Centro de Ciências Exatas, Ambientais e de Tecnologias

Faculdade de Análise de Sistemas

Curso de Sistemas de Informação

Paradigmas de Programação

2º Semestre de 2022

Volume 2

Prof. André Luís dos R.G. de Carvalho

Índice

ÍNDICE **2**

CAPÍTULO I: O PARADIGMA LÓGICO **4**

INTRODUÇÃO **5**

CÁLCULO DE PREDICADOS **5**

PROPOSIÇÕES 6
CLÁUSULAS 6
CLÁUSULAS DE HORN 7

RESOLUÇÃO **8**

EXEMPLO 10

CONCLUSÃO **18**

ANEXO I: REFERÊNCIAS **20**

CAPÍTULO II: A LINGUAGEM DE PROGRAMAÇÃO PROLOG **21**

INTRODUÇÃO **22**

ALGUMAS DEFINIÇÕES **22**

Símbolo *22*
Predicados *22*
Fatos *23*
Regras *23*

SWI-PROLOG **24**

COMENTÁRIOS 24
EXEMPLO DE UM PROGRAMA 24
PARA CARREGAR UM PROGRAMA 25
EXEMPLO DE EXECUÇÃO 25
PREDICADOS PREDEFINIDOS 26
Verificação de Tipo *26*
Comparação e Unificação *26*
Predicados de Controle *28*
OPERADORES 28
Aritméticos *28*
De Bit *28*
FUNÇÕES MATEMÁTICAS 28
FUNCTORS 29
Exemplo de Programa *29*
Exemplo de Execução *30*
LISTAS 30
Exemplo de Programa 1 ..... *31*
Exemplos de Execução *31*

Algoritmo da Resolução aplicada ao último Exemplo de Execução *31*
Exemplo de Programa 2 ..... *33*
Exemplo de Execução *33*
Algoritmo da Resolução aplicada ao Exemplo de Execução *33*

ANEXO I: EXERCÍCIOS PROPOSTOS **36**

ANEXO II: REFERÊNCIAS **52**

Capítulo I:
O Paradigma Lógico

Introdução

O paradigma lógico se baseia em lógica simbólica e usa inferência lógica para produzir resultados. Linguagens lógicas são profundamente diferentes das linguagens imperativas e das linguagens funcionais [Sebesta89].

A abordagem do paradigma lógico à resolução de problemas foi desenvolvida para prova automática de teoremas [Sebesta89]. Assim, escrever um programa lógico é essencialmente a mesma coisa que escrever um programa que prova um teorema [Dershem/Jipping90].

Para compreender melhor como funciona um programa baseado neste paradigma, examinemos agora como um teorema é provado.

Parte-se de uma série de fatos que são tidos como verdadeiros, ou axiomas em linguajar matemático. Manipula-se então os axiomas usando para tanto regras de inferência lógica até se chegar à proposição que se quer provar. A proposição a ser provada é considerada uma espécie de objetivo a ser alcançado.

Um programa escrito segundo o modelo de programação lógica envolve um banco de dados composto por uma série de fatos, e por uma coleção de regras de inferência que estabelecem relações entre fatos, tudo isto expresso em uma sintaxe própria à linguagem. Este banco de dados, aliado a um processo automático de inferência, é usado para verificar a validade de novas proposições.

Cálculo de Predicados

Uma proposição é uma afirmação que pode ou não ser verdadeira, e pode envolver diversos objetos e relações entre eles. A lógica formal foi desenvolvida para prover meios para descrever proposições formalmente, de modo a verificar sua veracidade.

Lógica simbólica pode ser usada para estas três finalidades básicas: expressar proposições, expressar relacionamentos entre proposições, e descrever como proposições podem ser inferidas a partir de proposições que se sabe serem verdadeiras.

Existe muita semelhança entre a lógica formal e a matemática. Na verdade muito da matemática pode ser pensado em termos da lógica formal. O cálculo de predicados, que é uma particular forma de lógica simbólica, tem sido muito usado em programação lógica.

Proposições

Proposições, no cálculo de predicados, são afirmações a cerca de objetos. Objetos são representados por termos simples que podem ser variáveis ou constantes. Constantes são símbolos que representam um determinado objeto. Variáveis são símbolos que representam objetos genéricos, podendo representar diferentes objetos em diferentes instantes. Variáveis somente podem ser introduzidos em uma proposição se introduzidos por um quantificador universal (∀) ou existencial (∃).

As proposições mais simples que existem são compostas por um único termo, e são chamadas de proposições atômicas. Termos são relações matemáticas escritas sob a forma de expressão funcional. Uma expressão funcional consiste de um functor, ou símbolo de função, seguido por uma lista ordenada de argumentos. Termos representam propriedades e relações entre objetos.

Proposições compostas podem ter duas ou mais proposições atômicas conectadas por uma negação (¬), por uma conjunção (∩), por uma disjunção (∪) ou por uma implicação (⊃).

Veja abaixo um exemplo de duas proposições. A primeira é uma proposição atômica, e afirma que joão é humano. A segunda é uma proposição composta, e afirma que todo humano é mortal.

```
⊃ humano (joao)
(∀X) humano (X) ⊃ mortal (X)
```

Cláusulas

Como foi descrito até agora, o cálculo de predicados permite muitas descrições diferentes para uma mesma proposição.

Para simplificar e padronizar a forma das cláusulas lógicas, define-se a forma clausal. Sem perda de generalidade, qualquer proposição pode ser reduzida à forma clausal [Sebesta89]. Veja abaixo a forma geral de uma proposição na forma clausal.

$$C_1 \cup C_2 \cup \ldots \cup C_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$$

Nesta proposição na forma clausal, tanto os A_i quanto os C_i são termos, em geral, predicados. Ela significa que se todos os A_i forem verdadeiros, pelo menos um dentre os C_i também será verdadeiro.

A parte da forma clausal constituída pelos termos C_i (a parte que se encontra à esquerda da implicação na cláusula acima, embora possa, eventualmente, em uma outra proposição, estar à direita, naturalmente, desde que o sinal de implicação se apresente também com sentido invertido) é chamada de consequente, ao passo que, a parte da forma clausal constituída pelos termos A_i (a parte que se encontra à direita da implicação na cláusula acima, embora possa, eventualmente, em uma outra proposição, estar à esquerda, naturalmente, desde que o sinal de implicação se apresente também com sentido invertido) é chamada de antecedente.

Na forma clausal não se emprega o quantificador universal (∀) e tampouco o quantificador existencial (∃). Por convenção, todas as variáveis presentes em uma cláusula devem ser encaradas como qualificadas implicitamente pelo quantificador universal (∀).

Cláusulas de Horn

Trata-se do tipo de cláusula que é usado na programação lógica. Possuem a característica de terem a parte consequente no máximo com um termo (eventualmente pode ser vazia).

Na programação lógica, são utilizadas quatro tipos de cláusulas de Horn, a saber:

1. **<u>Fato</u>:**

   Representam afirmações incondicionais, ou seja, afirmações que devem ser consideradas verdadeiras independentemente de quaisquer condições.

   Possuem a parte conseqüente com um termo e a parte antecedente vazia.

   <u>Ex</u>.:

   `humano(joao)⊂`

2. **<u>Regra</u>:**

Representam afirmações condicionais, ou seja, afirmações que, para serem consideradas verdadeiras, dependem de que certas condições se provem verdadeiras.

Possuem a parte conseqüente com um termo e a parte antecedente com um ou mais termos.

Ex.:

```
mortal(X)⊂ humano(X)
```

3. **Pergunta (ou objetivo):**

Representam perguntas que se pode fazer para um banco de conhecimento constituído por fatos e regras.

Possuem a parte conseqüente vazia e a parte antecedente com um ou mais termos.

Ex.:

```
⊂ mortal(joao)
```

4. **Sucesso:**

O Algoritmo da Resolução é o algoritmo que se emprega para obter respostas para perguntas a partir de um banco de conhecimentos constituído por fatos e regras. Este tipo de cláusula, quando obtida pelo Algoritmo da Resolução, faz com que seja entendido que o mesmo já chegou à determinação de uma resposta.

Possuem a parte conseqüente vazia e a parte antecedente também vazia.

Ex.:

```
⊂
```

Resolução

Resolução é uma regra de inferência concebida para ser aplicada a proposições na forma clausal que tem um potencial muito grande de aplicação na prova automática de teoremas [Sebesta89]. O conceito de resolução é o seguinte: suponha que existam duas proposições da forma:

$$C_1 \subset A_1$$
$$C_2 \subset A_2$$

Suponha ainda que C_1 seja idêntica a A_2. Poderíamos então chamar C_1 e A_2 de P. Assim, rescrever as duas proposições como segue:

$$P \subset A_1$$
$$C_2 \subset P$$

Agora, como A_1 implica P e P implica seja idêntica a C_2, é obvio que A_1 implica C_2. Assim, poder-se-ia escrever

$$C_2 \subset A_1$$

Este processo, através do qual pode-se obter esta proposição a partir das duas anteriores, é chamado de resolução, e constitui a base do processo de inferência automática das linguagens de programação lógicas.

Se tanto a parte antecedente quando a conseqüente das expressões possuírem diversos termos, a nova proposição inferida conterá todos os termos das duas proposições, exceto aquele comum às ambas que poderá ser cancelado.

Na verdade, o processo de inferência é bem mais complicado do que foi mostrado aqui. Por exemplo, se existirem variáveis nas proposições, será necessário encontrar valores para estas variáveis que permitam o sucesso do processo de emparelhamento.

O processo de determinação de valores para as variáveis é chamado de unificação, e a associação temporária de valores às variáveis é chamada instanciação.

É comum o processo de resolução produzir instanciações que se mostram incapazes de comprovar a veracidade da proposição objetivo, sendo forçado a retornar a objetivos anteriores e alterar instanciações que tenham sido feitas.

A propriedade crucial do mecanismo de resolução é sua habilidade de detectar inconsistências em um dado conjunto de proposições. Esta propriedade permite que se faça uso deste mecanismo para provar teoremas, o que é feito como explicado abaixo.

Imagine a prova de um teorema expresso em termos de cálculo de predicados como um conjunto pertinente de proposições ao qual se acrescenta o próprio teorema que se deseja provar negado.

O teorema negado é introduzido no conjunto de proposições para que o mecanismo de resolução acuse uma inconsistência, e o teorema seja provado por contradição. Tipicamente, o conjunto original de proposições é chamado de conjunto das hipóteses, e o teorema negado é chamado de objetivo.

Prova de teoremas é a base da programação lógica. Teoricamente, todo o mecanismo que foi visto até agora funciona muito bem. No entanto, na prática, programas lógicos podem ter um tempo de execução enorme se o banco de dados com as proposições for muito grande.

Normalmente, usa-se apenas um tipo restrito de forma clausal para representar proposições que se pretende trabalhar com o mecanismo de resolução. Este tipo especial de proposição é chamado de Cláusula de Horn.

A restrição que se impõe a estas cláusulas é que tenha a parte conseqüente vazia, ou constituída somente por uma proposição atômica.

Exemplo

Considere o seguinte banco de conhecimentos:

nasceu(jose,m,eleazar,ana)⊂
nasceu(sebastiao,m,eleazar,ana)⊂
nasceu(neusa,f,zair,maria)⊂
nasceu(paulo,m,zair,maria)⊂
nasceu(andre,m,jose,neusa)⊂
nasceu(zeluis,m,jose,neusa)⊂
nasceu(marcoantonio,m,sebastiao,zulma)⊂
nasceu(marcioaugusto,m,sebastiao,zulma)⊂
nasceu(tieni,f,sebastiao,nilce)⊂
nasceu(marcel,m,paulo,marialuiza)⊂
nasceu(marcelo,m,paulo,marialuiza)⊂
nasceu(marcioluis,m,paulo,marialuiza)⊂
nasceu(hugo,m,andre,lilia)⊂
…
morreu(zulma)⊂
…

casou(eleazar, ana)⊂
casou(zair, maria)⊂
casou(jose,neusa)⊂
casou(sebastiao,zulma)⊂
casou(sebastiao,nilce)⊂
casou(andre,lilia)⊂
…
separou(andre,lilia)⊂
…
pai(P,F)⊂nasceu(F,_,P,_)
mae(M,F)⊂nasceu(F,_,_,M)
genitor(G,F)⊂mae(G,F)
genitor(G,F)⊂pai(G,F)
avo(A,N)⊂pai(A,G)∩genitor(G,N)
…

Considere que seja feita a seguinte pergunta:

⊂avo(zair,andre)

O Algoritmo da Resolução faria as seguintes manipulações lógicas envolvendo a cláusula que exprime a pergunta e aquelas que constituem o banco de conhecimento:

Manipulação 1:
⊂avo(zair,andre)
avo(A,N)⊂pai(A,G)∩genitor(G,N)
~~avo(A,N)⊂avo(zair,andre)~~∩pai(A,G)∩genitor(G,N)
Fazendo as instanciações A\zair e N\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂pai(zair,G)∩genitor(G,andre)

Manipulação 2:
⊂pai(zair,G)∩genitor(G,andre)
pai(P,F)⊂nasceu(F,_,P,_)
~~pai(P,F)⊂pai(zair,G)~~∩nasceu(F,_,P,_)∩genitor(G,andre)
Fazendo as instanciações P\zair e F\G, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂nasceu(F,_,zair,_)∩genitor(F,andre)

Manipulação 3:
⊂nasceu(F,_,zair,_)∩genitor(F,andre)
nasceu(jose,m,eleazar,ana)⊂
~~nasceu(jose,m,eleazar,ana)⊂nasceu(F,_,zair,_)~~∩genitor(F,andre)

Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 3, voltanto a ter que verificar ⊂nasceu(F,_,zair,_)∩genitor(F,andre)

Manipulação 4:
⊂nasceu(F,_,zair,_)∩genitor(F,andre)
nasceu(sebastiao,m,eleazar,ana)⊂
~~nasceu(sebastiao,m,eleazar,ana)⊂nasceu(F,_,zair,_)~~∩genitor(F,andre)
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 4, voltanto a ter que verificar ⊂nasceu(F,_,zair,_)∩genitor(F,andre)

Manipulação 5:
⊂nasceu(F,_,zair,_)∩genitor(F,andre)
nasceu(neusa,f,zair,maria)⊂
~~nasceu(neusa,f,zair,maria)⊂nasceu(F,_,zair,_)~~∩genitor(F,andre)
Fazendo as instanciações F\neusa, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂genitor(neusa,andre)

Manipulação 6:
⊂genitor(neusa,andre)
genitor(G,F)⊂mae(G,F)
~~genitor(G,F)⊂genitor(neusa,andre)~~∩mae(G,F)
Fazendo as instanciações G\neusa e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂mae(neusa,andre)

Manipulação 7:
⊂mae(neusa,andre)
mae(M,F)⊂nasceu(F,_,_,M)
~~mae(M,F)⊂mae(neusa,andre)~~∩nasceu(F,_,_,M)
Fazendo as instanciações M\neusa e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂nasceu(andre,_,_,neusa)

Manipulação 8:
⊂nasceu(andre,_,_,neusa)
nasceu(jose,m,eleazar,ana)⊂
~~nasceu(jose,m,eleazar,ana)⊂nasceu(andre,_,_,neusa)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 8, voltanto a ter que verificar ⊂nasceu(andre,_,_,neusa)

Manipulação 9:
⊂nasceu(andre,_,_,neusa)
nasceu(sebastiao,m,eleazar,ana)⊂
~~nasceu(sebastiao,m,eleazar,ana)⊂ nasceu(andre,_,_,neusa)~~

Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 9, voltanto a ter que verificar ⊂nasceu(andre,_,_,neusa)

Manipulação 10:
⊂nasceu(andre,_,_,neusa)
nasceu(neusa,f,zair,maria)⊂
~~nasceu(neusa,f,zair,maria)⊂ nasceu(andre,_,_,neusa)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 10 voltanto a ter que verificar ⊂nasceu(andre,_,_,neusa)

Manipulação 11:
⊂nasceu(andre,_,_,neusa)
nasceu(paulo,m,zair,maria)⊂
~~nasceu(paulo,m,zair,maria)⊂ nasceu(andre,_,_,neusa)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 11, voltanto a ter que verificar ⊂nasceu(andre,_,_,neusa)

Manipulação 12:
⊂nasceu(andre,_,_,neusa)
nasceu(andre,m,jose,neusa)⊂
~~nasceu(andre,m,jose,neusa)⊂nasceu(andre,_,_,neusa)~~
Podemos cancelar os termos riscados em razão dos mesmo serem equivalentes, restando verificar ⊂

A cláusula constituida apenas por um símbolo de implicação (⊂) representa sucesso, i.e., a pergunta original tem resposta positiva.

Considere agora que seja feita a seguinte pergunta:

⊂avo(QUEM,andre)

O Algoritmo da Resolução faria as seguintes manipulações lógicas envolvendo a cláusula que exprime a pergunta e aquelas que constituem o banco de conhecimento:

Manipulação 1:
⊂avo(QUEM,andre)
avo(A,N)⊂pai(A,G)∩genitor(G,N)
~~avo(A,N)⊂avo(QUEM,andre)~~∩pai(A,G)∩genitor(G,N)
Fazendo as instanciações A\QUEM e N\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂pai(QUEM,G)∩genitor(G,andre)

Manipulação 2:
⊂pai(QUEM,G)∩genitor(G,andre)
pai(P,F)⊂nasceu(F,_,P,_)
~~pai(P,F)⊂pai(QUEM,G)~~∩nasceu(F,_,P,_)∩genitor(G,andre)
Fazendo as instanciações P\QUEM e F\G, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂nasceu(F,_,QUEM,_)∩genitor(F,andre)

Manipulação 3:
⊂nasceu(F,_,QUEM,_)∩genitor(F,andre)
nasceu(jose,m,eleazar,ana)⊂
~~nasceu(jose,m,eleazar,ana)⊂nasceu(F,_,QUEM,_)~~∩genitor(F,andre)
Fazendo as instanciações F\jose e QUEM\eleazar, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂genitor(jose,andre)

Manipulação 4:
⊂genitor(jose,andre)
genitor(G,F)⊂mae(G,F)
~~genitor(G,F)⊂genitor(jose,andre)~~∩mae(G,F)
Fazendo as instanciações G\jose e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂mae(jose,andre)

Manipulação 7:
⊂mae(jose,andre)
mae(M,F)⊂nasceu(F,_,_,M)
~~mae(M,F)⊂mae(jose,andre)~~∩nasceu(F,_,_,M)
Fazendo as instanciações M\jose e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂nasceu(andre,_,_,jose)

Manipulação 8:
⊂nasceu(andre,_,_,jose)
nasceu(jose,m,eleazar,ana)⊂
~~nasceu(jose,m,eleazar,ana)⊂nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 8, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 9:
⊂nasceu(andre,_,_,jose)
nasceu(sebastiao,m,eleazar,ana)⊂
~~nasceu(sebastiao,m,eleazar,ana)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 9, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 10:

⊂nasceu(andre,_,_,jose)
nasceu(neusa,f,zair,maria)⊂
~~nasceu(neusa,f,zair,maria)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 10 voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 11:
⊂nasceu(andre,_,_,jose)
nasceu(paulo,m,zair,maria)⊂
~~nasceu(paulo,m,zair,maria)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 11, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 12:
⊂nasceu(andre,_,_,jose)
nasceu(andre,m,jose,neusa)⊂
~~nasceu(andre,m,jose,neusa)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 12, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 13:
⊂nasceu(andre,_,_,jose)
nasceu(zeluis,m,jose,neusa)⊂
~~nasceu(zeluis,m,jose,neusa)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 13, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 14:
⊂nasceu(andre,_,_,jose)
nasceu(marcoantonio,m,sebastiao,zulma)⊂
~~nasceu(marcoantonio,m,sebastiao,zulma)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 14, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 15:
⊂nasceu(andre,_,_,jose)
nasceu(marcioaugusto,m,sebastiao,zulma)⊂
~~nasceu(marcioaugusto,m,sebastiao,zulma)⊂ nasceu(andre,_,_,jose)~~

Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 15, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 16:
⊂nasceu(andre,_,_,jose)
nasceu(tieni,f,sebastiao,nilce)⊂
~~nasceu(tieni,f,sebastiao,nilce)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 16, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 17:
⊂nasceu(andre,_,_,jose)
nasceu(marcel,m,paulo,marialuiza)⊂
~~nasceu(marcel,m,paulo,marialuiza)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 17, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 18:
⊂nasceu(andre,_,_,jose)
nasceu(marcelo,m,paulo,marialuiza)⊂
~~nasceu(marcelo,m,paulo,marialuiza)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 18, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 19:
⊂nasceu(andre,_,_,jose)
nasceu(marcioluis,m,paulo,marialuiza)⊂
~~nasceu(marcioluis,m,paulo,marialuiza)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 19, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 20:
⊂nasceu(andre,_,_,jose)
nasceu(hugo,m,andre,lilia)⊂
~~nasceu(hugo,m,andre,lilia)⊂ nasceu(andre,_,_,jose)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 20, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 21:

⊂nasceu(andre,_,_,jose)
Como não existem mais fatos ou regras que tenham o predicado nasceu no conseqüente, desfaremos a Manipulação 7, voltanto a ter que verificar ⊂mae(jose,andre)

Manipulação 22:
⊂mae(jose,andre)
Como não existem mais fatos ou regras que tenham o predicado mae no conseqüente, desfaremos a Manipulação 4, voltanto a ter que verificar ⊂genitor(jose,andre)

Manipulação 23:
⊂genitor(jose,andre)
genitor(G,F)⊂pai(G,F)
~~genitor(G,F)⊂genitor(jose,andre)~~∩pai(G,F)
Fazendo as instanciações G\jose e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂pai(jose,andre)

Manipulação 24:
⊂pai(jose,andre)
pai(P,F)⊂nasceu(F,_,P,_)
~~pai(P,F)~~⊂pai~~(jose,andre)~~∩nasceu(F,_,P,_)
Fazendo as instanciações P\jose e F\andre, podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂nasceu(andre,_,jose,_)

Manipulação 25:
⊂nasceu(andre,_,jose,_)
nasceu(jose,m,eleazar,ana)⊂
~~nasceu(jose,m,eleazar,ana)⊂nasceu(andre,_,jose,_)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 25, voltanto a ter que verificar ⊂nasceu(andre,_,jose,_)

Manipulação 26:
⊂nasceu(andre,_,jose,_)
nasceu(sebastiao,m,eleazar,ana)⊂
~~nasceu(sebastiao,m,eleazar,ana)⊂ nasceu(andre,_,jose,_)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 26, voltanto a ter que verificar ⊂nasceu(andre,_,jose,_)

Manipulação 27:
⊂nasceu(andre,_,jose,_)
nasceu(neusa,f,zair,maria)⊂
~~nasceu(neusa,f,zair,maria)⊂ nasceu(andre,_,jose,_)~~

Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 27 voltanto a ter que verificar ⊂nasceu(andre,_,jose,_)

Manipulação 28:
⊂nasceu(andre,_,jose,_)
nasceu(paulo,m,zair,maria)⊂
~~nasceu(paulo,m,zair,maria)⊂ nasceu(andre,_,jose,_)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, desfaremos a Manipulação 28, voltanto a ter que verificar ⊂nasceu(andre,_,_,jose)

Manipulação 29:
⊂nasceu(andre,_,jose,_)
nasceu(andre,m,jose,neusa)⊂
~~nasceu(andre,m,jose,neusa)⊂ nasceu(andre,_,jose,_)~~
Podemos cancelar os termos riscados em razão dos mesmo serem equivalentes, restando verificar ⊂

A cláusula constituida apenas por um símbolo de implicação (⊂) representa sucesso e, como a variável QUEM ficou valendo eleazar na Manipulação 3, a pergunta original tem QUEM=eleazar. Para obter mais respostas para a pergunta original, desfaz-se a manipulação 29 como se a mesma não houvesse tido sucesso no cancelamento, continuando o processo até atingir sucesso novamente.

Conclusão

As linguagens lógicas são muitas vezes chamadas de declarativas porque os programas lógicos são constituídos por declarações, que chamamos de proposições, em vez de serem constituídos por atribuições e comandos para controle do fluxo de execução.

Dentre as áreas onde atual e potencialmente podem ser encontradas aplicações baseadas no paradigma funcional, podemos destacar: sistemas gerenciadores de bancos de dados relacionais, sistemas especialistas, processamento de linguagens naturais e educação.

Uma característica essencial das linguagens lógicas é que a semântica de cada um de seus elementos constituintes, as proposições, pode ser compreendida de maneira isolada, posto que o significado de uma proposição não interfere no significado de outra, o que não acontece nas linguagens que seguem o paradigma imperativo [Sebesta89].

Linguagens lógicas são não procedimentais, o que significa que, diferentemente do que acontece nas linguagens imperativas, nestas linguagens não se especifica como atingir um resultado, e sim a forma do resultado.

A diferença fundamental é que, nas linguagens lógicas, assume-se a existência de recursos suficientes para que seja determinada a forma de se atingir um certo resultado.

Para tanto, tudo que deve existir é um meio conciso de fornecer as informações relevantes no contexto do problema a ser resolvido, e um mecanismo de inferência para se chegar aos resultados desejados.

O cálculo de predicados provê a forma para comunicar as informações relevantes, e o mecanismo de resolução provê a técnica de inferência.

Anexo I: Referências

[Sebesta89] Sebesta, R.W., "Concepts of Programming Languages", The Benjamin/Cummings Publishing Company, Inc., 1989.

[Dershem/Jipping90] Dershem, H.L; Jipping, M.J., "Programming Languages: Structures and Models", Wadsworth, Inc., 1990.

Capítulo II:
A Linguagem
de Programação PROLOG

Introdução

A linguagem de programação de uso corrente que melhor representa o paradigma lógico é a linguagem Prolog, que é uma abreviação de programação em lógica.

Sua sintaxe é uma versão modificada de cálculo de predicados. Foi projetada no início dos anos 70, num trabalho conjunto de Alain Colmerauer e Phillippe Roussel, da Universidade de Aix-Marsseille, e de Robert Kowalski, da Universidade de Edimburgo. Sua primeira implementação data do ano de 1972.

Algumas Definições

Símbolo

O conceito de símbolo é idêntico ao conceito de átomo simbólico na linguagem LISP. Assim, os entendemos como algo que simboliza um objeto de interesse do mundo real.

Veja abaixo exemplos de símbolos:

```
jose      pera      cadeira      verde
```

Note que sempre grafamos símbolos empregando letras minúsculas.

Predicados

Entendemos que um predicado é uma forma de expressar uma propriedade de um símbolo ou um vínculo entre símbolos. Predicados tem forma funcional e podem não ter argumentos ou ter qualquer número de argumentos. Predicados podem ter múltiplas definições.

Veja abaixo exemplos de predicados:

```
pessoa (jose)      fruta (goiaba)      pai (joao, maria)
```

Eles representam, respectivamente, que:

- José é uma pessoa;
- Goiaba é uma fruta; e
- João é pai de Maria.

Note que sempre grafamos predicados empregando letras minúsculas.

Fatos

Um fato é uma afirmação a respeito de um ou mais símbolos. Um fato sempre é sempre representado por um predicado seguida por um ponto final.

Veja abaixo exemplos de fatos:

```
pessoa(jose).
fruta(goiaba).
pai(joao, maria).
```

Eles nos afirmam, respectivamente, que:

- José é uma pessoa;
- Goiaba é uma fruta; e
- João é pai de Maria.

Regras

Uma regra é uma afirmação condicional a respeito de um ou mais símbolos. Uma regra tem sempre a seguinte forma geral:

$P_1(E_{a1},E_{a2},...,E_{an})$:- [not] $P_2(E_{b1},E_{b2},...,E_{bm})${|,|;|[not] $P_3(E_{c1}, E_{c2},...,E_{ck})$}.

Onde os P_i's são predicados, os E_i's são os envolvidos nos predicados, o :- deve ser entendido como implicação, o not deve ser entendido como negação, a vírgula deve ser entendida com conjunção e o ponto-e-vírgula como disjunção.

Uma regra pode envolver símbolos e/ou variáveis. Quando temos um símbolo envolvido em uma regra, entendemos que afirmações estão sendo feitas a respeito de um objeto específico simbolizado por aquele símbolo. Quando temos uma variável envolvida em uma regra, entendemos que afirmações estão sendo feitas a um objeto genérico, não a um específico.

Veja abaixo alguns exemplos de regras:

```
bom_aluno(hugo):-assiste_aulas(hugo),
                 estuda_em_casa(hugo).

avo(A,N):-pai(A,P),pai(P,N).
```

Note que sempre grafamos símbolos e predicados com letras minúsculas e variáveis com letras maiúsculas.

SWI-Prolog

Comentários

Comentários em SWI-PROLOG são como em C, i.e., tudo quando estiver delimitado pelos caracteres `/*` e `*/` será considerado um comentário.

Também são admitidos comentários de linha que são introduzidos pelo caractere % e são fechados ao mudar de linha.

Exemplo de um Programa

```
nasceu(jose,m,eleazar,ana).
nasceu(sebastiao,m,eleazar,ana).
nasceu(neusa,f,zair,maria).
nasceu(paulo,m,zair,maria).
nasceu(andre,m,jose,neusa).
nasceu(zeluis,m,jose,neusa).
nasceu(marcoantonio,m,sebastiao,zulma).
nasceu(marcioaugusto,m,sebastiao,zulma).
nasceu(tieni,f,sebastiao,nilce).
nasceu(marcel,m,paulo,marialuiza).
nasceu(marcelo,m,paulo,marialuiza).
nasceu(marcioluis,m,paulo,marialuiza).
nasceu(hugo,m,andre,lilia).

morreu(zulma).

casou(eleazar, ana).
casou(zair, maria).
casou(jose,neusa).
casou(sebastiao,zulma).
casou(sebastiao,nilce).
casou(andre,lilia).

separou(andre,lilia).

pai(P,F):- nasceu(F,_,P,_).
mae(M,F):-nasceu(F,_,_,M).
genitor(G,F):-mae(G,F).
genitor(G,F):-pai(G,F).
avo(A,N):-pai(A,G),genitor(G,N).
```

Para carregar um Programa

Gravar o programa em um arquivo com extensão .PL. Supondo que a pasta Bin do SWI-Prolog foi colocada no Path e que seu programa foi gravado em um arquivo de nome Progrma.PL, para acionar o SWI-Prolog, deve-se dar o seguinte comando:

PlWin –f Programa.PL

Exemplo de Execução

```
?- pai(andre,hugo).
Yes

?- pai(jose,QUEM).

QUEM = andre ;
QUEM = zeluis ;
No

?- pai(P,F).
P = eleazar          F = jose ;
P = eleazar          F = sebastiao ;
P = zair             F = neusa ;
P = zair             F = paulo ;
P = jose             F = andre ;
P = jose             F = zeluis ;
P = sebastiao        F = marcoantonio ;
P = sebastiao        F = marcioaugusto ;
P = sebastiao        F = tieni ;
P = paulo            F = marcel ;
P = paulo            F = marcel ;
P = paulo            F = marcelo ;
P = paulo            F = marcioluis ;
P = andre            F = hugo ;
No

?- pai(andre,_).
Yes

?- pai (jose,QUEM),pai(QUEM,hugo).
QUEM = andre ;
No

?-
```

Predicados Predefinidos

Verificação de Tipo

1. **var**(TERMO**)**

   Sucede se TERMO for uma variável livre;

2. **nonvar**(TERMO**)**

   Sucede se TERMO não for uma variável livre;

3. **integer**(TERMO)

   Sucede se TERMO representar um número inteiro;

4. **float**(TERMO)

   Sucede se TERMO representar um número real;

5. **number**(TERMO)

   Sucede se TERMO representar um número (inteiro ou real);

6. **atom**(TERMO)

   Sucede se TERMO representar um símbolo;

7. **string**(TERMO)

   Sucede se TERMO representar uma cadeia de caracteres;

8. **atomic**(TERMO)

   Sucede se TERMO representar um símbolo, uma cadeia de caracteres, um número inteiro ou um número real;

9. **compound**(TERMO)

   Sucede se TERMO for composto, i.e., representar uma lista ou um functor;

10. **ground**(TERMO)

    Sucede se TERMO não estiver associado a uma variável livre.

Comparação e Unificação

Termos compostos são primeiramente verificados levando-se em conta, nesta ordem, (1) a quantidade de seus argumentos; (2) seu nome (alfabeticamente); (3) recursivamente, seus argumentos, da esquerda para a direita.

1. TERMO1 **==** TERMO2

   Sucede se TERMO1 for igual a TERMO2.

2. TERMO1 **\==** TERMO2

   Sucede se TERMO1 não for igual a TERMO2. Tem o mesmo significado que \TERMO1 == TERMO2.

3. TERMO1 **=** TERMO2

   Unifica TERMO1 com TERMO2. Sucede se a unificação tiver sucesso.

4. unify_with_occurs_check(TERMO1,TERMO2)

   Unifica TERMO1 com TERMO2, se assegurando que TERMO1 não ocorra em TERMO2. Sucede se a unificação tiver sucesso.

5. TERMO1 **\=** TERMO2

   Sucede se a unificação de TERMO1 com TERMO2 não for possível. Tem o mesmo significado que \TERMO1=TERMO2.

6. TERMO1 **=@=** TERMO2

   Sucede se TERMO1 for estruturalmente igual a TERMO2.

7. TERMO1 **\=@=** TERMO2

   Sucede se TERMO1 não for estruturalmente igual a TERMO2. Tem o mesmo significado que \TERMO1 **=@=** TERMO2.

8. TERMO1 **@<** TERMO2

   Sucede se TERMO1 for menor que TERMO2.

9. TERMO1 **@=<** TERMO2

   Sucede se TERMO1 for menor ou igual a TERMO2.

10. TERMO1 **@>** TERMO2

    Sucede se TERMO1 for maior que TERMO2.

11. TERMO1 **@>=** TERMO2

    Sucede se TERMO1 for maior ou igual a TERMO2.

Predicados de Controle

1. **fail**

   Sempre falha.

2. **true**

   Sempre sucede.

3. **repeat**

   Sempre sucede e prove um número infinito de pontos de escolha.

4. **!**

   Cut. Descarta pontos de escolha que o precedem na cláusula.

Operadores

Aritméticos

Em SWI-Prolog os operadores aritméticos são os seguintes: ****** ou ^ (potência), ***** (multiplicação), + (adição), - (menos unário), - (subtração), / (divisão), // (divisão inteira); **mod** (resto da divisão inteira).

De Bit

Em SWI-Prolog os operadores binários são os seguintes: /\ (and bit a bit), \/ (or bit a bit), **xor** (xor bit a bit), \ (not bit a bit), << (deslocamento de bits para a esquerda) e >> (deslocamento de bits para a direita).

Funções Matemáticas

1. **abs**: valor absoluto;
2. **acos**: inverso do cosseno;
3. **asin**: inverso do seno;
4. **atan**: inverso da tangente;
5. **ceil** ou **ceiling**: arredondamento para o próximo inteiro;
6. **cos**: cosseno;

7. **e**: constante de Neper;
8. **exp**: exponenciação (base e);
9. **float**: conversão explícita para real;
10. **float_fractional_part**: parte fracionária de um real;
11. **float_integer_part**: parte inteira de um real;
12. **floor**: arredondamento para o inteiro predecessor;
13. **integer** ou **round**: arredondamento para o inteiro mais próximo;
14. **log**: logarítmo natural;
15. **log10**: logarítmo de base 10;
16. **max**: máximo de dois números;
17. **min**: mínimo de dois números;
18. **random**: gera um número aleatório;
19. **rem**: resto da divisão inteira;
20. **truncate**: elimina a parte fracionária;
21. **pi**: constante pi;
22. **sin**: seno;
23. **sqrt**: raiz quadrada;
24. **tan**: tangente.

Functors

Functors são domínios que representam argumentos de predicados que são eles próprios predicados.

Exemplo de Programa

```
pagou(jose,comida(giovanetti,100.00)).
pagou(jose,comida(nacional,50.00)).
```

```
pagou(jose,aluguel(otot,1000.00)).
pagou(jose,loja(renner,70.00)).
pagou(jose,loja(skina,50.00)).

pagou(raul,comida(nacional,30.00)).
pagou(raul,comida(allesbier,100.00)).
pagou(raul,loja(skina,50.00)).

pagou(joao,comida(nacional,40.00)).
pagou(joao,comida(allesbier,70.00)).
pagou(joao,comida(nacional,50.00)).
pagou(joao,aluguel(serra,1000.00)).
pagou(joao,loja(americana,70.00)).
pagou(joao,loja(skina,30.00)).
```

Exemplo de Execução

```
?- pagou(jose,Oque).
Oque = comida(giovanetti,100.00) ;
Oque = comida(nacional,50.00) ;
Oque = aluguel(otot,1000.00) ;
Oque = loja(renner,70.00) ;
Oque = loja(skina,50.00) ;

No
?- pagou(Quem,comida(allesbier,Quanto)).
Quem = raul    Quanto = 100.00 ;
Quem = joao    Quanto = 70.00 ;

No
```

Listas

Listas são coleções de elementos e mesma natureza. Uma lista é representada por uma série de símbolos separados por vírgulas e entre colchetes. Veja o exemplo abaixo:

```
[maca, uva, pera, goiaba, abacaxi]
```

Para manipularmos uma lista, devemos promover uma unificação entre a lista e um padrão. Por exemplo, se desejássemos saber o primeiro elemento da lista acima e o que resta dela retirado o primeiro elemento faríamos:

```
[maca, uva, pera, goiaba, abacaxi] = [Primeiro | Resto]
```

e teríamos a variável `Primeiro` unificada com o símbolo maca e a variável `Resto` unificada com a lista `[uva, pera, goiaba, abacaxi]`.

Um outro exemplo seria se desejássemos saber os 3 primeiros elementos da lista acima e o que resta dela retirados os 3 primeiros elementos faríamos:

```
[maca, uva, pera, goiaba, abacaxi] = [P, S, T | Resto]
```

e teríamos a variável `P` unificada com o símbolo `maca`, a variável `S` unificada com o símbolo `uva`, a variável `T` unificada com o símbolo `pera` e a variável `Resto` unificada com a lista `[goiaba, abacaxi]`.

Vale ressaltar a possibilidade de fazermos em SWI-PROLOG listas de listas.

Exemplo de Programa 1

Considere o programa abaixo que implementa um predicado que recebe dois parâmetros, (1) e (2), ambos listas de números inteiros, sucedendo, se ambas as listas forem idênticas, e falhando, caso contrário.

```
identicas([],[]).
identicas([P1|R1],[P2|R2]):-P1==P2,identicas(R1,R2).
```

Exemplos de Execução

```
?- identicas([1,2,3],[1,2]).
No

?- identicas([1,2],[1,2,3]).
No

?- identicas([1,2,3],[1,2,4]).
No

?- identicas([1,2,3],[1,2,3]).
Yes
```

Algoritmo da Resolução aplicada ao último Exemplo de Execução

Reescrevendo o programa na forma de cláusulas de Horn:

identicas([],[])⊂
identicas([P1|R1],[P2|R2])⊂ P1=P2∩identicas(R1,R2)

Reescrevendo a pergunta na forma de cláusulas de Horn:

⊂identicas([1,2,3],[1,2,3])

O Algoritmo da Resolução faria uma série de manipulações lógicas, envolvendo a cláusula que exprime a pergunta e aquelas que constituem o banco de conhecimento. Veja abaixo as manipulações lógicas em questão:

Manipulação 1:
⊂identicas([1,2,3],[1,2,3])
identicas([],[])⊂
~~identicas([],[])⊂identicas([1,2,3],[1,2,3])~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, tendo em vista que [] e [1,2,3] são, irremediavelmente constantes com valores diferentes, desfaremos a Manipulação 1, voltanto a ter que verificar ⊂identicas([1,2,3],[1,2,3])

Manipulação 2:
⊂identicas([1,2,3],[1,2,3])
identicas([P1|R1],[P2|R2])⊂P1=P2∩identicas(R1,R2)
~~identicas([P1|R1],[P2|R2])⊂identicas([1,2,3],[1,2,3])~~∩P1=P2∩identicas(R1,R2)
Fazendo as instanciações P1\1, R1\[2,3], P2\1 e R2\[2,3], podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂1=1∩identicas([2,3],[2,3])

Manipulação 3:
⊂~~1=1~~∩identicas([2,3],[2,3])
Como o termo riscado representa uma verdade incontestável, podemos cancela-lo, restando verificar ⊂ identicas([2,3],[2,3])

Manipulação 4:
⊂ identicas([2,3],[2,3])
identicas([],[])⊂
~~identicas([],[])⊂identicas([2,3],[2,3])~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, tendo em vista que [] e [2,3] são, irremediavelmente constantes com valores diferentes, desfaremos a Manipulação 1, voltanto a ter que verificar ⊂identicas([2,3],[2,3])

Manipulação 5:
⊂identicas([2,3],[2,3])
identicas([P1|R1],[P2|R2])⊂P1=P2∩identicas(R1,R2)
~~identicas([P1|R1],[P2|R2])⊂identicas([2,3],[2,3])~~∩P1=P2∩identicas(R1,R2)
Fazendo as instanciações P1\2, R1\[3], P2\2 e R2\[3], podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂2=2∩identicas([3],[3])

Manipulação 6:
⊂~~2=2~~∩identicas([3],[3])

Como o termo riscado representa uma verdade incontestável, podemos cancela-lo, restando verificar ⊂ identicas([3],[3])

Manipulação 7:
⊂ identicas([3],[3])
identicas([],[])⊂
~~identicas([],[])⊂identicas([3],[3])~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, tendo em vista que [] e [3] são, irremediavelmente constantes com valores diferentes, desfaremos a Manipulação 1, voltanto a ter que verificar ⊂identicas([3],[3])

Manipulação 8:
⊂identicas([3],[3])
identicas([P1|R1],[P2|R2])⊂P1=P2∩identicas(R1,R2)
~~identicas([P1|R1],[P2|R2])⊂identicas([3],[3])~~∩P1=P2∩identicas(R1,R2)
Fazendo as instanciações P1\3, R1\[], P2\3 e R2\[], podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂3=3∩identicas([],[])

⊂ identicas([],[])
identicas([],[])⊂
~~identicas([],[])⊂identicas([],[])~~
Podemos cancelar os termos riscados, já que os mesmos são equivalentes, restando verificar ⊂

A cláusula constituida apenas por um símbolo de implicação (⊂) representa sucesso, i.e., a pergunta original tem resposta positiva.

Exemplo de Programa 2

Considere o programa abaixo que implementa um predicado que recebe uma lista de símbolos e um símbolo, e instancia uma variável com a lista que se obtém retirada a primeira ocorrência do dado símbolo da referida lista.

```
remove ([P|R],P,R).
remove ([P|R],E,[P|NR]):-P \== E,remove(R,E,NR).
```

Exemplo de Execução

```
?- remove([maca,uva,pera,uva,abacaxi],uva,S).
S = [maca, pêra, uva, abacaxi]

Yes
```

Algoritmo da Resolução aplicada ao Exemplo de Execução

Reescrevendo o programa na forma de cláusulas de Horn:

remove ([P|R],P,R)⊂

remove ([P|R],E,[P|NR])⊂P≠E∩remove(R,E,NR)

Reescrevendo a pergunta na forma de cláusulas de Horn:

⊂remove([maca,uva,pera,uva,abacaxi],uva,S)

O Algoritmo da Resolução faria uma série de manipulações lógicas, envolvendo a cláusula que exprime a pergunta e aquelas que constituem o banco de conhecimento.

Note que, tendo em vista que, em virtude das recursões, será, eventualmente, possível coexistirem em uma mesma cláusula, variáveis distintas, porém com o mesmo nome, oriundas de instâncias recursivas diferentes.

Para que não haja confusão a respeito destas variáveis, ao tomarmos do banco de conhecimento uma cláusula, numeraremos todas suas variáveis com o número da manipulação em curso.

Veja abaixo as manipulações lógicas em questão:

Manipulação 1:
⊂remove([maca,uva,pera,uva,abacaxi],uva,S)
remove ($[P_1|R_1]$,P_1,R_1)⊂
~~remove ($[P_1|R_1]$,P_1,R_1)⊂ remove([maca,uva,pera,uva,abacaxi],uva,S)~~
Como não existem instanciações capazes de possibilitar o cancelamento dos termos riscados, já que os mesmo jamais poderão se tornar equivalentes, tendo em vista que P_1\maca e P_1\uva são instanciações conflitantes, desfaremos a Manipulação 1, voltanto a ter que verificar ⊂remove([maca,uva,pera,uva,abacaxi],uva,S)

Manipulação 2:
⊂remove([maca,uva,pera,uva,abacaxi],uva,S)
remove ($[P_2|R_2]$,E_2,$[P_2|NR_2]$)⊂P_2≠E_2,remove(R_2,E_2,NR_2)
~~remove ($[P_2|R_2]$,E_2,$[P_2|NR_2]$)⊂ remove([maca,uva,pera,uva,abacaxi],uva,S)~~∩P_2≠E_2∩remove(R_2,E_2,NR_2).
Fazendo as instanciações P\maca, R\[uva,pera,uva,abacaxi] e E\uva, bem como a unificação S/[maca|NR_2], podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂maca≠uva∩remove([uva,pera,uva,abacaxi],uva,NR_2)

Manipulação 3:
⊂~~maca≠uva~~∩remove([uva,pera,uva,abacaxi],uva,NR_2)
Como o termo riscado representa uma verdade incontestável, podemos cancela-lo, restando verificar ⊂remove([uva,pera,uva,abacaxi],uva,NR)

Manipulação 4:
⊂remove([uva,pera,uva,abacaxi],uva,NR_2)
remove ($[P_4|R_4]$,P_4,R_4)⊂

~~remove ([P_4|R_4],P_4,R_4)⊂ remove([uva,pera,uva,abacaxi],uva,NR_2)~~

Fazendo as instanciações P\uva, R\[pera,uva,abacaxi] e NR_2\[pera,uva,abacaxi], podemos cancelar os termos riscados por te-los tornado equivalentes, restando verificar ⊂

A cláusula constituida apenas por um símbolo de implicação (⊂) representa sucesso, i.e., a pergunta original tem resposta.
Note que S\[maca|NR_2] e que NR_2\[pera,uva,abacaxi], logo, S\[maca,pera,uva,abacaxi], ficando assim removida a primeira ocorrência do símbolo uva.
Um exercício interessante pode ser modificar o programa para que sejam removidas todas as ocorrências do símbolo desejado, e não apenas a primeira.

Anexo I: Exercícios Propostos

Exercícios com Números

1. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número natural até 9; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com o símbolo que expressa por extenso o número recebido. O predicado deverá falhar, caso o número fornecido não esteja entre 0 e 9.

2. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número real que expressa uma temperatura dada em graus Fahrenheit; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em graus Célsius, a temperatura dada. A relação entre graus Celsius e graus Fahrenheit é C = 5/9 (F -32).

3. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número real que expressa uma temperatura dada em graus Kelvin; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em graus Célsius, a temperatura dada. A relação entre graus Celsius e graus Kelvin é C = K – 275,15.

4. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número real que expressa uma temperatura dada em graus Rankine; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em graus Célsius, a temperatura dada. A relação entre graus Celsius e graus Rankine é C = (R/1.8) – 275,15.

5. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número real que expressa, em centímetros, o tamanho da base (B) de um triângulo; (2) um número real que expressa, em centímetros, a altura (A) do supra referido triângulo; e (3) uma variável não instanciada. O predicado deverá instanciar a

variável com um número real que expressa, em centímetros quadrados, a área do triângulo em questão. A relação entre essas grandezas é Area = (B * A) / 2.

6. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número real que expressa, em centímetros, o tamanho (L) dos lados de um quadrado; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do retângulo em questão. A relação entre essas grandezas é Area = L^2.

7. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número real que expressa, em centímetros, o tamanho (m) do lado menor de um retângulo; (2) um número real que expressa, em centímetros, o tamanho (M) do lado maior do supra referido retângulo; e (3) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do retângulo em questão. A relação entre essas grandezas é Area = m . M

8. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número real que expressa, em centímetros, o tamanho da diagonal menor (d) de um losângo; (2) um número real que expressa, em centímetros, o tamanho da diagonal maior (D) do supra referido losângo; e (3) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do losângo em questão. A relação entre essas grandezas é Area = (d . D) / 2.

9. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número real que expressa, em centímetros, o tamanho da base (B) de um paralelogramo; (2) um número real que expressa, em centímetros, a altura (A) do supra referido paralelogramo; e (3) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do paralelogramo em questão. A relação entre essas grandezas é Area = B . A.

10. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: (1) um número real que expressa, em centímetros, o tamanho da base menor

(b) de um trapézio; (2) um número real que expressa, em centímetros, o tamanho da base maior (B) do supra referido trapézio; (3) um número real que expressa, em centímetros, a altura do supra referido trapézio; e (4) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do trapézio em questão. A relação entre essas grandezas é Area = ((b+B) . A) / 2.

11. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: (1) um número natural que expressa a quantidade de lados de um polígono regular; (2) um número real que expressa, em centímetros, o tamanho da base (B) do supra referido polígono regular; (3) um número real que expressa, em centímetros, o tamanho da apótema (A) do supra referido polígono regular medida em centímetros de sua base (B) e de sua apótema (A), ou seja, a reta imaginária que une seu centro ao meio de sua base; e (4) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do polígono regular em questão. A relação entre essas grandezas é Area A relação entre essas grandezas é Area = (Q.B.A)/2.

12. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número real que expressa, em centímetros, o tamanho (R) do raio de um círculo; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, em centímetros quadrados, a área do círculo em questão. A relação entre essas grandezas é Area = $\pi.R^2$, sendo π constante e aproximadamente igual a 3,1415.

13. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número real que expressa, em kilogramas, o peso de uma certa pessoa; (2) um número real que expressa, em centímetros, a altura da supra referida pessoa; e (3) uma variável não instanciada. O predicado deverá instanciar a variável com um número real que expressa, o Índice de Massa Corpórea (ou BMI, Body Mass Index) da supra referida pessoa.

14. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) um número natural; e (2) uma variável não instanciada. O predicado deverá instanciar a variável com a soma dos divisores do número natural dado.

15. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número natural que expressa a hora de um horário; (2) um número natural que expressa os minutos do supra referido horário; e (3) um número natural que expressa os segundos do supra referido horário. O predicado deve suceder, caso os parâmetros fornecidos representem um horário válido, e falhar, caso contrário.

16. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número natural que expressa o dia de uma data; (2) um número natural que expressa o mês da supra referida data; (3) um número natural que expressa o ano da supra referida data. O predicado deve suceder, caso os parâmetros fornecidos representem uma data válida, e falhar, caso contrário. Lembrar dos meses de 30 e 31 dias, bem como do mês de fevereiro e seus 28 ou 29 dias. Anos bissextos são anos divisíveis por 4 mas não por 100, ou então anos divisíveis por 400.

17. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: as medidas em centímetro de três retas. O predicado deve suceder, caso retas com as medidas fornecidas possam formar um triângulo, e falhar, caso contrário.

18. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: as medidas em centímetro de três retas e uma variável não instanciada. Quando não for possível formar um triângulo com as três retas, o predicado deve falhar; quando for possível formar um triângulo com as três retas, o predicado deve instanciar a variável em questão com um dos seguintes simbolos: EQUILATERO (no caso das três retas terem a mesma medida), ISOSCELES (no caso de duas delas terem a mesma medida, porém diferente da medida da terceira) ou ESCALENO (no caso das três retas terem medidas diferentes).

19. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: as medidas em graus de três ângulo. O predicado deve suceder, caso seja possível um triângulo ter tais ângulos internos, e falhar, caso contrário.

20. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: as medidas em em graus de três ângulos e uma variável não instanciada. Quando não for possível um triângulo ter tais ângulos internos, o predicado deve falhar;

quando for possível um triângulo ter tais ângulos internos, o predicado deve instanciar a variável em questão com um dos seguintes simbolos: ACUTANGULO (no caso dos três ângulos serem agudos, ou seja, menores que 90 graus), RETANGULO (no caso de um dos ângulos medir 90 graus) ou OBTUSANGULO (no caso de um dos ângulos ser obtuso, ou seja, medir mais do que 90 graus).

21. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: os coeficientes **a** e **b** de uma equação de primeiro grau e uma variável não instanciada. O predicado deve instanciar a variável em questão com a raiz da da equação.

22. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: os coeficientes **a**, **b** e **c** de uma equação de segundo grau e uma variável não instanciada. medidas em em graus de três ângulos e uma variável não instanciada. O predicado deve instanciar a variável com a raiz da equação, calculada conforme se calcula raiz de equação de primeiro grau, quando **a** for igual a zero, com a raiz da equação calculada pelo método de Baskara, quando delta for igual a zero, indicando que há apenas uma raiz, ou com as raizes da equação, também calculadas pelo método de Baskara (fornece uma delas e, quando for perguntado se há mais respostas, fornece a outra delas), quando o delta for positivo, indicando que há duas raizes distintas. Seu predicado deve falhar, quando delta for menor que zero, indicando que aquela equação não tem raizes. Teste seu programa com 0x2+2x+3=0 (não é equação de 2º grau), 1x2 4x+5=0 (não tem raízes reais), 4x2-4x+1=0 (tem apenas uma raiz) e 1x2-5x+6=0 (tem duas raízes).

23. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros, dois números inteiros e uma variável não instanciada. O predicado deverá instanciar a variável em questão com a soma dos números inteiros compreendidos no intervalo, incluindo os limites (no caso dos limites sejam iguais, com um dos limites). O predicado deverá falhar, caso o primeiro número fornecido seja maior do que o segundo.

24. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros, (1) um número natural; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o fatorial do número recebido, ou falhar, caso o número fornecido seja negativo.

25. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros, dois números naturais; e uma variável não instanciada. O predicado deverá instanciar a variável em questão com o MDC dos dois números recebidos, ou falhar, caso um dos números fornecidos seja negativo.

    Saiba que $mdc(X,Y) = \begin{cases} Y, \text{ se resto } (X / Y) = 0; \\ mdc\ (Y, \text{ resto } (X / Y)), \text{ cc.} \end{cases}$

26. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros, (1) um número natural; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a soma dos divisores do número recebido, ou falhar, caso o número fornecido seja negativo.

27. Escreva um programa SWI-Prolog que implementa um predicado que recebe como parâmetro um número natural. O predicado deverá suceder, caso o número fornecido seja primo, ou falhar, caso contrário (ou caso o número fornecido seja negativo).

28. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros, dois números naturais e uma variável não instanciada. O predicado deverá instanciar a variável em questão com a soma dos números primos compreendidos no intervalo, incluindo os limites (ou com um dos limites, caso os limites sejam iguais e sejam números primos). O predicado deverá falhar, caso o primeiro número fornecido seja maior do que o segundo ou se algum deles for negativo.

29. Escreva um programa SWI-Prolog que implementa um predicado que recebe como parâmetro um número natural. O predicado deverá suceder, caso o número fornecido seja perfeito, ou falhar, caso contrário (ou caso o número fornecido seja negativo). Perfeitos são os números iguais à soma de seus divisores distintos de si próprio (6 é; 10 não é).

30. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros, dois números naturais e uma variável não instanciada. O predicado deverá instanciar a variável em questão com a soma dos números perfeitos compreendidos no intervalo, incluindo os limites (ou com um dos limites, caso os limites sejam iguais e sejam números perfeitos). O predicado deverá falhar, caso o primeiro número fornecido seja maior do que o segundo ou se algum deles for negativo.

31. Escreva um programa SWI-Prolog que implementa um predicado que recebe como parâmetro dois números naturais. O predicado deverá suceder, caso os números fornecidos sejam amigos, ou falhar, caso contrário (ou caso algum dos números fornecidos seja negativo). Perfeitos são os números iguais à soma de seus divisores distintos de si próprio (6 é; 10 não é). Amigos são os números que têm a soma de seus divisores distintos de si próprio igual ao outro (220 e 284 são; 123 e 321 não são).

32. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros, dois números naturais e uma variável não instanciada. O predicado deverá instanciar a variável em questão com a soma dos pares de números amigos compreendidos no intervalo, incluindo os limites. O predicado deverá falhar, caso o primeiro número fornecido seja maior do que o segundo ou se algum deles for negativo.

33. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros, (1) um número natural; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número natural que dem os dígitos na ordem inversa àquela do número fornecido. Pode ser que seja necessário implementar algum predicado auxiliar.

34. Escreva um programa SWI-Prolog que implementa um predicado que recebe como parâmetro um número natural. O predicado deverá suceder, caso o número recebido seja palíndromo (mesma leitura da esquerda para a direita e da direita para a esquerda), ou falhar, caso contrário.

Exercícios sobre Listas

35. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a quantidade de elementos da lista dada.

36. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista;; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o primeiro elemento da lista dada.

37. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista;; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o último elemento da lista dada.

38. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um número de ordem (1, 2, 3, etc) ao qual chamaremos de N; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o N^{ézimo} elemento da lista dada.

39. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (2) um possível elemento da lista dada. O predicado deverá suceder, no caso do possível elemento dado, de fato, ser elemento da lista dada. O predicado deverá falhar, caso contrário.

40. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento da lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a quantidade de vezes que o possível elemento dado ocorre como elemento da lista dada.

41. Usando o predicado implementado no exercício anterior, escreva programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (2) um possível elemento da lista dada. O predicado deverá suceder, no caso do possível elemento dado, de fato, ser elemento da lista dada. O predicado deverá falhar, caso contrário.

42. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inclusão do possível elemento dado como primeiro elemento da lista dada.

43. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inclusão do possível elemento dado como último elemento da lista dada.

44. Escreva um programa em SWI-Prolog que implementa um predicado que recebe quatro parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; (3) um número de ordem (1, 2, 3, etc) ao qual chamaremos de N; e (4) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inclusão do possível elemento dado como N^{ézimo} elemento da lista dada.

45. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção do elemento mais à esquerda (primeiro elemento) da lista dada.

46. Escreva um programa em SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção do elemento mais à direita (último elemento) da lista dada.

47. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um número de ordem (1, 2, 3, etc) ao qual chamaremos de N; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção do N^{ézimo} elemento da lista dada.

48. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção da primeira ocorrência do referido possível elemento da lista dada.

49. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção da última ocorrência do referido possível elemento da lista dada.

50. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) uma lista; (2) um possível elemento para a lista dada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção de todas as ocorrências do referido possível elemento da lista dada.

51. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros: os parâmetros (1) e (2) serão listas; e o parâmetro (3) será uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da concatenação das duas listas dadas.

52. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inversão da lista dada.

Exercícios sobre Listas de Números

53. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um número; (2) uma lista de números; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inclusão ordenada do referido número na lista de números dada.

54. Usando o predicado implementado no exercício anterior, escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da ordenação da lista de números dada.

55. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o menor número da lista de números dada.

56. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da primeira ocorrência do menor número da lista de números dada.

57. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da última ocorrência do menor número da lista de números dada.

58. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada de todas as ocorrências do menor número da lista de números dada.

59. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o maior número da lista de números dada.

60. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da primeira ocorrência do maior número da lista de números dada.

61. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da última ocorrência do maior número da lista de números dada.

62. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada de todas as ocorrências do maior número da lista de números dada.

63. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número que representa a soma dos números presentes na lista de números dada.

64. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número que representa a média aritmética dos números presentes na lista de números dada.

Exercícios sobre Listas Generalizadas

65. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de listas de elementos atômicos; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com uma lista contendo as quantidades de elementos de cada uma das listas contidas na lista recebida.

66. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com uma lista de elementos atômicos contendo os elementos atômicos contidos direta ou indiretamente na lista recebida.

67. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) um valor atômico. O predicado deverá suceder caso a lista contenha, direta ou indiretamente, o valor atômico dado.

68. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) um valor, não necessariamente atômico. O predicado deverá suceder caso a lista contenha, direta ou indiretamente, o valor dado.

69. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com um número inteiro que expressa o MENOR nível de encaixamento presente na lista dada (1, se a lista contem ao menos um elemento aômico; 2, se a lista contem ao menos um elemento, cujos elementos sejam todos atômicos; 3, se a lista contem ao menos um elemento que seja uma lista de elementos atômicos; e assim por diante).

70. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com um número inteiro que expressa o MAIOR nível de encaixamento presente na lista dada (1, se a lista contem ao menos um elemento aômico; 2, se a lista contem ao menos um elemento, cujos elementos sejam todos atômicos; 3, se a lista contem ao menos um elemento que seja uma lista de elementos atômicos; e assim por diante).

71. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o elemento atômico mais à esquerda da referida lista.

72. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem ao retirar da lista dada seu elemento atômico mais à esquerda.

73. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o elemento atômico mais à direita da referida lista.

74. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem ao retirar da lista dada seu elemento atômico mais à direita.

75. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um valor atômico; (2) uma lista generalizada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção da primeira ocorrência (seja no nível que for) do referido valor atômico da lista generalizada dada.

76. Escreva um programa em SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um valor atômico; (2) uma lista generalizada; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da remoção de todas as ocorrências (seja no nível que for) do referido valor atômico da lista generalizada dada.

77. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada de números; e (2) uma variável não instanciada. O

predicado deverá instanciar a variável em questão com o maior número (seja no nível que for) da lista generalizada dada.

78. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada do maior número da lista generalizada dada, seja no nível que for.

79. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista generalizada de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número que representa a soma dos elementos (seja no nível que for) da lista de números dada.

80. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de números; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número real que representa a média aritmética dos elementos (seja no nível que for) da lista de números dada.

Exercícios sobre Listas de Functors

Em todos os exercícios abaixo, toda menção a functors deve ser entendida como fazendo referência a functors que representam os resultados acadêmicos de uma disciplina e que são assim constituídos: **disciplina(nm,fd,nd)**, sendo **nm** o nome da matéria, **fd** a frequência na disciplina e **nd** a nota na disciplina.

81. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o nome da matéria com maior nota da lista de functors dada.

82. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número que representa a soma das notas da lista de functors dada.

83. Escreva um programa SWI-Prolog que implementa um predicado que recebe três parâmetros: (1) um functor; (2) uma lista de functors ordenada em ordem crescente de nota; e (3) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da inclusão ordenada do referido functor na lista de functors dada.

84. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de fuctors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da ordenação, em ordem crescente de nota, da lista de functors dada.

85. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de fuctors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da ordenação, em ordem decrescente de frequência, da lista de functors dada.

86. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o nome da matéria com menor frequência da lista de functors dada.

87. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da disciplina de maior nota da lista de functors dada.

88. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com a lista que se obtem da retirada da disciplina de menor frequência da lista de functors dada.

89. Escreva um programa SWI-Prolog que implementa um predicado que recebe dois parâmetros: (1) uma lista de functors; e (2) uma variável não instanciada. O predicado deverá instanciar a variável em questão com o número real que representa a média aritmética das notas da lista de functors dada.

Anexo II: Referências

[Sebesta89] Sebesta, R.W., "Concepts of Programming Languages", The Benjamin/Cummings Publishing Company, Inc., 1989.

[Baranauskas93] Baranauskas, M.C.C., "Procedimento, Função, Objeto ou Lógica? Linguagens de Programação vistas pelos seus Paradígmas" *in* Computadores e Conhecimento - Repensando a Educação, Valente, J.A. (Org.), NIED, UNICAMP, 1993.