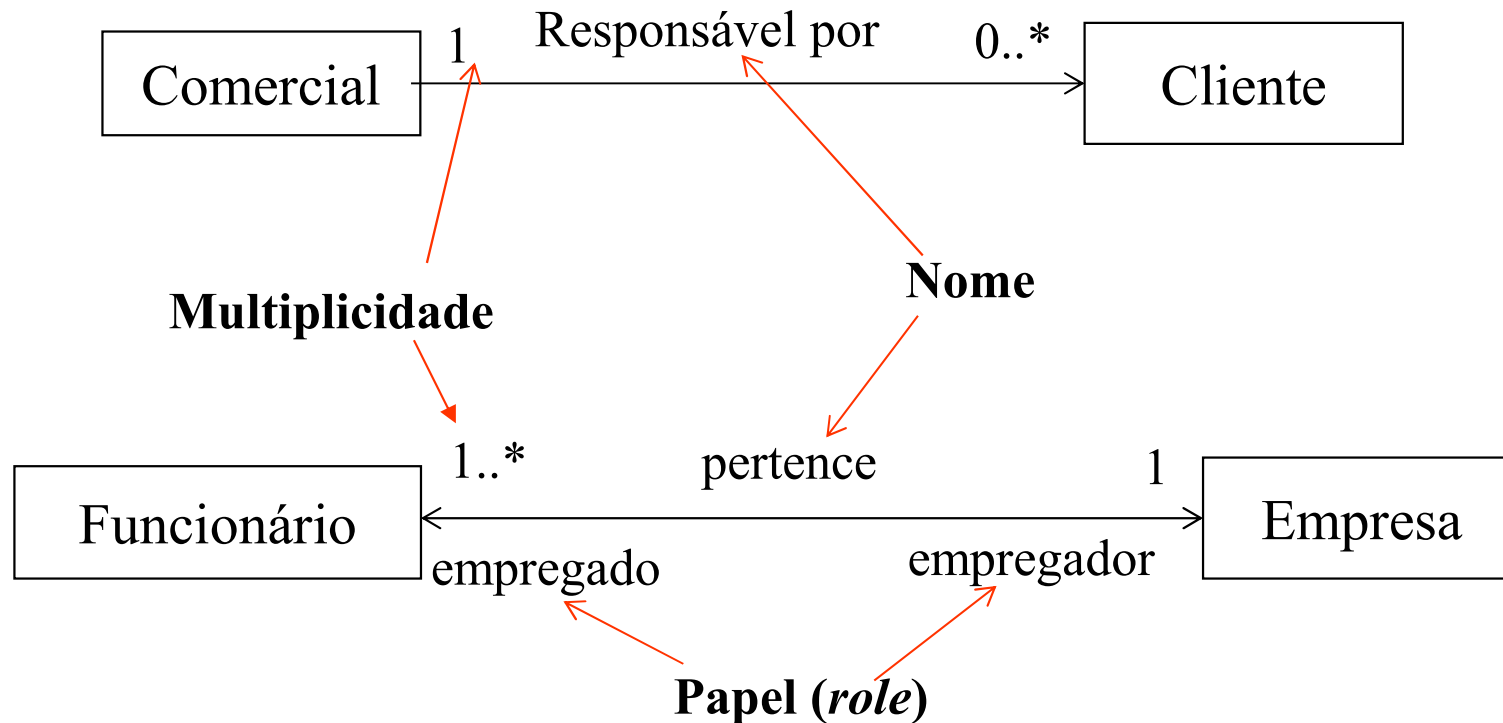
- Na linguagem gráfica UML (*Unified Modelling Language*), os diagramas de classes definem as relações existentes entre as diferentes classes de objectos num dado domínio
 - **Classe** – descrição de um conjunto de objectos que partilham os mesmos atributos, operações, relações e semântica; estes objectos podem ser:
 - Objectos físicos
 - Conceitos que não tenham uma existência palpável
 - **Atributo** – uma propriedade de uma classe
 - **Operação (ou método)** – é a implementação de um serviço que pode ser executado por qualquer objecto da classe
 - **Instância** de uma classe – um objecto que pertence a essa classe, ou seja, constitui uma concretização dessa classe
 - As instâncias de uma classe diferenciam-se pelos valores dos atributos
 - As instâncias “herdam” todos os atributos e métodos da sua classe

UML / Diagramas de Classes – 2:

Relações

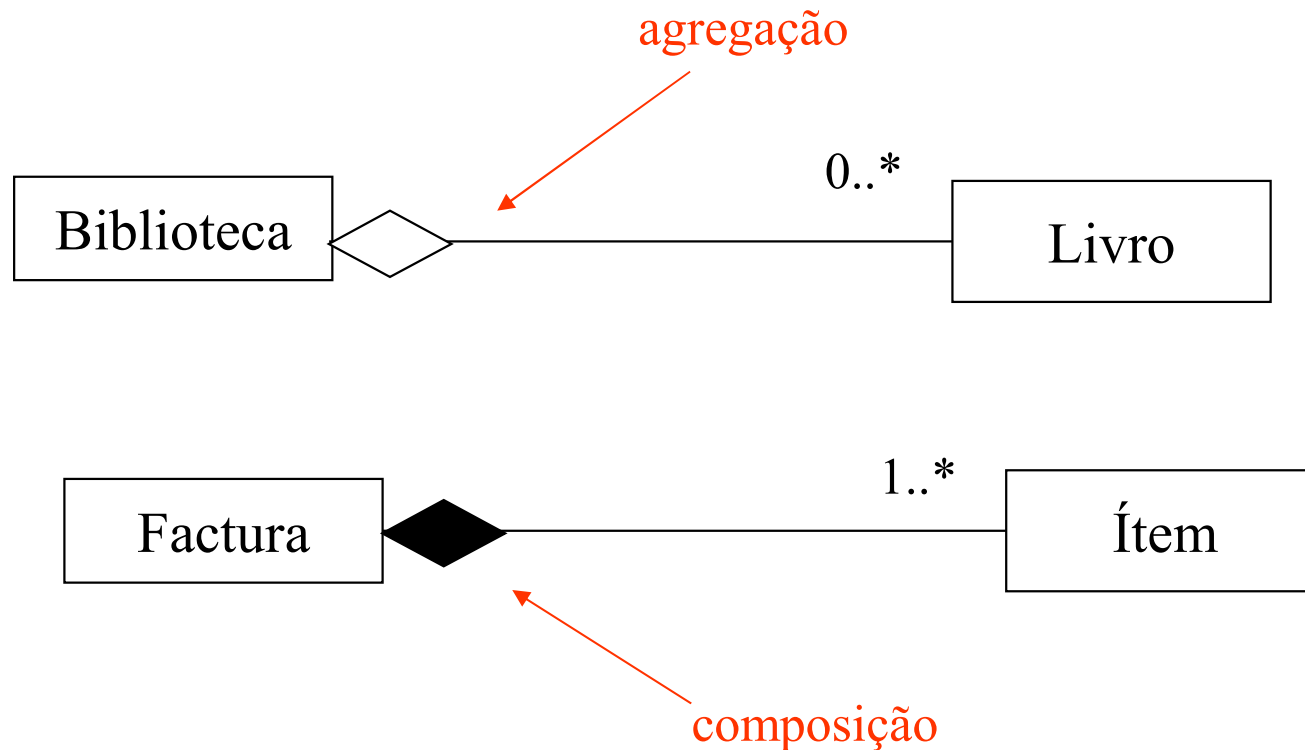
- Um aluno lê um livro
 - **Associação** : classe A “usa a”/ “interage com” classe B
- Um recibo tem vários itens
 - **Composição**: relação todo-parte
- Uma biblioteca tem vários livros
 - **Agregação**: representa relação estrutural entre instâncias de duas classes, em que a classe agregada existe independentemente da outra
- Um aluno é uma pessoa
 - **Generalização**: classe A generaliza classe B e B especializa A (super-classe/sub-classe)

UML / Diagramas de Classes – 3: Associação

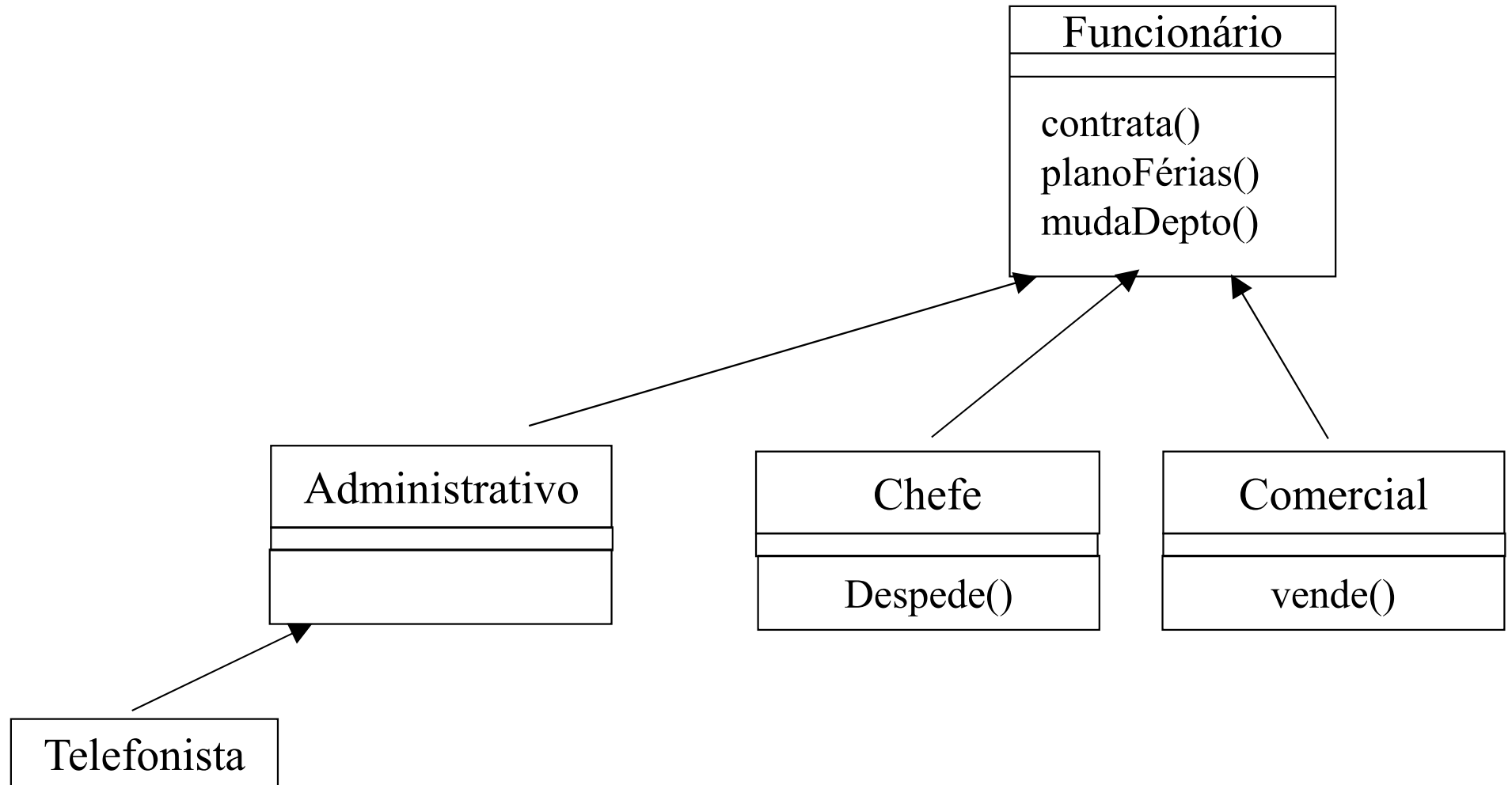


UML / Diagramas de Classes – 4:

Agregação e Composição



UML / Diagramas de Classes – 5: Generalização



Redes semânticas *versus* UML

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

Indução versus Dedução

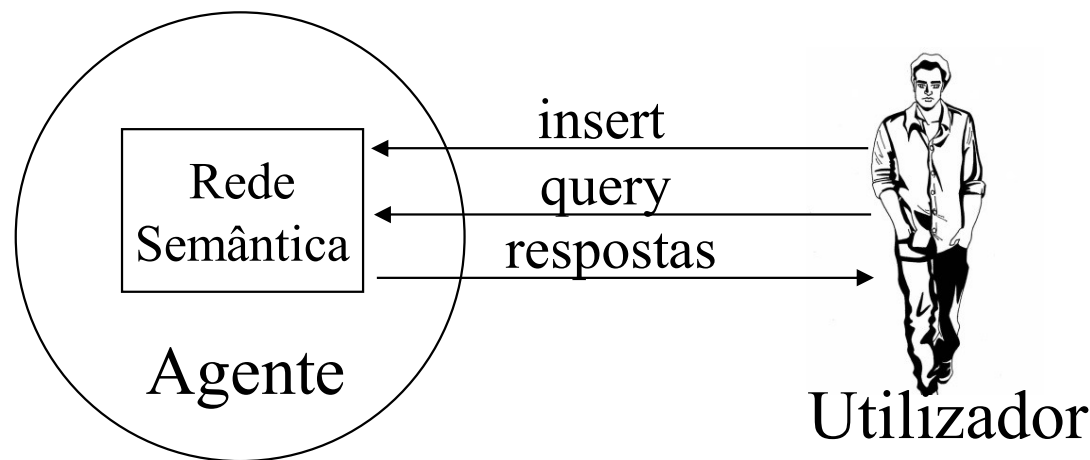
- Dedução – permite inferir casos particulares a partir de regras gerais
 - Preserva a verdade
 - As regras de inferência apresentadas anteriormente são regras dedutivas
- Indução – é o oposto da dedução; permite inferir regras gerais a partir de casos particulares
 - É a base principal da aprendizagem

Indução

- Exemplo:
 - Casos conhecidos
 - O gato Tareco gosta de leite
 - O gato Pirata gosta de leite
 - Regra inferida
 - Os gatos (normalmente) gostam de leite
 - Nas redes semânticas, a indução pode ser vista como uma “herança de baixo para cima”

Redes Semânticas em Python

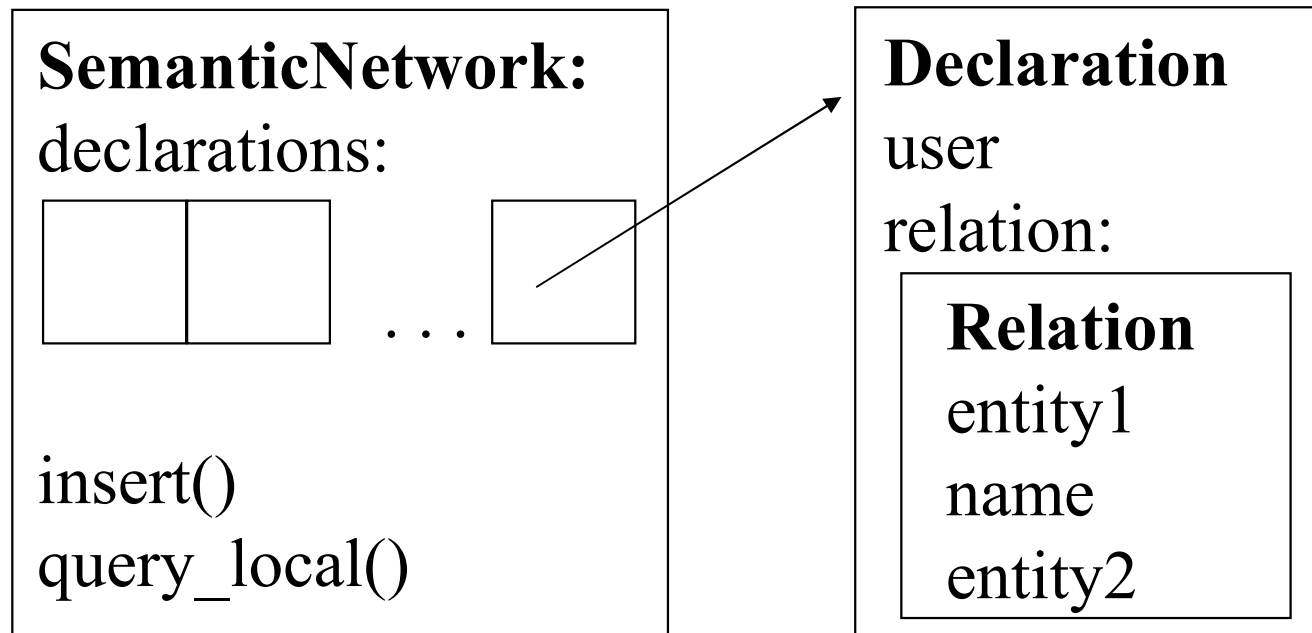
- Vamos criar uma rede semântica, definida como um conjunto de declarações
- Cada declaração associa uma relação semântica ao indivíduo que a declarou
 - Declaration(user,relation)



Redes Semânticas em Python

- Uma relação pode ser dos três tipos seguintes:
 - `Member(obj, type)` – um objecto é membro de um tipo
 - `Subtype(subtype, supertype)` – um tipo é subtipo de outro
 - `Association(entity1, name, entity2)` – uma entidade (objecto ou tipo) está associada a outra
- Operações principais:
 - `insert` – introduzir uma nova declaração
 - `query_local` – questionar a rede semântica sobre as declarações existentes
- Através da introdução incremental de declarações por diferentes interlocutores, emulamos de forma simplificada um processo de aprendizagem, em que o conhecimento é adquirido através da interacção com outros agentes

Redes Semânticas em Python



- Nota: ver módulo usado nas aulas práticas

Representação do conhecimento

- Redes semânticas
 - Redes semânticas genéricas
 - Sistemas de “frames”
 - Herança e raciocínio não-monotônico
 - Relação com diagramas UML
 - Implementação em Python
- Lógica proposicional e lógica de primeira ordem
- Linguagem KIF
- Engenharia do conhecimento
- Ontologia geral
- Redes de Bayes

Lógicas

- Uma lógica tem:
 - Síntaxe - descreve o conjunto de frases ou fórmulas que é possível escrever.
 - Nota: Estas são as fórmulas bem formadas ou WFF (do inglês *Well Formed Formula*)
 - Semântica - estabelece a relação entre as frases escritas nessa linguagem e os factos que representam.
 - Exemplo: a semântica da lógica proposicional é definida através de tabelas de verdade.
 - Regras de inferência - permitem manipular as frases, gerando umas a partir das outras; as regras de inferência são a base do processo de raciocínio.

Lógica Proposicional

- Baseada em proposições
 - *Proposição* = frase declarativa elementar que pode ser verdadeira ou falsa
 - Exemplos:
 - “A neve é branca”
 - “O açúcar é um hidrocarbono”
 - Variável proposicional = uma variável que toma o valor de verdade de uma dada proposição
- Uma fórmula em lógica proposicional é composta por uma ou mais variáveis proposicionais ligadas por conectivas lógicas
 - Uma frase proposicional elementar é um frase composta por uma única variável proposicional

Lógica de Primeira Ordem

- Componentes:
 - *Objectos* ou *entidades*
 - Exemplos: 1215, DDinis, Aveiro
 - *Expressões funcionais*
 - Exemplos: Potencia(4,3), Pai-de(Paulo)
 - Nota 1: Os objectos podem ser considerados como expressões funcionais cuja aridade é zero
 - Nota 2: A noção de *termo* engloba quer os objectos quer as expressões funcionais
 - *Predicados* ou *relações*
 - Exemplos: Pai(Rui, Paulo), Irmão(Paulo,Rosa)
 - Nota: Por definição, os argumentos de um predicado são termos.
- Aqui, as frases elementares são os predicados

Conectivas Lógicas

- Servem para combinar frases lógicas elementares por forma a obter frases mais complexas
- As conectivas lógicas mais comuns são as seguintes:
 - \wedge (conjunção)
 - \vee (disjunção)
 - \Rightarrow (implicação)
 - \neg (negação)

Variáveis, Quantificadores

- Na lógica de primeira ordem, os argumentos dos predicatos podem ser variáveis, usadas para representar termos não especificados
 - Exemplos: x , y , pos , $soma$, pai , ...
- Quantificação universal
 - $\forall x A \equiv$ ‘Qualquer que seja x , a fórmula A é verdadeira’
 - Se A é uma fórmula bem formada, então $\forall x A$ também é uma fórmula bem formada.
- Quantificação existencial
 - $\exists x A \equiv$ ‘Existe um x , para o qual a fórmula A é verdade’
 - Se A é uma fórmula bem formada, então $\exists x A$ também é uma fórmula bem formada.

Lógica de Primeira Ordem - Gramática

Fórmula \rightarrow *FórmulaAtômica*

| *Fórmula Conectiva* *Formula*

| *Quantificador Variável, ... Fórmula*

| \neg *Fórmula*

| $($ *Fórmula* $)$

FórmulaAtômica \rightarrow *Predicado* $($ *Termo* $,$... $)$ | *Termo* $=$ *Termo*

Termo \rightarrow *Função* $($ *Termo* $,$... $)$ | *Constante* | *Variável*

Conectiva \rightarrow \Rightarrow | \wedge | \vee | \Leftrightarrow

Quantificador \rightarrow \exists | \forall

Constante \rightarrow A | X1 | Paula | ...

Variável \rightarrow a | x | s | ...

Predicado \rightarrow Portista | Cor | ...

Função \rightarrow Registo | Mãe | ...

Exemplos

- “Todos em Oxford são espertos”:
 - $\forall x \text{ Estuda}(x, \text{Oxford}) \Rightarrow \text{Esperto}(x)$
 - Erro comum: Usar \wedge em vez de \Rightarrow
 $\forall x \text{ Estuda}(x, \text{Oxford}) \wedge \text{Esperto}(x)$
Significa “Todos estão em Oxford e todos são espertos”
- “Alguém em Oxford é esperto”:
 - $\exists x \text{ Estuda}(x, \text{Oxford}) \wedge \text{Esperto}(x)$
 - Erro comum: Usar \Rightarrow em vez de \wedge
 $\exists x \text{ Estuda}(x, \text{Oxford}) \Rightarrow \text{Esperto}(x)$
qualquer estudante de outra universidade forneceria uma interpretação verdadeira.
- “Existe uma pessoa que gosta de toda a gente”
 - $\exists x \forall y \text{ Gosta}(x, y)$

Interpretações em Lógica Proposicional

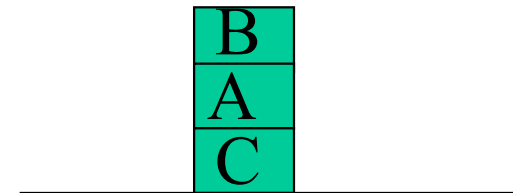
- Na lógica proposicional, uma interpretação de uma fórmula é uma atribuição de valores de verdade ou falsidade às várias proposições que nela ocorrem
 - Exemplo: a fórmula $A \wedge B$ tem quatro interpretações possíveis.
- Satisfatibilidade - Uma interpretação satisfaz uma fórmula se a fórmula toma o valor ‘verdadeiro’ para essa interpretação.
- Modelo de uma fórmula - uma interpretação que satisfaz essa fórmula.
- Tautologia - uma fórmula cujo valor é ‘verdadeiro’ em qualquer interpretação.

Interpretações em Lógica de Primeira Ordem

- Uma interpretação de uma fórmula em lógica de primeira ordem é o estabelecimento de uma correspondência entre as várias constantes que ocorrem na fórmula e os objectos do mundo, funções e relações que essas constantes representam.

– Exemplo:

- Objectos: A, B, C, Chão
- Funções: nenhuma
- Relações:
 - Em_cima_de: { $\langle B, A \rangle$, $\langle A, C \rangle$, $\langle C, \text{Chão} \rangle$ }
 - Livre: { $\langle B \rangle$ }
- Assumindo o estado dado pela figura, esta interpretação constitui um modelo



Lógica - Regras de Substituição - I

- São válidas quer na lógica proposicional quer na lógica de primeira ordem
- Leis de DeMorgan
$$\neg (A \wedge B) \equiv \neg A \vee \neg B$$
$$\neg (A \vee B) \equiv \neg A \wedge \neg B$$
- Dupla negação:
$$\neg \neg A \equiv A$$
- Definição da implicação:
$$A \Rightarrow B \equiv \neg A \vee B$$
- Transposição:
$$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$$

Lógica - Regras de Substituição - II

- Comutação

$$A \wedge B \equiv B \wedge A$$

$$A \vee B \equiv B \vee A$$

- Associação:

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$$

$$(A \vee B) \vee C \equiv A \vee (B \vee C)$$

- Distribuição:

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

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

Lógica - Regras de Substituição - III

- Leis de DeMorgan generalizadas (estas são específicas da lógica de primeira ordem):

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

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

Exercícios

- Representar as seguintes frases em lógica de primeira ordem:
 - Só um aluno chumbou a História
 - Nem todos os estudantes se inscreveram simultaneamente a *Introdução à Inteligência Artificial e Sistemas Inteligentes*
 - A melhor nota a História foi mais elevada do que a melhor nota a Biologia
 - Todos os Portistas gostam de Pinto da Costa
 - Existe um Sportinguista que gosta de todos os Benfiquistas que não são espertos
 - Existe um Barbeiro que barbeia toda a gente menos ele próprio

CNF e Forma Clausal

- Uma fórmula na *forma normal conjuntiva* (abreviado *CNF*, de *Conjunctive Normal Form*) é uma fórmula que consiste de uma conjunção de cláusulas.
- Uma *cláusula* é uma fórmula que consiste de uma disjunção de literais.
- Um *literal* é uma fórmula atômica (literal positivo) ou a negação de uma fórmula atômica (literal negativo).
 - Nota: na lógica proposicional uma fórmula atômica é uma proposição.
- *Forma clausal* é a representação de uma fórmula CNF através do conjunto das respectivas cláusulas

Conversão de uma Fórmula Proposicional para CNF e forma clausal

- Através dos seguintes passos:
 - Remover implicações
 - Reduzir o âmbito de aplicação das negações
 - Associar e distribuir até obter a forma CNF
- Exemplo:
 - Fórmula original: $A \Rightarrow (B \wedge C)$
 - Após remoção de implicações: $\neg A \vee (B \wedge C)$
 - Forma CNF: $(\neg A \vee B) \wedge (\neg A \vee C)$
 - Forma clausal: $\{ \neg A \vee B, \neg A \vee C \}$

Conversão para forma clausal em Lógica de Primeira Ordem - I

- Renomear variáveis, de forma a que cada quantificador tenha uma variável diferentes
- Remover as implicações
- Reduzir o âmbito das negações, ou seja, aplicar a negação
- Para estas transformações, aplicar as regras de substituição já apresentadas

Exemplo

Fórmula original:

$$\forall x \forall y \neg(p(x,y) \Rightarrow \forall y q(y,y))$$

Variáveis renomeadas:

$$\forall a \forall b \neg(p(a,b) \Rightarrow \forall c q(c,c))$$

Implicações removidas:

$$\forall a \forall b \neg(\neg p(a,b) \vee \forall c q(c,c))$$

Negações aplicadas:

$$\forall a \forall b (p(a,b) \wedge \exists c \neg q(c,c))$$

Conversão para forma clausal em

Lógica de Primeira Ordem - II

- Skolemização
 - Nome dado à eliminação dos quantificadores existenciais
 - Substituir todas as ocorrências de cada variável quantificada existencialmente por uma função cujos argumentos são as variáveis dos quantificadores universais exteriores
- Remover quantificadores universais

Exemplo (cont.)

Skolemizada aplicada:

$$\forall a \forall b (p(a,b) \wedge \neg q(f(a,b), f(a,b)))$$

Quantificadores removidos:

$$p(a,b) \wedge \neg q(f(a,b), f(a,b))$$

Conversão para forma clausal em Lógica de Primeira Ordem - III

- Converter para CNF
 - Usar as regras de substituição relativas à comutação, associação e distribuição
- Converter para a forma clausal, ou seja, eliminar conjunções
- Renomear variáveis de forma a que uma variável não apareça em mais do que uma fórmula

Exemplo (cont.)

Convertida para a forma clausal:
 $\{ p(a,b) , \neg q(f(a,b), f(a,b)) \}$

Variáveis renomeadas:
 $\{ p(a_1,b_1) , \neg q(f(a_2,b_2), f(a_2,b_2)) \}$

Lógica - Regras de Inferência

- Modus Ponens: $\{ A, A \Rightarrow B \} \vdash B$
- Modus Tolens: $\{ \neg B, A \Rightarrow B \} \vdash \neg A$
- Silogismo hipotético: $\{ A \Rightarrow B, B \Rightarrow C \} \vdash A \Rightarrow C$
- Conjunção: $\{ A, B \} \vdash A \wedge B$
- Eliminação da conjunção: $\{ A \wedge B \} \vdash A$
- Disjunção: $\{ A, B \} \vdash A \vee B$
- Silogismo disjuntivo (ou resolução unitária):
 $\{ A \vee B, \neg B \} \vdash A$
- Resolução: $\{ A \vee B, \neg B \vee C \} \vdash A \vee C$
- Dilema construtivo:
 $\{ (A \Rightarrow B) \wedge (C \Rightarrow D), A \vee C \} \vdash B \vee D$
- Dilema destrutivo:
 $\{ (A \Rightarrow B) \wedge (C \Rightarrow D), \neg B \vee \neg D \} \vdash \neg A \vee \neg C$

Lógica de Primeira Ordem

- Regras de Inferência específicas

- *Instanciação universal:*
 $\{ \forall x P(x) \} \vdash P(A)$
- *Generalização existencial*
 $\{ P(A) \} \vdash \exists x P(x)$

Consequências Lógicas, Provas

- Consequência lógica
 - Diz-se que A é consequência lógica do conjunto de fórmulas em Δ , e escreve-se
$$\Delta \models A,$$
se A toma o valor ‘verdadeiro’ em todas as interpretações para as quais cada uma das fórmulas em Δ toma também o valor verdadeiro.
- Definição de Prova
 - Uma sequência de fórmulas $\{ A_1, A_2, \dots, A_n \}$ é uma prova (ou dedução) de A_n a partir de um conjunto de fórmulas Δ sse cada uma das fórmulas A_i está em Δ ou pode ser inferida a partir das fórmulas $A_1 \dots A_{i-1}$.
 - Neste caso escreve-se: $\Delta \vdash A_n$

Correcção, Completude

- Correcção - Diz-se que um conjunto de regras de inferência é correcto se todas as fórmulas que gera são consequências lógicas
- Completude - Diz-se que um conjunto de regras de inferência é completo se permite gerar todas as consequências lógicas.
- Um sistema de inferência correcto e completo permite tirar consequências lógicas sem ter que analisar caso a caso as várias interpretações.

Metateoremas

- Teorema da dedução:
 - Se $\{ A_1, A_2, \dots, A_n \} \models B$, então $A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow B$, e vice-versa.
- Redução ao absurdo:
 - Se o conjunto de fórmulas Δ é satisfatível (logo tem pelo menos um modelo) e $\Delta \cup \{\neg A\}$ não é satisfatível, então $\Delta \models A$.

Resolução não é Completa

- A resolução é uma regra de inferência correcta (gera fórmulas necessariamente verdadeiras)

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

- A resolução não é completa.
 - Exemplo - A resolução não consegue derivar a seguinte consequência lógica:

$$\{ A \wedge B \} \models A \vee B$$

Refutação por Resolução

- A refutação por resolução é um mecanismo de inferência completo
 - Neste caso, usa-se a resolução para provar que a negação da consequência lógica é inconsistente com a premissa (*meta-teorema da redução ao absurdo*).
 - No exemplo dado, prova-se que
$$(A \wedge B) \wedge \neg(A \vee B)$$
é inconsistente (basta mostrar que é possível derivar a fórmula ‘Falso’).
- Passos da refutação por resolução:
 - Converter a premissa e a negação da consequência lógica para um conjunto de cláusulas.
 - Aplicar a resolução até obter a cláusula vazia.

Substituições, Unificação

- A aplicação da *substituição* $s = \{ t_1/x_1, \dots, t_n/x_n \}$ a uma fórmula W denota-se $SUBST(W,s)$ ou Ws ; Significa que todas as ocorrências das variáveis x_1, \dots, x_n em W são substituídas pelos termos t_1, \dots, t_n
- Duas fórmulas A e B são unificáveis se existe uma substituição s tal que $As = Bs$. Nesse caso, diz-se que s é uma *substituição unificadora*.
- A *substituição unificadora mais geral* (ou *minimal*) é a mais simples (menos extensa) que permite a unificação.

Resolução e Refutação na Lógica de Primeira Ordem

- Resolução:
 $\{ A \vee B, \neg C \vee D \} \vdash \text{SUBST}(A \vee D, g)$
em que B e C são unificáveis sendo g a sua substituição unificadora mais geral
- A regra da resolução é correcta
- A regra da resolução não é completa
- Tal como na lógica proposicional, também aqui a refutação por resolução é completa

Resolução com Cláusulas de Horn

- O mecanismo de prova baseado na refutação por resolução é completo e correcto mas não é eficiente (na verdade é NP-completo)
- Uma cláusula de Horn é uma cláusula que tem no máximo um literal positivo
 - Exemplos:
$$\begin{array}{ccc} A & & \neg A \vee B \\ \neg A \vee B \vee \neg C & & \neg A \vee \neg B \end{array}$$
- Existem algoritmos de dedução baseados em cláusulas de Horn cuja complexidade temporal é linear
 - As linguagens Prolog e Mercury baseiam-se em cláusulas de Horn

Representação do conhecimento

- Redes semânticas
 - Redes semânticas genéricas
 - Sistemas de “frames”
 - Herança e raciocínio não-monotônico
 - Relação com diagramas UML
 - Implementação em Python
- Lógica proposicional e lógica de primeira ordem
- Linguagem KIF
- Engenharia do conhecimento
- Ontologia geral
- Redes de Bayes

KIF (= Knowledge Interchange Format)

- Esta é uma linguagem desenhada para representar o conhecimento trocado entre agentes.
 - A motivação para a criação do KIF é similar à que deu origem a outros formatos de representação, como o PostScript.
- Pode ser usada também para representar os modelos internos de cada agente.
- Características principais:
 - Semântica puramente declarativa (o Prolog é também uma linguagem declarativa, mas a semântica depende em parte do modelo de inferência)
 - Pode ser tão ou mais expressiva quanto a lógica de primeira ordem.
 - Permite a representação de meta-conhecimento (ou seja, conhecimento sobre o conhecimento)

KIF – características gerais

- O mundo é conceptualizado em termos de objectos e relações entre objectos
- Uma relação é um conjunto arbitrário de listas de objectos.
 - Exemplo: a relação $<$ é o conjunto de todos os pares (x,y) em que $x < y$.
- O universo de discurso é o conjunto de todos os objectos cuja existência é conhecida, presumida ou suposta.
 - Os objectos podem ser *concretos* ou *abstratos*
 - Os objectos podem ser *primitivos* (não decomponíveis) ou *compostos*

KIF - Componentes da linguagem

- Caracteres
- Lexemas
 - Lexemas especiais (aqueles que têm um papel pré-definido na própria linguagem)
 - Palavras
 - Códigos de caracteres
 - Blocos de códigos de caracteres
 - Cadeias de caracteres
- Expressões
 - Termos - objectos primitivos ou compostos
 - Frases - expressões com valor lógico
 - Definições - frases verdadeiras por definição

KIF - termos

- Constante
- Variável individual
- Expressão funcional
 - $(functor\ arg1\ ..\ argn)$
 - $(functor\ arg1\ ..\ argn\ seqvar)$
- Lista
 - $(listof\ t1\ \dots\ tn)$
- Termo lógico
 - $(if\ c1\ t1\ ..\ cn\ tn\ default)$
- Código de caracter, bloco de códigos de caracteres e cadeia de caracteres
- Citação (quotation)
 - $(quote\ lista)$ ou $'lista$

KIF - frases

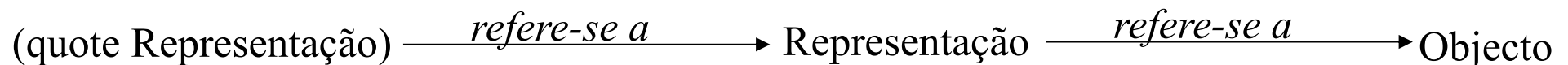
- Constante: true, false
- Equação
(= *termo1 termo2*)
- Inequação
(/= *termo1 termo2*)
- Frase relacional
(*relação t1 .. tn*)
- Frase lógica: construída com as conectivas lógicas ('not', 'and', 'or', '=>', '<= ', '<=>')
- Frase quantificada
(forall *var1 ... varn frase*)
(exists *var1 ... varn frase*)

KIF - definições

- Definição de objectos
 - Igualdade: (defobject $s := t$)
Exemplo: (defobject nil := (listof))
 - Conjunção: (defobject $s p1 .. pn$)
 - etc.
- Definição de funções
 - (deffunction $f(v1 .. vn) := t$)
 - Exemplo:
 - (deffunction head (?l) := (if (= (listof ?x @items) ?l) ?x))
- Definição de relações (=predicados)
 - (defrelation $r(v1 .. vn) := p$)
 - etc.
 - Exemplos:
 - (defrelation null (?l) := (= ?l (listof)))
 - (defrelation list (?x) :=
(exists (@l) (= ?x (listof (@l)))))

KIF - meta-conhecimento

- Pode formalizar-se conhecimento sobre o conhecimento
- O mecanismo da citação (quotation) permite tratar expressões como objectos
- Por exemplo a ocorrência da palavra `joão` numa expressão designará uma pessoa; entretanto a expressão `(quote joão)` ou `'joão` designa a própria palavra `joão` e não o objecto ou pessoa a que ela se refere.
- Outros exemplos:
 `(acredita joão '(material lua queijo))`
 `(=> (acredita joão ?p) (acredita ana ?p))`
- Graficamente, podemos ilustrar da forma seguinte:



KIF - dimensões de conformação

- KIF é uma linguagem altamente expressiva
- No entanto, KIF tende a sobrecarregar os sistemas de geração e de inferência
- Por isso, foram definidas várias dimensões de conformação
- Um perfil de conformação é uma selecção de níveis de conformação para cada uma das dimensões referidas

KIF - perfis de conformação

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

Representação do conhecimento

- Redes semânticas
 - Redes semânticas genéricas
 - Sistemas de “frames”
 - Herança e raciocínio não-monotônico
 - Relação com diagramas UML
 - Implementação em Python
- Lógica proposicional e lógica de primeira ordem
- Linguagem KIF
- Engenharia do conhecimento
- Ontologia geral
- Redes de Bayes

Engenharia do Conhecimento

- Uma *base de conhecimento* (BC) é um conjunto de representações de *factos* e *regras* de funcionamento do mundo; factos e regras recebem a designação genérica de *frases*.
- Engenharia do conhecimento é o processo ou actividade de construir bases de conhecimento. Isto envolve:
 - Estudar o domínio de aplicação – frequentemente através de entrevistas com peritos (processo de *aquisição de conhecimento*)
 - Determinar os objectos, conceitos e relações que será necessário representar
 - Escolher um vocabulário para entidades, funções e relações (por vezes chamado *ontologia*)
 - Codificar conhecimento genérico sobre o domínio (um conjunto de *axiomas*)
 - Codificar descrições para problemas concretos, interrogar o sistema e obter respostas.
 - Por vezes o domínio é tão complexo que não é praticável codificar à mão todo o conhecimento necessário. Neste caso usa-se *aprendizagem automática*.

Identificação de objectos, conceitos e relações - 1

- Na modelação em análise de sistemas e engenharia de software coloca-se o mesmo problema
 - Assim, para um problema complexo de representação do conhecimento, não é descabido seguir uma metodologia de análise em boa parte similar às que se usam nos sistemas de informação
- Algumas das palavras que usamos para descrever um domínio em linguagem natural dão naturalmente origem a nomes de objectos, conceitos e relações
 - Substantivos comuns → *conceitos* (também chamados *classes* ou *tipos*)
 - Substantivos próprios → *objectos* (também chamados *instâncias*)
 - Verbo “ser” → pode indicar uma relação de *instanciação* (entre objecto e tipo) ou de *generalização* (entre subtipo e tipo)
 - Verbos “ter” e “conter” → podem indicar uma relação de composição
 - Outros verbos → podem sugerir outras relações relevantes

Identificação de objectos, conceitos e relações - 2

- Convem avaliar a importância para o problema das palavras utilizadas bem como dos objectos, conceitos e relações subjacentes
 - Não considerar substantivos que identifiquem objectos, conceitos ou relações irrelevantes para o problema
 - Quando vários substantivos aparecem a referir-se ao mesmo conceito, escolher o mais representativo ou adequado
- Um conceito mais abstracto pode ser criado atribuindo-lhe o que é comum a outros dois ou mais conceitos previamente identificados

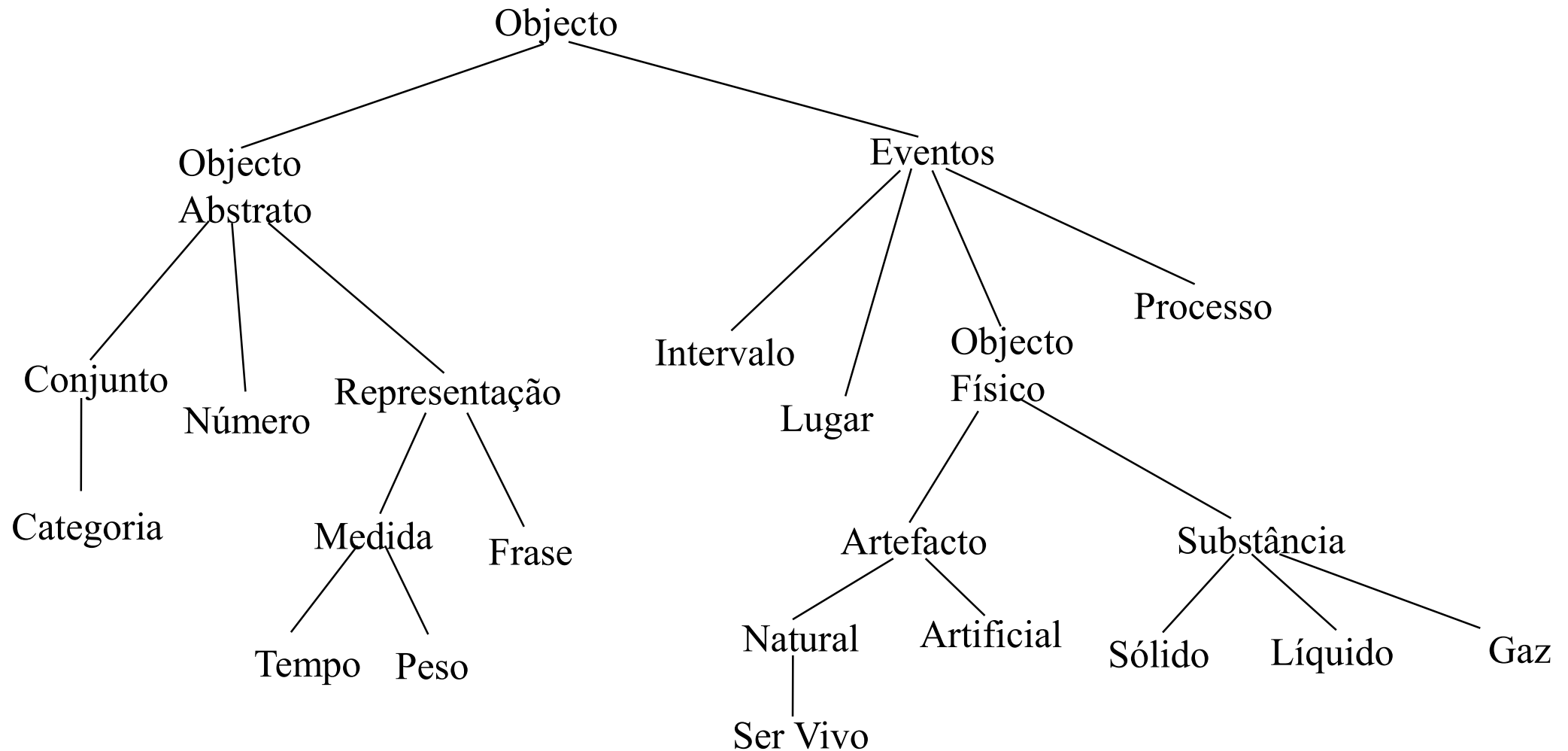
Ontologias

- Uma ontologia é um vocabulário sobre um domínio conjugado com relações hierárquicas como *membro* e *subtipo* e eventualmente outras.
- O objectivo de uma ontologia é captar a essência da organização do conhecimento num domínio.

Ontologia Geral

- Uma ontologia geral, aplicável a uma grande variedade de domínios de aplicação, envolve as seguintes noções:
 - Categorias, tipos ou classes
 - Medidas numéricas
 - Objectos compostos
 - Tempo, espaço e mudanças
 - Eventos e processos (eventos contínuos)
 - Objectos físicos
 - Substâncias
 - Objectos abstractos e crenças

Uma possível ontologia geral



Representação do conhecimento

- Redes semânticas
 - Redes semânticas genéricas
 - Sistemas de “frames”
 - Herança e raciocínio não-monotônico
 - Relação com diagramas UML
 - Implementação em Python
- Lógica proposicional e lógica de primeira ordem
- Linguagem KIF
- Engenharia do conhecimento
- Ontologia geral
- Redes de Bayes

Redes de crença bayesianas

- Também conhecidas simplesmente como “redes de Bayes”
- Permitem representar conhecimento impreciso em termos de um conjunto de variáveis aleatórias e respectivas dependências
 - As dependências são expressas através de probabilidades condicionadas
 - A rede é um grafo dirigido acíclico

Axiomas das probabilidades

- Para uma qualquer proposição a , a sua probabilidade é um valor entre 0 e 1:

$$0 \leq P(a) \leq 1$$

- Proposições necessariamente verdadeiras têm probabilidade 1

$$P(\text{true}) = 1$$

- Proposições necessariamente falsas têm probabilidade 0

$$P(\text{false}) = 0$$

- A probabilidade da disjunção é a soma das probabilidades subtraída da probabilidade da intercepção:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

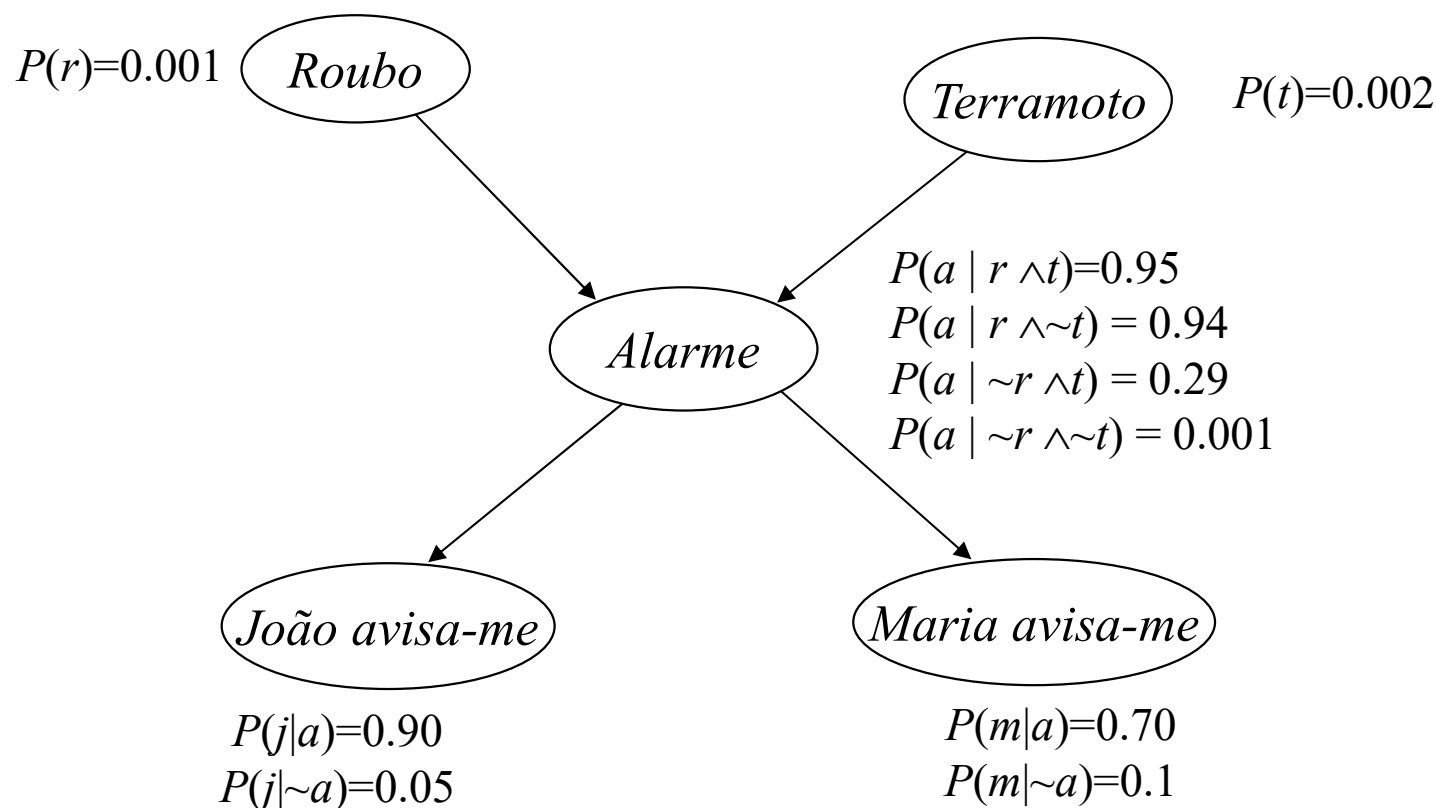
Probabilidades condicionadas

- Uma probabilidade condicionada $P(a|b)$ identifica a probabilidade de ser verdadeira a proposição a na condição de (isto é, sabendo nós que) a proposição b é verdadeira
- Pode calcular-se da seguinte forma:

$$P(a | b) = \frac{P(a \wedge b)}{P(b)}$$

Redes de crença bayesianas – exemplo

- Por simplicidade, focamos em variáveis aleatórias booleanas:



Redes de crença bayesianas – probabilidade conjunta

- A probabilidade conjunta identifica a probabilidade de ocorrer uma dada combinação de valores de todas as variáveis da rede:

$$P(x_1 \wedge \dots \wedge x_n) = \prod_{i=1}^n P(x_i \mid \text{pais}(x_i))$$

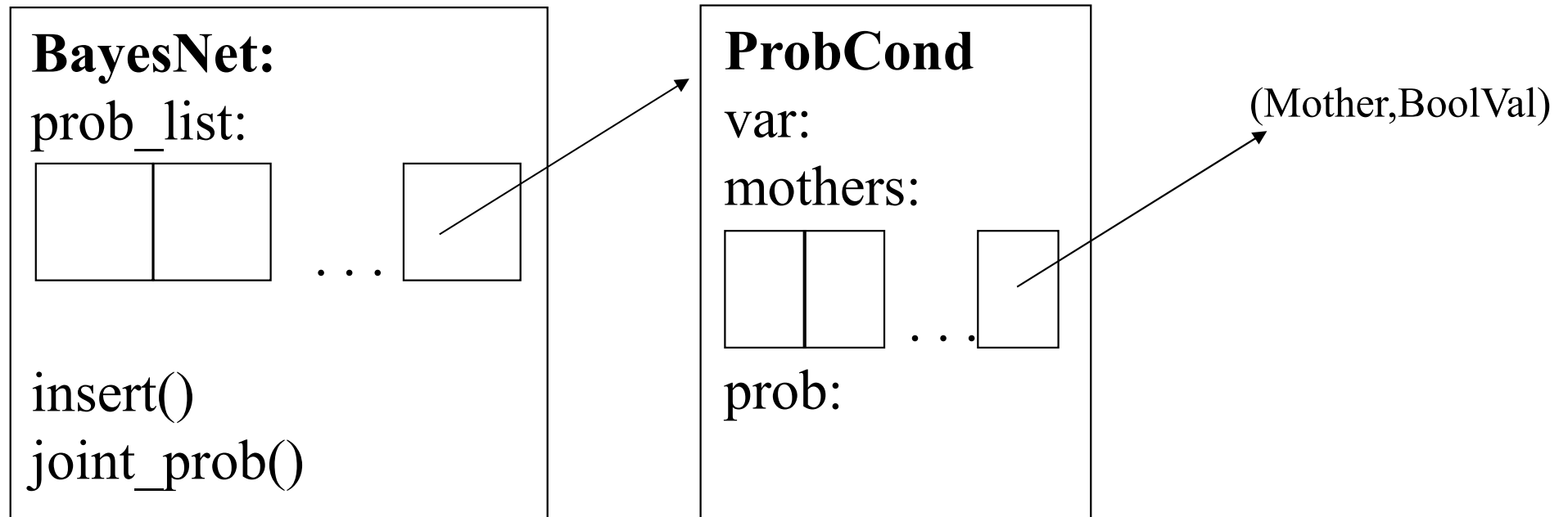
- Assim, no exemplo anterior, a probabilidade de o alarme tocar e o João e a Maria ambos avisarem num cenário em que não há roubo nem terremoto, é dada por:

$$\begin{aligned} & P(j \wedge m \wedge a \wedge \sim t \wedge \sim r) \\ &= P(j \mid a) \times P(m \mid a) \times P(a \mid \sim r \wedge \sim t) \times P(\sim r) \times P(\sim t) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 \\ &= 0.000628 \end{aligned}$$

Redes Bayesianas em Python

- Vamos criar uma rede de crença bayesiana, representada com base numa lista de probabilidades condicionadas
 - Classe `BayesNet()`
- A probabilidade condicionada de uma dada variável ser verdadeira, dados os valores (`True` ou `False`) das variáveis mães, é representado pela seguinte classe:
 - Classe `ProbCond(var,mother_vals,prob)`
 - Exemplo: `ProbCond("a", [("r",True), ("t",True)], 0.95)`
- Operações principais:
 - `insert` – introduzir uma nova probabilidade condicionada na rede
 - `joint_prob` – obter a probabilidade conjunta para uma dada conjunção de valores de todas as variáveis da rede

Redes de crença em Python



- Nota: ver módulo usado nas aulas práticas

Redes de crença bayesianas – probabilidade individual

- A probabilidade individual é a probabilidade de um valor específico (*verdadeiro* ou *falso*) de uma variável
- Calcula-se somando as probabilidades conjuntas das situações em que essa variável tem esse valor específico
- O cálculo das probabilidades conjuntas pode restringir-se à variável considerada e às outras variáveis das quais depende (ascendentes na rede bayesiana)
 - Exemplo: o conjunto dos ascendentes de “João avisa” é { “alarme”, “roubo” e “terramoto” }

Redes de crença bayesianas – probabilidade individual

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

- Seja:
 - $C = \{x_1, \dots, x_n\}$ – conjunto de variáveis da rede
 - $x_i \in C$ – uma qualquer variável da rede
 - $v_i \in \{v, f\}$ – valor de x_i cuja probabilidade se pretende calcular
 - $\{a_1, \dots, a_k\} \subset C$ – conjunto das variáveis da rede que são ascendentes de x_i