Inatel

M020 – Matemática Discreta

1. LÓGICA FORMAL

1.5 Programação Lógica

Marcelo Vinícius Cysneiros Aragão

marcelovca90@inatel.br

Programação Lógica

- As linguagens de programação com as quais você, provavelmente, está familiarizado, tais como C++ e Java, são conhecidas como **linguagens procedurais**.
- A maior parte do conteúdo de um programa escrito em uma linguagem procedural consiste em **instruções para executar o algoritmo** que o programador acredita que resolverá o problema em consideração.
- O programador, portanto, está dizendo para o computador como resolver o problema passo a passo.

Programação Lógica

- Algumas linguagens de programação, ao invés de procedurais, são **linguagens declarativas** ou **linguagens descritivas**.
- Uma linguagem declarativa baseia-se na lógica de predicados; uma tal linguagem já vem equipada com suas próprias regras de inferência.
- Um programa escrito em uma linguagem declarativa contém apenas proposições – de fato, **fbfs predicadas** – que são declaradas como **hipóteses**.
- A execução de um programa declarativo permite ao usuário colocar perguntas, procurando informação sobre conclusões possíveis dedutíveis das hipóteses.

Programação Lógica

- Após obter a pergunta do usuário, o programa liga sua "**máquina de inferência**" e aplica suas regras de inferência às hipóteses para ver quais das conclusões respondem à pergunta do usuário.
- O programa, lembre-se, contém apenas as hipóteses, e não apenas alguma instrução explícita sobre que passos fazer e em que ordem. Podemos dizer que o mecanismo de inferência age por trás do pano para construir uma sequência de demonstração.
- É a **natureza mecânica** da aplicação das regras de inferência que torna essas "demonstrações automáticas de teoremas" possíveis.

Prolog

- A linguagem de programação Prolog, abreviação de PROgramming in LOGic, é uma linguagem de programação declarativa. O conjunto de declarações que constituem um programa em Prolog é também conhecido como um **banco de dados Prolog**.
- Os itens em um banco de dados Prolog tem uma entre duas formas, conhecidas em Prolog como ***fatos*** e ***regras***.
- As regras de Prolog, no entanto, são apenas outra espécie de fatos e não devem ser confundidas com regras de inferência.

Prolog

- Os **fatos de Prolog** permitem a definição de predicados através da declaração de quais itens pertencentes a algum conjunto universo satisfazem os predicados.
- Como exemplo, suponha que queremos criar um programa em Prolog que descreva as cadeias alimentares de uma determinada região ecológica.
- Poderíamos começar com um predicado binário *come*. Descreveríamos, então, o predicado fornecendo os pares de elementos no domínio que tornam *come* verdadeiro.
- Poderíamos ter, então, os fatos a seguir no nosso banco de dados:

come(urso, peixe), *come*(urso, raposa) e *come*(veado, grama)

Prolog

- Os detalhes precisos de uma proposição em Prolog variam de uma implementação para outra, de modo que estamos vendo apenas o espírito da linguagem, usando um pseudocódigo semelhante à linguagem Prolog.
- Nesse exemplo, “urso”, “peixe”, “raposa”, “veado” e “grama” são **constantes**, pois representam elementos específicos do conjunto universo.
- Como o domínio propriamente dito nunca é especificado exceto através dos predicados, neste ponto poderíamos considerar o conjunto universo como os elementos citados anteriormente.

Prolog

- É responsabilidade do usuário manter compreensão e utilização consistentes dos predicados em um programa Prolog.
- Assim, *come*(urso, peixe) poderia ser usado para representar o fato de que ursos comem peixes ou de que peixes comem ursos! Vamos impor a **convenção** de que *come*(*x,y*) significa que "*x* come *y*".
- Poderíamos adicionar a descrição de dois predicados unários, *animal* e *planta*, ao banco de dados colocando os fatos:

  animal(urso), *animal*(peixe), *animal*(raposa), *animal*(veado) e *planta*(grama)

- Com esse programa (banco de dados) em Prolog, podemos colocar algumas perguntas simples.

Banco de dados:
come(urso, peixe)
come(urso, raposa)
come(veado, grama)
animal(urso)
animal(peixe)
animal(raposa)
animal(veado)
planta(grama)

Exemplo 37

A pergunta

is(*animal*(urso))

simplesmente questiona se o fato *animal*(urso) pertence ao banco de dados.[14] Como esse fato está no banco de dados, o programa responderia à pergunta respondendo que sim. (Essa é uma seqüência de demonstração de um passo — não há necessidade de nenhuma regra de inferência.) Outro diálogo com o programa poderia ser

is(*come*(veado, grama))
yes
is(*come*(urso, coelho))
no

Exemplo 38

Banco de dados:
come(urso, peixe)
come(urso, raposa)
come(veado, grama)
animal(urso)
animal(peixe)
animal(raposa)
animal(veado)
planta(grama)

A pergunta

which(x: *come*(urso, x))

Gera a resposta

peixe
raposa

O programa em Prolog respondeu à pergunta buscando no banco de dados todos os fatos da forma *come*(urso, x), onde x é uma variável. A resposta "peixe" é dada primeiro porque a busca é feita ordenadamente, de cima para baixo. ❖

Problema Prático 28

Dado o banco de dados

come(urso, peixe)
come(urso, raposa)
come(veado, grama)
animal(urso)
animal(peixe)
animal(raposa)
animal(veado)
planta(grama)

diga qual vai ser a resposta do programa à pergunta

which(x: *come*(x, y) **and** *planta*(y))

28. veado (veado come grama)

Prolog

- Observações:
  - O exemplo 37 mostra um questionamento simples.
  - O exemplo 38 mostra que perguntas podem incluir variáveis.
  - O problema prático 28 mostra que perguntas podem conter os conectivos lógicos **and** (e), **or** (ou) e **not** (não).

Banco de dados:
come(urso, peixe)
come(urso, raposa)
come(veado, grama)
animal(urso)
animal(peixe)
animal(raposa)
animal(veado)
planta(grama)

Prolog

- O segundo tipo de item em um programa em Prolog é uma **regra de Prolog**.
- Uma regra é uma descrição de um predicado através de um condicional.
- Por exemplo, poderíamos usar uma regra para definir um predicado para *presa*:

presa(*x*) **if** *come*(*y,x*) **and** *animal*(*x*)

- Isso diz que x é uma presa se x é um animal que é comido.
- Se adicionarmos essa regra ao nosso banco de dados, então a resposta à pergunta **which**(*x* : *presa*(*x*)) seria “peixe, raposa”.

Cláusulas de Horn e Resolução

- Como os fatos e as regras do Prolog se relacionam com a lógica de predicados mais formalmente?
- Podemos descrever os fatos em nosso banco de dados pelas fbfs

$C(u, p)$
$C(u, r)$
$C(v, g)$
$A(u)$
$A(p)$
$A(r)$
$A(v)$
$P(g)$

- Uma **cláusula de Horn** é uma fbf composta de um ou mais predicados ou de negação(ões) de predicado(s) (tendo variáveis ou constantes como argumentos) conectada(s) por disjunções, de tal forma que no máximo um predicado não esteja negado.

e a regra pela fbf

$$C(y, x) \wedge A(x) \rightarrow Pr(x)$$

Cláusulas de Horn e Resolução

- **Quantificadores universais** não fazem parte explícita da regra como ela aparece em um programa em Prolog, mas a linguagem trata a regra como se estivesse universalmente quantificada.

$$(\forall y)(\forall x)[C(y,x) \wedge A(x) \rightarrow Pr(x)]$$

- A linguagem usa, repetidamente, a **particularização universal** para retirar os quantificadores universais e permitir às variáveis assumir todos os valores do conjunto universo.

Cláusulas de Horn e Resolução

- Tanto os fatos quanto as regras são exemplos de cláusulas de Horn.
- Uma **cláusula de Horn** é uma fbf composta de um ou mais predicados ou de negação(ões) de predicado(s) (tendo variáveis ou constantes como argumentos) conectada(s) por disjunções, de tal forma que no máximo um predicado não esteja negado.
- Assim, o fato $C(v, g)$ é um exemplo de cláusula de Horn, pois consiste em um único predicado não-negado.

Cláusulas de Horn e Resolução

- A fbf $[C(y,x)]' \vee [A(x)]' \vee Pr(x)$ é um exemplo de uma cláusula de Horn, já que consiste em três predicados conectados por disjunções com apenas $Pr(x)$ não-negado.
- Pelas leis de De Morgan, ela é equivalente a $[C(y,x) \wedge A(x)]' \vee Pr(x)$ que, por sua vez, é equivalente a $C(y,x) \wedge A(x) \rightarrow Pr(x)$ e, portanto, representa a regra em nosso programa em Prolog.

Cláusulas de Horn e Resolução

- A regra de inferência usada pelo Prolog é chamada de **resolução**.
- Duas cláusulas de Horn em um banco de dados Prolog são resolvidas em uma nova cláusula de Horn se uma delas contiver um predicado não-negado que corresponda a um predicado negado na outra.
- A nova cláusula elimina o termo correspondente e fica, então, disponível para uso em resposta às perguntas.

Cláusulas de Horn e Resolução

Duas cláusulas de Horn em um banco de dados Prolog são resolvidas em uma nova cláusula de Horn se uma delas contiver um predicado não-negado que corresponda a um predicado negado na outra.

- Por exemplo,

$$A(a)$$

$$[A(a)]' \vee B(b)$$

é resolvida para $B(b)$. Isso significa que, a partir de

$$A(a), [A(a)]' \vee B(b)$$

que é equivalente a

$$A(a), A(a) \rightarrow B(b)$$

o Prolog infere que $B(b)$ o que é, simplesmente, uma aplicação do modus ponens.

- Portanto, a regra de inferência do Prolog inclui o modus ponens como um caso particular.

Cláusulas de Horn e Resolução

- Ao aplicar a regra de resolução, as variáveis são consideradas "correspondentes" a qualquer símbolo constante. (Como já dito, essa é uma aplicação repetida da particularização universal.)
- Em qualquer nova cláusula resultante, as variáveis são substituídas pelas suas constantes associadas de maneira coerente.

Cláusulas de Horn e Resolução

- Assim, em resposta a uma pergunta “quais x são presas?”, Prolog busca no banco de dados uma regra que tenha o predicado desejado $Pr(x)$ como consequente.
- Encontra

$$[C(y,x)]' \vee [A(x)]' \vee Pr(x)$$

- Procura, então, no banco de dados por outras cláusulas que podem ser resolvidas com essa. A primeira delas é o fato $C(u,p)$. Essas duas cláusulas se resolvem em

$$[A(p)]' \vee Pr(p)$$

- Note que a constante p substituiu x em todos os lugares.

Cláusulas de Horn e Resolução

- Usando essa nova cláusula, ela pode ser resolvida junto com o fato $A(p)$ para se obter $Pr(p)$.
- Tendo obtido todas as resoluções possíveis do fato $C(u, p)$, Prolog volta para trás, procurando uma nova cláusula que possa ser resolvida com a regra; dessa vez, encontraria $C(u, r)$.

Cláusulas de Horn e Resolução

$C(u,p)$
$C(u,r)$
$C(v,g)$
$A(u)$
$A(p)$
$A(r)$
$A(v)$
$P(g)$

Como um exemplo mais sofisticado de resolução, suponha que adicionamos a regra

caçado(x) **if** *presa*(x)

ao banco de dados. Essa regra em forma simbólica[15] é

$$[Pr(x)] \rightarrow H(x)$$

ou, como uma cláusula de Horn,

$$[Pr(x)]' \vee H(x)$$

Ela se resolve com a regra que define presa

$$[C(y, x)]' \vee [A(x)]' \vee Pr(x)$$

para dar a nova regra

$$[C(y, x)]' \vee [A(x)]' \vee H(x)$$

A pergunta

which(x: *caçado*(x))

vai usar essa nova regra para concluir

peixe
raposa

Exemplo 39

Suponha que um banco de dados Prolog contenha as seguintes informações:

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(*x*) **if** *come*(*y*, *x*) **and** *animal*(*x*)

Poderíamos, então, ter o seguinte diálogo:

is(*animal*(coelho))

yes

is(*come*(lince, grama))

no

which(*x*: *come*(*x*, peixe))

urso
guaxinim

which(*x*, *y*: *come*(*x*, *y*) **and** *planta*(*y*))

peixinho alga
coelho grama
veado grama

which(*x*: *presa*(*x*))

peixe
peixinho
peixe
guaxinim
raposa
coelho
veado
veado

Note que o peixe é listado duas vezes na resposta à última pergunta, pois os peixes são comidos por ursos (fato 1) e por guaxinins (fato 3). Analogamente, veados são comidos por ursos e por linces. ❖

Problema Prático 29

a. Formule uma regra de Prolog que define o predicado *predador*.
b. Adicione essa regra ao banco de dados do Exemplo 39 e diga qual seria a resposta à pergunta

which(*x*: *predador*(*x*))

29. a. *predador*(*x*) **if** *come*(*x*, *y*) **and** *animal*(*y*)

b. urso
peixe
guaxinim
urso
urso
raposa
urso
lince

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(*x*) **if** *come*(*y*, *x*) **and** *animal*(*x*)

Recorrência

- As regras do Prolog são condicionais. Os antecedentes podem depender dos fatos, como em

$$presa(x) \textbf{ if } come(y,x) \textbf{ and } animal(x)$$

- ou de outras regras, como em

$$caçado(x) \textbf{ if } presa(x)$$

- O antecedente de uma regra pode, também, depender da própria regra, de modo que a regra é definida em termos de si mesma.
- Uma definição na qual o item sendo definido é ele próprio é chamada de uma **definição recorrente**.

Recorrência

- Como exemplo, suponha que queremos usar o banco de dados ecológicos do Exemplo 39 para estudar cadeias alimentares.
- Podemos, então, definir uma relação binária *na-cadeia-alimentar*(*x,y*) que significa que "*y* pertence à cadeia alimentar de *x*".
- Isso, por sua vez, significa uma entre duas coisas:
  1. *x* come *y* diretamente
  2. *x* come alguma coisa que come alguma coisa que come alguma coisa ... que come *y*

Recorrência

- O caso 2 pode ser reformulado da seguinte maneira:

  2’. *x* come *z* e *y* pertence à cadeia alimentar de *z*

- O caso 1 é simples de testar dos fatos existentes mas, sem (2’), *na-cadeia-alimentar* não significaria nada diferente de *come*.

- Por outro lado, (2’) sem (1) nos remete a um caminho de comprimento infinito de alguma coisa comendo alguma coisa comendo alguma coisa e assim por diante, sem nada a nos dizer quando parar.

Recorrência

- Definições recorrentes precisam de informação específica de quando parar.
- A regra de Prolog para *na-cadeia-alimentar* incorpora (1) e (2’):
  - *na-cadeia-alimentar*(*x,y*) **if** *come*(*x,y*)
  - *na-cadeia-alimentar*(*x,y*) **if** *come*(*x,z*) **and** *na-cadeia-alimentar*(*z,y*)
- Essa é uma regra recorrente porque define o predicado *na-cadeia-alimentar* em termos de *na-cadeia-alimentar*.

Recorrência

- Uma regra recorrente é necessária quando a propriedade sendo descrita pode passar de um objeto para o próximo.
- O predicado *na-cadeia-alimentar* (*nca*) tem essa propriedade:

$$nca(x, y) \wedge nca(y, z) \rightarrow nca(x, z)$$

Exemplo 40

Após a inclusão da regra na-cadeia-alimentar ao banco de dados do Exemplo 39, é feita a seguinte pergunta:

which(*y*: *na-cadeia-alimentar*(urso, *y*))

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(*x*) **if** *come*(*y*, *x*) **and** *animal*(*x*)

Exemplo 40

Após a inclusão da regra na-cadeia-alimentar ao banco de dados do Exemplo 39, é feita a seguinte pergunta:

which(*y*: *na-cadeia-alimentar*(urso, *y*))

A resposta é a seguinte (os números foram colocados para referência):

1. peixe
2. guaxinim
3. raposa
4. veado
5. peixinho
6. alga
7. peixe
8. peixinho
9. alga
10. coelho
11. grama
12. grama

Prolog aplica primeiro o caso simples

na-cadeia-alimentar(urso, *y*) **if** *come*(urso, *y*)

obtendo as respostas de 1 a 4 diretamente dos fatos *come*(urso, peixe), *come*(urso, guaxinim) e assim por diante.

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(*x*) **if** *come*(*y*, *x*) **and** *animal*(*x*)

Exemplo 40

Passando para o caso recorrente

na-cadeia-alimentar(urso, *y*) **if** *come*(urso, *z*) **and** *na-cadeia-alimentar*(*z*, *y*)

uma correspondência para *come*(urso, *z*) ocorre com *z* igual a "peixe". Prolog procura, então, todas as soluções para a relação *na-cadeia-alimentar*(peixe, *y*). Usando primeiro o fato simples de *na-cadeia-alimentar*, ocorre uma correspondência com o fato *come*(peixe, peixinho). Isso resulta na resposta 5, peixinho. Como não existem outros fatos da forma *come*(peixe, *y*), a próxima coisa a ser tentada é o caso recorrente de *na-cadeia-alimentar*(peixe, *y*):

na-cadeia-alimentar(peixe, *y*) **if** *come*(peixe, *z*) **and** *na-cadeia-alimentar*(*z*, *y*)

Uma correspondência para *come*(peixe, *z*) ocorre para *z* igual a "peixinho". Prolog procura, então, todas as soluções para a relação *na-cadeia-alimentar*(peixinho, *y*). Usando primeiro o fato simples de *na-cadeia-alimentar*, ocorre uma correspondência com o fato *come*(peixinho, alga). Isso resulta na resposta 6, alga. Como não existem outros fatos da forma *come*(peixinho, *y*), a próxima coisa a ser tentada é o caso recorrente de *na-cadeia-alimentar*(peixinho, *y*):

na-cadeia-alimentar(peixinho, *y*) **if** *come*(peixinho, *z*) **and** *na-cadeia-alimentar*(*z*, *y*)

Uma correspondência para *come*(peixinho, *z*) ocorre com *z* igual a "alga". Prolog procura, então, todas as soluções para a relação *na-cadeia-alimentar*(alga, *y*). Uma pesquisa em todo o banco de dados não revela nenhum fato da forma *come*(alga, *y*) (ou *come*(alga, *z*)), de modo que nem o caso simples, nem o recorrente pode ser usado.

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(*x*) **if** *come*(*y*, *x*) **and** *animal*(*x*)

Exemplo 40

A Fig. 1.2 ilustra a situação neste ponto. Prolog chegou a um beco sem saída com *na-cadeia-alimentar*(alga, *y*) e vai voltar para trás, subindo. Como não existe nenhum outro fato da forma *come*(peixinho,

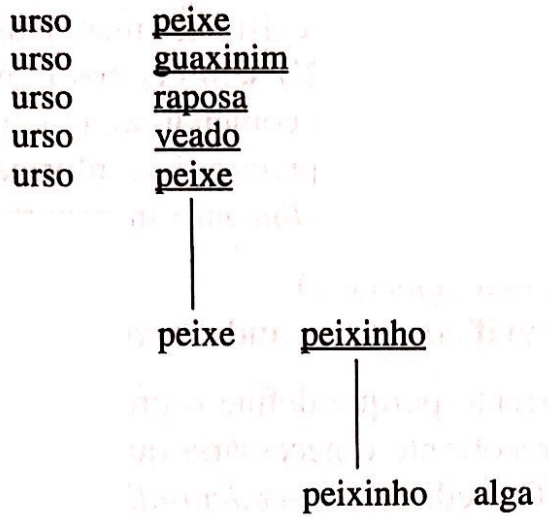


Fig. 1.2

z), a busca para soluções de *na-cadeia-alimentar*(peixinho, *y*) termina. Então, já que não há nenhum outro fato da forma *come*(peixe, *z*), a procura de soluções para *na-cadeia-alimentar*(peixe, *y*) termina. Voltando ainda mais para trás, existe uma outra correspondência para *come*(urso, *z*) com *z* igual a "guaxinim" que vai gerar outro caminho de busca. ❖

Recorrência

- No exemplo 40, uma vez que Prolog começa a investigação *na-cadeia-alimentar*(peixe,y), todas as respostas que puderem ser obtidas através da exploração desse caminho (respostas 5 e 6) são geradas antes das outras (respostas de 7 a 12).
- Explorar tão longe quanto for possível um determinado caminho e depois voltar por esse mesmo caminho antes de explorar outros é chamado de uma estratégia de **busca em profundidade.**

Problema Prático 30

Faça o acompanhamento da execução do programa em Prolog do Exemplo 40 e explique por que as respostas de 7 a 12 ocorrem. ❖

30. As respostas 7 a 9 resultam de *na-cadeia-alimentar*(guaxinim, *y*); as respostas 10 e 11 resultam de *na-cadeia-alimentar*(raposa, *y*); a resposta 12 resulta de *na-cadeia-alimentar*(veado, *y*).

Sistemas Especialistas

- Foram desenvolvidos muitos programas de aplicações interessantes, em Prolog e em linguagens de programação lógica semelhantes, que reúnem um banco de dados de fatos e regras sobre algum domínio e depois usam esse banco de dados para chegar a conclusões.
- Tais programas são conhecidos como **sistemas especialistas**, **sistemas baseados no conhecimento** ou **sistemas baseados em regras**.

Sistemas Especialistas

- O banco de dados de um sistema especialista tenta retratar o conhecimento (“captar a experiência”) de um especialista humano em um campo particular, incluindo tanto os fatos conhecidos pelo especialista, como seus métodos de raciocínio para chegar a conclusões a partir desses fatos.
- O sistema especialista completo não apenas simula as ações de um especialista humano mas também pode ser questionado a fim de indicar por que tomou certas decisões e não outras.

Sistemas Especialistas

- Foram construídos sistemas especialistas que simulam, por exemplo:
  - a diagnose de um médico especialista a partir dos sintomas de um paciente;
  - as decisões de um gerente de fábrica sobre o controle de válvulas em uma facilidade química baseado nas leituras dos sensores;
  - as decisões de um comprador de roupas para uma loja baseada em pesquisa de mercado;
  - as escolhas feitas por um consultor especificando uma configuração para um sistema baseado nas necessidades do consumidor, e muitos outros.
- O desafio em construir um sistema especialista reside em extrair todos os fatos e regras pertinentes do especialista humano.

Exercícios 1-8

Os Exercícios de 1 a 6 se referem ao banco de dados do Exemplo 39; encontre os resultados da pergunta em cada caso.

1. **is**(*come*(urso, peixinho))
2. **is**(*come*(raposa, coelho))
★3. **which**(x: *come*(guaxinim, x))
4. **which**(x: *come*(x, grama))
5. **which**(x: *come*(urso, x) **and** *come*(x, coelho))
6. **which**(x: *presa*(x) **and not**(*come*(raposa, x)))
★7. Formule uma regra de Prolog que defina "herbívoro" para adicionar ao banco de dados do Exemplo 39.
8. Se a regra do Exercício 7 for incluída no banco de dados do Exemplo 39, diga qual vai ser a resposta à pergunta

   which(x: *herbívoro*(x))

come(urso, peixe)
come(peixe, peixinho)
come(peixinho, alga)
come(guaxinim, peixe)
come(urso, guaxinim)
come(urso, raposa)
come(raposa, coelho)
come(coelho, grama)
come(urso, veado)
come(veado, grama)
come(lince, veado)

animal(urso)
animal(peixe)
animal(peixinho)
animal(guaxinim)
animal(raposa)
animal(coelho)
animal(veado)
animal(lince)
planta(grama)
planta(alga)

presa(x) **if** *come*(y, x) **and** *animal*(x)

Exercício 9

9. Um banco de dados Prolog contém os dados a seguir, onde *patrão*(*x*, *y*) significa que "*x* é patrão de *y*" e *supervisor*(*x*, *y*) significa que "*x* é supervisor de *y*".

patrão(Miguel, Joana)
patrão(Judite, Miguel)
patrão(Anita, Judite)
patrão(Judite, Kim)
patrão(Kim, Henrique)
patrão(Anita, Samuel)
patrão(Henrique, Jeferson)
patrão(Miguel, Hamal)

supervisor(*x*, *y*) **if** *patrão*(*x*, *y*)
supervisor(*x*, *y*) **if** *patrão*(*x*, *z*) **and** *supervisor*(*z*, *y*)

Encontre os resultados das seguintes perguntas:

a. **which**(*x*: *patrão*(*x*, Samuel))
b. **which**(*x*: *patrão*(Judite, *x*))
c. **which**(*x*: *supervisor*(Anita, *x*))

Exercício 10

10. Construa um banco de dados Prolog que fornece informações sobre estados e suas capitais. Algumas cidades são grandes, outras são pequenas. Alguns estados estão na região sul, outros na região nordeste.

a. Escreva uma pergunta para encontrar todas as capitais pequenas.
b. Escreva uma pergunta para encontrar todos os estados que têm capitais pequenas.
c. Escreva uma pergunta para encontrar todos os estados na região nordeste com capitais grandes.
d. Formule uma regra para definir cidades cosmopolitas como sendo capitais grandes dos estados na região sul.
e. Escreva uma pergunta para encontrar todas as cidades cosmopolitas.

Exercício 11

★11. Suponha que existe um banco de dados Prolog que fornece informações sobre autores e os livros que escreveram. Os livros são classificados como sendo de ficção, biografia ou referência.

a. Escreva uma pergunta para saber se Machado de Assis escreveu Iracema.
b. Escreva uma pergunta para descobrir todos os livros escritos por Jorge Amado.
c. Formule uma regra para definir autores de livros que não são de ficção.
d. Escreva uma pergunta para encontrar todos os autores de livros que não são de ficção.

Exercício 17 (7ª edição)

17. Suponha que exista um banco de dados Prolog que forneça informações sobre uma família. Os predicados *homem*, *mulher* e *genitorde* estão incluídos.
    a. Formule uma regra para definir *paide*.
    b. Formule uma regra para definir *filhade*.
    c. Formule uma regra recorrente para definir *ancestralde*.

https://pt.wikipedia.org/wiki/Silvio_Santos

Exercício 18 (7ª edição)

18. Suponha que exista um banco de dados Prolog que forneça informações sobre as peças em um motor de automóvel. Os predicados *grande*, *pequena* e *partede* estão incluídos.

a. Escreva uma consulta para encontrar todas as peças pequenas que fazem parte de outras peças.

b. Escreva uma consulta para encontrar todas as peças grandes que são formadas por peças pequenas.

c. Formule uma regra recorrente para definir *componentede*.`

Transmissão
- Diferencial
  - Rolamento
  - Cubo
- Embreagem
  - Disco
  - Tampa

Suspensão
- Amortecedor
  - Mola
  - Batente
- Pneu
  - Bico
  - Carcaça

Exercício 19 (7ª edição)

19. Suponha que exista um banco de dados Prolog que forneça informações sobre os ingredientes nos itens do cardápio de um restaurante. Estejam incluídos os predicados *seco*, *líquido*, *perecível* e *ingredientesde*.

a. Escreva uma consulta para encontrar todos os ingredientes secos de outros ingredientes.

b. Escreva uma consulta para encontrar todos os ingredientes perecíveis que contêm líquidos como ingredientes.

c. Formule uma regra recursiva para definir *encontradoem*.

Bolo (item)

- Massa
  - Ovo
  - Fermento
  - Farinha
  - Água
  - Manteiga
- Cobertura
  - Leite
  - Chocolate

Exercício 20 (7ª edição)

20. Suponha que exista um banco de dados Prolog que forneça informações sobre voos da linha aérea SV (Sempre Voando). Estejam incluídos os predicados *cidade* e *voo*. Aqui *voo*(*X*, *Y*) significa que SV tem um voo direto (sem parada) da cidade *X* para a cidade *Y*.

a. Escreva uma consulta para encontrar todas as cidades a que você pode ir por um voo direto saindo de Belo Horizonte.

b. Escreva uma consulta para encontrar todas as cidades que têm voo direto para o Rio de Janeiro.

c. Formule uma regra recursiva para definir *rota*, em que *rota*(*X*, *Y*) significa que você pode ir da cidade *X* para a cidade *Y* usando a SV, mas pode não ser um voo direto.

Belo Horizonte	Rio de Janeiro	São Paulo
São Paulo	São Paulo	Belo Horizonte
Rio de Janeiro	Belo Horizonte	Rio de Janeiro
Itabira	Volta Redonda	Guarulhos

Resolução dos Exercícios em Prolog

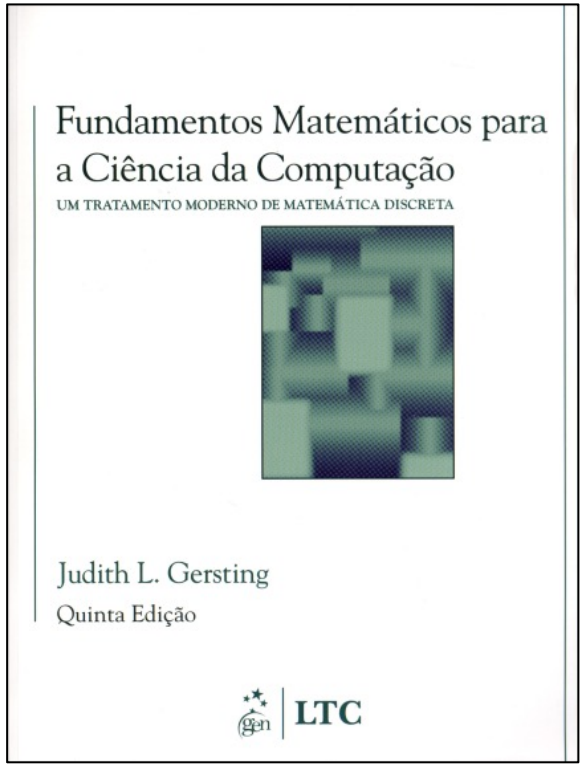
- Códigos disponíveis no seguinte repositório do GitHub:

https://github.com/marcelovca90-inatel/M020

- Interpretador utilizado: SWI-Prolog for SHaring

https://swish.swi-prolog.org/

Referência Bibliográfica



GERSTING, Judith L.; IÓRIO, Valéria de Magalhães, Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta. 5 ed. Rio de Janeiro, RJ: LTC, 2004, 597 p. ISBN 978-85-216-1422-7.