Lógica de Primeira Ordem

Capítulo 9

Inferência proposicional

- Prova semântica: através da enumeração de interpretações e verificação de modelos
- Prova sintática: uso de regras de inferência

Inferência Proposicional versus Inferência de primeira ordem

- Inferência de primeira ordem pode ser realizada convertendo-se a base de conhecimento para lógica proposicional e utilizando-se a inferência proposicional.

Regras de inferência para quantificadores

- Técnica de *proposicionalização*
  - Instanciação universal
  - Instanciação do existencial

Instanciação universal

$\forall x\ Rei(x) \wedge Guloso(x) \Rightarrow \text{Perverso}(x)$

Podemos deduzir:

$Rei(João) \wedge Guloso(\text{João}) \Rightarrow \text{Perverso}(João)$

$Rei(Pedro) \wedge Guloso(Pedro) \Rightarrow \text{Perverso}(Pedro)$

$Rei((pai(João)) \wedge Guloso(pai(João)) \Rightarrow$
$\text{Perverso}(pai(João))$

…

$$\frac{\forall \nu \alpha}{\text{SUBST}(\{\nu/g\},\alpha)}$$

Instanciação do existencial

$\exists x\ Coroa(x) \wedge NaCabeça(x,João)$

Podemos deduzir:

$Coroa(C1) \wedge NaCabeça(C1,João)$, sendo C1 um símbolo constante novo na KB

Símbolo de constante *k*
→ *constante de Skolem*

$$\frac{\exists \nu \alpha}{SUBST(\{\nu/k\},\alpha)}$$

Proposionalização

- **Problema**: quando a KB incluir um símbolo de função, o conjunto de substituições de termos básicos é infinito!
  - Ex.: pai(pai(pai ... (pai(pai(João))) ...))
- Teorema de Herbrand:
  - *se uma sentença é conseqüência lógica da KB original, então existe uma prova envolvendo apenas um sub-conjunto finito da KB proposicionalizada*.

Completude

- A técnica de proposicionalização é completa:
  - Qualquer sentença que é consequência lógica da KB em LPO pode ser provada na KB em LP (KB convertida)

Decidibilidade da LPO

- Não é possível saber se uma sentença é consequência lógica até que a prova termine

*A questão de consequência lógica no caso da lógica de primeira ordem é **semidecidível** – existem algoritmos que respondem "sim" para toda sentença que é consequência lógica mas não existe nenhum algoritmo que também responda "não" para toda sentença que não é consequência lógica.*

Exemplo

$\forall x\ Rei(x) \wedge Guloso(x) \Rightarrow \text{Perverso}(x)$

$Rei(João)$

$Guloso(João)$

$Irmão(Ricardo,João)$

Aplicando a Instanciação Universal à primeira sentença:

$Rei(\text{João}) \wedge Guloso(\text{João}) \Rightarrow Perverso(João)$

$Rei(\text{Ricardo}) \wedge Guloso(\text{Ricardo}) \Rightarrow Perverso(Ricardo)$

podemos inferir:

$$Perverso(João)$$

Exemplo

Note que a sentença:

$\forall\ x\ \mathit{Rei}(\text{Ricardo}) \wedge \mathit{Guloso}(\text{Ricardo}) \Rightarrow \mathit{Perverso}(\mathit{Ricardo})$

não foi usada na prova

Podemos saber quais sentenças são relevantes para provarmos uma sentença α?

Regra de inferência de primeira ordem

Para provar ***Perverso(João)*** usando a implicação

$$\forall\, x\ Rei(\mathrm{x}) \wedge Guloso(\mathrm{x}) \Rightarrow Perverso(\mathrm{x})$$

precisamos saber se existe uma substituição θ que torne a premissa da implicação idênticas a sentenças que já estão na KB → θ={x/João}

Regra de inferência de primeira ordem

Para provar ***Perverso(João)*** usando a implicação

$$\forall\, x\ Rei(\mathrm{x}) \wedge Guloso(\mathrm{x}) \Rightarrow Perverso(\mathrm{x})$$

e

$$\forall\, y\ Guloso(\mathrm{y})$$

precisamos saber se existe uma substituição θ que torne a premissa da implicação idênticas a sentenças que já estão na KB → θ={x/João, y/João}

Modus Ponens generalizado

Dada as sentenças atômicas p_i , p_i' e q , para as quais exista uma substituição θ tal que $SUBST(\theta, p_i') = SUBST(\theta, p_i)$, para todo i,

$$\frac{p_1', p_2', \ldots, p_n', (p_i \wedge p_i \wedge \ldots \wedge p_n \rightarrow q)}{SUBST(\theta,q)}$$

Modus Ponens generalizado

- Versão **elevada** (*lifted*) do Modus Ponens
  - Essa regra eleva o Modus Ponens da lógica proposicional à lógica de primeira ordem

- Vantagens das regras de inferências elevadas: *só efetuam as substituições necessárias para permitir a derivação de inferências específicas*

Unificação

Unificar(p,q) = θ onde

SUBST(θ,p) = SUBST (θ,q)

Unificar(Conhece(João,x), Conhece(y, Mãe(y))) =

{y/João,x/Mãe(João)}

Unificar(Conhece(João,x), Conhece(x, Elizabeth) =

falha

Problema 1: as duas expressões utilizam o mesmo nome de variável

Unificação

Solução: *padronização separada* (renomear as variáveis)

Unificar(Conhece(João,x), Conhece(z23, Elizabeth) =
{x/Elizabeth, z23/João}

Unificação

Problema 2: podem existir várias maneiras de unificar sentenças:

Para todo par de expressões que pode ser unificada, existe um único **unificador mais geral** (UMG) que é exclusivo para renomear variáveis

Algoritmo de unificação (I)

Função UNIFICAR(x,y,θ) **devolve** uma substituição que
torna x e y idênticas ou devolve *falha*

entrada: x, y, uma variável, const, lista ou composto
θ, a substituição construida até agora

se θ=falha **então devolve** *falha*

se x=y **então devolve** θ

se VARIAVEL?(x) **então devolve** UNIFICAR-VAR(x,y, θ)

se VARIAVEL?(y) **então devolve** UNIFICAR-VAR(y,x, θ)

se COMPOSTO?(x) **e** COMPOSTO?(y) **e** FUNC[x] = FUNC[y] **então devolve**
UNIFICAR(ARGS[x], ARGS[y], θ))

se LISTA?(x) **e** LISTA?(y) **então devolve** UNIFICAR(RESTO[x], RESTO[y],
UNIFICAR(PRIMEIRO[x], PRIMEIRO[y], θ))

devolve *falha*

Algoritmo de unificação (I)

Função UNIFICAR(x,y,θ) **devolve** uma substituição que torna x e y idênticas ou devolve *falha*

entrada: x, y, uma variável, const, lista ou composto
θ, a substituição construida até agora

se θ=*falha* **então devolve** *falha*

se x=y **então devolve** θ

se VARIAVEL?(x) **então devolve** UNIFICAR-VAR(x,y, θ)

se VARIAVEL?(y) **então devolve** UNIFICAR-VAR(y,x, θ)

se COMPOSTO?(x) e COMPOSTO?(y) **então devolve** UNIFICAR(ARGS[x], ARGS[y], UNIFICAR(FUNC[x], FUNC[y], θ)) (como a chamada interna é executada antes da mais externa, se as funtores forem diferentes os argumentos não podem ser unificados)

se LISTA?(x) e LISTA?(y) **então devolve** UNIFICAR(RESTO[x], RESTO[y], UNIFICAR(PRIMEIRO[x], PRIMEIRO[y], θ))

devolve *falha*

Algoritmo de unificação (II)

Função UNIFICAR-VAR(var,x,θ) **devolve** uma substituição θ

entrada: var, uma variável

x, qualquer expressão

θ, a substituição construida até agora

se {var/val}∈ θ **então devolve** UNIFICAR(val,x, θ)

se {x/val}∈ θ **então devolve** UNIFICAR(var,val, θ)

se VERIFICAR-OCORRENCIA?(var,x) **então devolve** *falha*

senão devolver *adicionar* {var/x} a θ

VERIFICAR-OCORRENCIA, tem complexidade quadrática no tamanho das expressões que estão sendo unificadas

Armazenamento e recuperação

- Armazenar e recuperar podem ser definidas como funções mais primitivas que o TELL e ASK
- RECUPERAR: devolve todos os unificadores tais que a consulta q se unifica com alguma sentença da KB

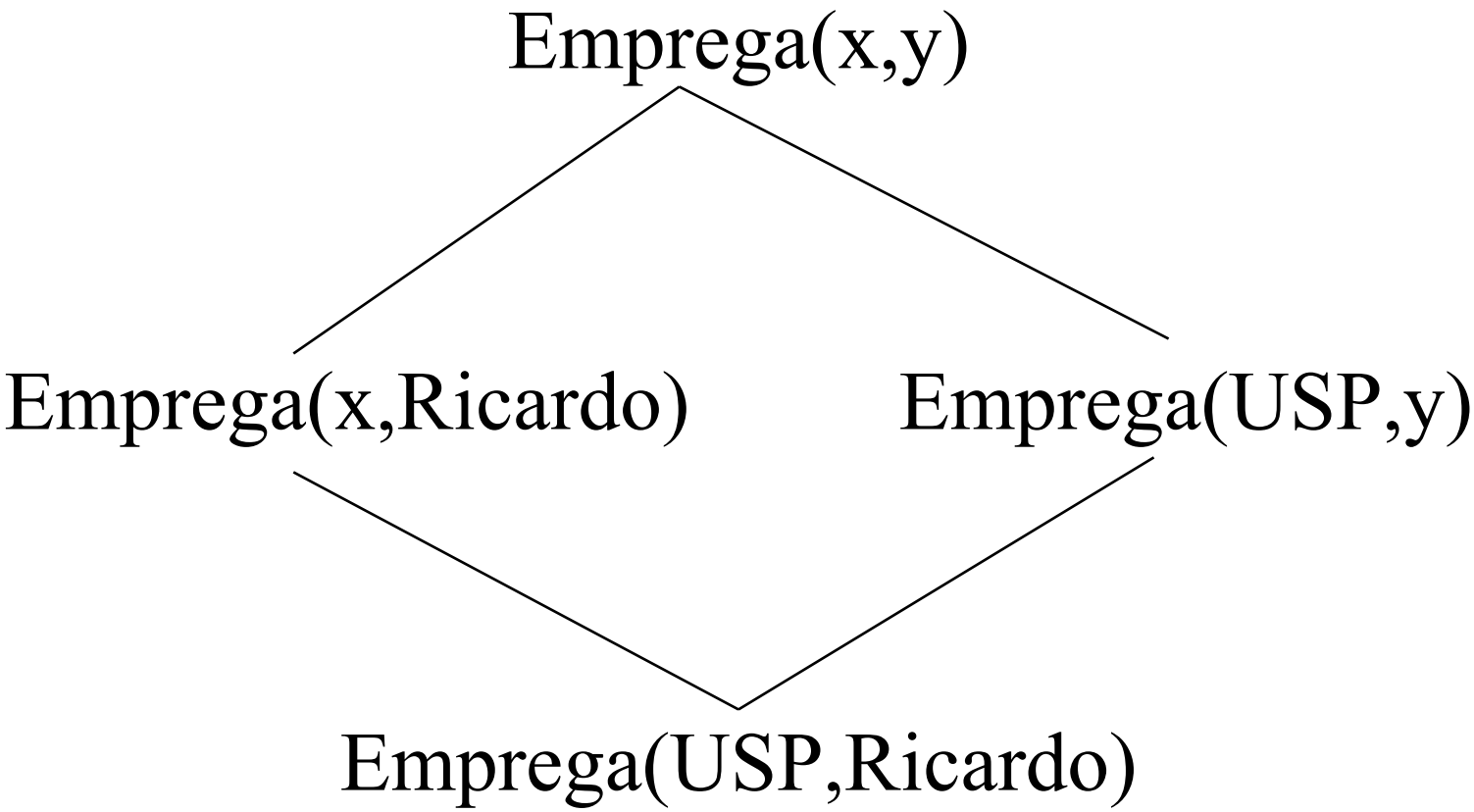
$$KB \models SUBST(\theta, q)$$

- Uso de uma tabela de hash para indexar os fatos da KB por várias chaves:
  - predicado e primeiro argumento ou
  - predicado e segundo argumento ...

Reticulado de subordinação

- Dada uma sentença a ser armazenada, é possível construir índices para todas as consultas possíveis que se unificam com ela
- Essas consultas formam um **reticulado de subordinação**

Reticulado de subordinação



Reticulado de subordinação

Propriedades:

- O filho de qualquer nó no reticulado é obtido a partir de seu pai por uma única substituição
- O mais alto descendente comum de dois nós quaisquer é o resultado da aplicação do unificador geral

- Para um predicado com *n* argumentos, o reticulado contém $O(2^n)$ nós. Se forem permitidos símbolos de funções, o número de nós será exponencial no tamanho dos termos da sentença a ser armazenada → número muito grande de índices
- Para a maioria dos sistemas de IA, o número de fatos é pequeno o bastante para uma indexação eficiente. Isso pode não ser verdade para bancos de dados comerciais.

Resolução

- Como foi dito para a Lógica Proposicional ao invés de usarmos o conjunto de (7) regras de inferência definidos como *consistentes* podemos usar uma regra de inferência única: a **resolução**
  - ... gerando um algoritmo de inferência completo quando acoplado a um algoritmo de busca completo

7 regras de inferência para a LP

$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$	Modus Ponens	Da implicação e da premissa infere-se a conclusão
$\frac{\alpha_1 \Lambda \alpha_2 \Lambda \ \ldots \ \Lambda \alpha_n}{\alpha_n}$	Eliminação	Da conjunção infere-se qualquer α_n
$\frac{\alpha_1, \alpha_2, \ \ldots\ , \ \alpha_n}{\alpha_1 \Lambda \alpha_2 \Lambda \ \ldots \ \Lambda \alpha_n}$	Introdução da conjunção	De uma lista de sentenças infere-se a sua conjunção
$\frac{\alpha_i}{\alpha_1 V \alpha_2 V \ \ldots \ V \alpha_n}$	Introdução da disjunção	De uma sentença, infere-se sua disjunção com qualquer outra
$\frac{\neg\neg\alpha}{\alpha}$	Negação dupla	De uma negação dupla infere-se uma senetnça positiva
$\frac{\alpha \ V \ \beta, \neg\beta}{\alpha}$	Resolução simples	Se uma das disjunções for falsa, pode-se inferir que a outra é verdade
$\frac{\alpha \ V \ \beta, \neg\beta \ V \ \gamma}{\alpha \ V \ \gamma}$	Resolução	β , não pode ser Verdade e Falso ao mesmo tempo

Regra de resolução unitária

$$\frac{l_1 \vee \ldots \vee l_k \text{ , m}}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \ldots \vee l_k}$$

onde l_i e m são literais complementares (, isto é, l é a negação de m).

Regra de resolução completa

$$l_1 \vee \ldots \vee l_k \, , m_1 \vee \ldots \vee m_k$$

$$l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{i-1} \vee m_{i+1} \ldots m_k$$

onde l_i e m_i são literais complementares.

Regra de resolução completa

- Pega duas cláusulas e transforma numa nova cláusula, contendo todos os literais das duas cláusulas originais, *exceto* os dois literais complementares.
- ***Fatoração***: a cláusula resultante deve conter apenas uma cópia de cada literal

Regra de resolução

- Não serve para gerar todas as conseqüências lógicas da KB mas serve para provar se a KB satisfaz ou não uma sentença α.
- Como a regra de resolução só se aplica a disjunções de literais, é preciso transformar a KB na *forma normal conjuntiva* (FNC)
  - Toda sentença da LP ou LPO é logicamente equivalente a uma *conjunção de disjunções de literais* (FNC)

Forma Normal Conjuntiva

- Como converter a sentença do Mundo do Wumpus:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

para a forma normal conjuntiva?

Forma Normal Conjuntiva

1. Eliminar $\Leftrightarrow$
   - Substituindo $\alpha \Leftrightarrow \beta$ por $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
2. Eliminar $\rightarrow$
   - Substituindo $\alpha \rightarrow \beta$ por $\neg\alpha \vee \beta$
3. Eliminar a negação de expressões (deixar somente a negação de literais), através das seguintes equivalências lógicas:

$\neg(\neg\alpha) \equiv \alpha$

$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$

Inferência baseada na resolução

Usa o princípio de prova por contradição:

- para provar que KB |= α, mostramos que (KB $\wedge\neg$ α) é não satisfatível (isto é, não existe um modelo que satisfaz a fórmula)

Cláusula vazia - contradição

A cláusula vazia é gerada resolvendo-se duas cláusulas unitárias, P e $\neg P$, o que representa uma contradição

cláusula vazia → *disjunção de nenhum disjunto* → equivale a *Falso*

Algoritmo de resolução

1. (KB $\wedge\neg$ α) é convertida em uma FNC
2. Aplica a regra de resolução na FNC resultante → cada par de cláusulas que contém literais complementares é *resolvido* para gerar uma nova cláusula que é inserida na KB, se ainda não estiver presente.
3. O passo 2 continua até que:
   - Não exista nenhuma cláusula nova que possa ser adicionada, nesse caso KB $|\neq \alpha$ ou
   - Uma aplicação da regra de resolução deriva a ***cláusula vazia***, nesse caso KB $|= \alpha$

Algoritmo de resolução

função RESOLUÇÃO-LP(KB,α) devolve *verdadeiro* ou *falso*

entradas: KB e uma sentença α (consulta) em LP

cláusulas $\leftarrow$ o conjunto de cláusulas FNC de KB $\wedge \neg \alpha$

nova $\leftarrow$ { }

repita

para cada C_i, C_j **em** *cláusulas* **faça**

resolventes $\leftarrow$ RESOLVER-LP(C_i, C_j)

se *resolventes* contém a cláusula vazia **então**

devolver *verdadeiro*

nova $\leftarrow$ *nova* $\cup$ *resolventes*

se *nova* $\subseteq$ *cláusulas* **então devolver** *falso*

cláusulas $\leftarrow$ *cláusulas* $\cup$ *nova*

Algoritmo de resolução

função RESOLUÇÃO-LP(KB,α) devolve *verdadeiro* ou *falso*

entradas: KB e uma sentença α (consulta) em LP

cláusulas ← {}
cláusulas' ← o conjunto de cláusulas FNC de KB $\wedge\neg$ α
nova ← { }
repita
  para cada C_i, C_j **em** *cláusulas'* **faça**
    resolventes ← RESOLVER-LP(C_i, C_j)
    se *resolventes* contém a cláusula vazia **então**
      devolver *verdadeiro*
    nova ← *nova* ∪ *resolventes*
  cláusulas ← *cláusulas'* ∪ *nova*
até cláusulas = cláusulas'
devolva falso

Exercício: resolução no Mundo do Wumpus

- *Se não há brisa em [1,1] então não há poços nos quadrados vizinhos*
- KB:

$$(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

- Queremos provar $\alpha \equiv \neg P_{1,2}$
- Portanto, KB $\wedge \neg \alpha$ fica:

$$(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \wedge \neg(\neg P_{1,2})$$

Exercício: resolução no Mundo do Wumpus

- KB $\wedge \neg \alpha$ ➔

$$(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \wedge \neg(\neg P_{1,2})$$

- Corresponde à CNF:

...

(prova)

Cláusulas de Horn

- Cláusula de Horn: uma disjunção de literais com no máximo um literal positivo. Ex:

$$\neg A \vee \neg B \vee \neg C \vee D$$

- Cláusulas de Horn podem ser convertidas para implicações com premissas positivas e conclusão com um único literal positivo

$$A \wedge B \wedge C \rightarrow D$$

Tipos de cláusulas de Horn

Cláusula definida:

$$\neg A \vee \neg B \vee \neg C \vee D$$

corpo

cabeça

Fato: cláusula sem literais negativos

$$\rightarrow D$$

Restrições de integridade: cláusulas sem literais positivos

$$\neg A \vee \neg B \vee \neg C$$

premissas

$$A \wedge B \wedge C \rightarrow \textit{Falso}$$

conclusão

Encadeamento para frente e para trás

- A inferência com cláusulas de Horn pode ser feita com algoritmos de ***encadeamento para frente*** e ***encadeamento para trás***
  - Algoritmos que consomem tempo linear em relação ao tamanho da KB

Encadeamento para frente

Queremos verificar se KB |= α

- A partir do conjunto de fatos da KB verificamos as premissas de uma implicação. Se todas forem verdadeiras, a conclusão é adicionada à KB (*modus ponens*).
- Esse processo continua até α ser adicionada à KB ou até não ser possível fazer inferências adicionais (*ponto fixo*)

Encadeamento para frente numa KB de cláusulas de Horn (KB |= Q)

$P \rightarrow Q$

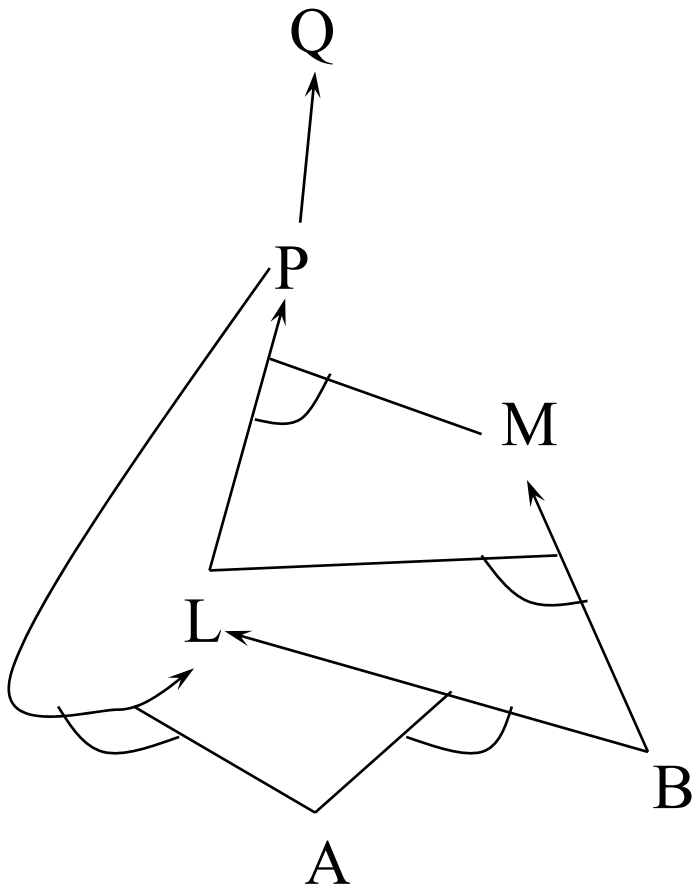
$L \wedge M \rightarrow P$

$B \wedge L \rightarrow M$

$A \wedge B \rightarrow L$

A

B



Raciocínio orientado a dados;
a partir dos dados

Encadeamento para trás

- Raciocínio orientado por metas; a partir da consulta
  - Se Q já é verdade na KB o algoritmo para
  - Caso contrário, o algoritmo procura as implicações na KB que possuem Q como conclusão
  - Se é possível provar todas as premissas dessa implicação (também por encadeamento para trás) então Q é verdadeira
- Raciocínio em que se baseiam as linguagens de programação lógica., por exemplo, PROLOG