

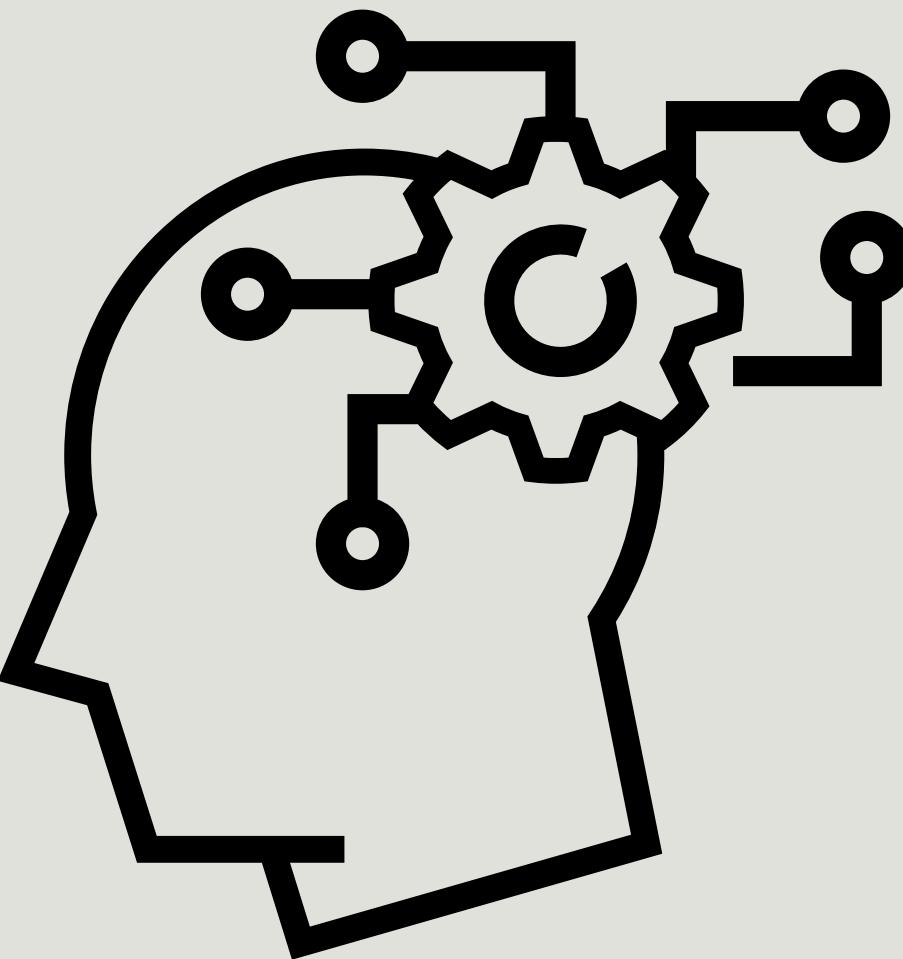
PARADIGMA LÓGICO



O que é?

O paradigma lógico é um modelo de programação baseado na lógica formal, especialmente na lógica de **predicados de primeira ordem**.

No paradigma lógico, em vez de descrever como executar as tarefas passo a passo (como nos paradigmas imperativo e procedural), descreve-se **o que deve ser verdadeiro**.



O que é?



Conceitos básicos:

- Fatos
- Regras
- Consultas

O que é?

Como funciona?

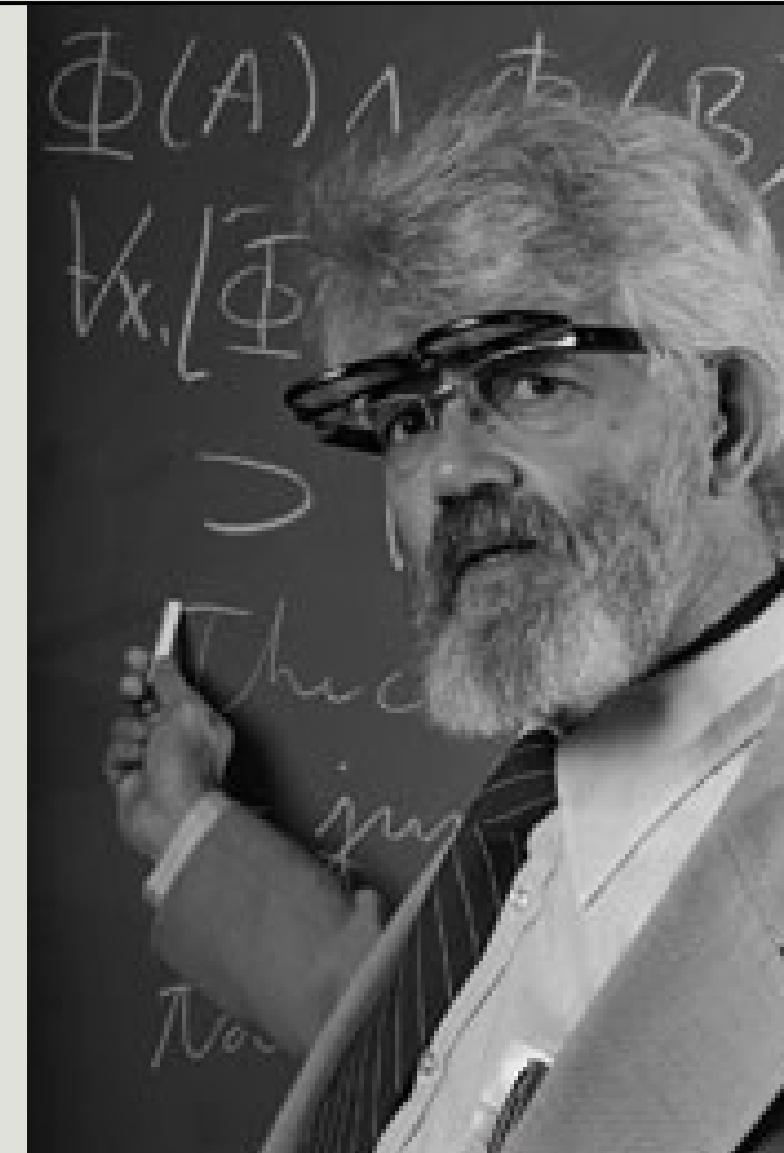
No paradigma lógico, você escreve um programa declarando fatos e regras que descrevem o problema. Quando você faz uma consulta, o sistema de inferência lógica usa esses fatos e regras para responder à consulta.



Histórico

IDEIA INICIAL

John McCarthy propôs:
"Programas para manipular com sentenças
instrumentais comuns apropriadas à
linguagem formal (muito provavelmente
uma parte do cálculo de predicado)".



John McCarthy

Surgimento do Prolog



A linguagem de programação Prolog nasceu de um projeto que não tinha por foco a implementação de uma linguagem de programação, mas o processamento de linguagens naturais.

Durante e após a Segunda Guerra Mundial, houve um aumento significativo no interesse pela **automação e pelo processamento de informações**. Isso levou a uma busca por métodos eficazes para representar conhecimento e raciocinar sobre ele de forma computacionalmente viável.

1920-1930

Nesse período, os avanços na lógica matemática levaram ao desenvolvimento de formalismos para descrever o **raciocínio lógico e a dedução formal**. O trabalho de lógicos como Kurt Gödel e Alonzo Church foi fundamental para estabelecer os limites e possibilidades da lógica formal.

1940-1950

O interesse pela **inteligência artificial** e a necessidade de **linguagens de programação mais expressivas** impulsionaram o desenvolvimento de linguagens baseadas em lógica, como **LISP** e **Prolog**. O trabalho de John McCarthy e outros na inteligência artificial ajudou a popularizar o uso de **linguagens de programação lógica**.

1950-1960

Como mencionado anteriormente, o **Prolog** foi desenvolvido no início da década de 1970 por **Alain Colmerauer** e **Philippe Roussel** na Universidade de Marselha, na França. O Prolog representou uma abordagem inovadora para a programação, permitindo que os programadores especificassem relações lógicas e regras de inferência em vez de instruções sequenciais.

1960-1970

1970-1980

Durante esta década, o Prolog e suas variantes, como o **SWI-Prolog**, ganharam destaque em pesquisa acadêmica e aplicações industriais. O Prolog foi amplamente utilizado em áreas como inteligência artificial, processamento de linguagem natural, sistemas especialistas e engenharia de software.

1980- ~

Desde os anos 1980, o Prolog e o paradigma lógico continuaram a ser desenvolvidos e aplicados em uma variedade de contextos. O SWI-Prolog, em particular, se destacou como uma implementação de **código aberto de Prolog** com uma **comunidade ativa e suporte contínuo**.

Atualmente

Evolução Tecnológica

O paradigma lógico continua a evoluir com o avanço da tecnologia. Implementações modernas de Prolog, como o SWI-Prolog, incorporam melhorias de desempenho, suporte a bibliotecas e ferramentas, integração com outras linguagens e ambientes de desenvolvimento, entre outras características.

Integração com outros paradigmas

Embora o paradigma lógico seja distinto, ele frequentemente se integra com outros paradigmas de programação. Por exemplo, é comum ver o Prolog sendo utilizado em conjunto com linguagens funcionais, imperativas ou orientadas a objetos para aproveitar as vantagens de cada paradigma em diferentes partes de um sistema.

Atualmente

Desafios e Oportunidades

Como qualquer paradigma de programação, o paradigma lógico enfrenta desafios e oportunidades contínuas. Desafios incluem questões de **desempenho, escalabilidade, eficiência e expressividade da linguagem**. No entanto, há oportunidades para inovação e progresso através do desenvolvimento de novas técnicas, ferramentas e aplicações.

Continuidade na Pesquisa Acadêmica

O paradigma lógico continua a ser objeto de estudo e pesquisa em instituições acadêmicas em todo o mundo. Pesquisadores exploram novas técnicas, algoritmos e aplicações para estender e aprimorar as capacidades do Prolog e do paradigma lógico em geral.

Instalação Prolog

LINUX:

Para instalar o SWI -Prolog no Linux (Ubuntu) siga os seguintes passos:

- Adicione ppa ppa:swi-prolog/stable às fontes de software do seu sistema:
- Abra um terminal (Ctrl+Alt+T) e digite:

sudo add-apt-repository ppa:swi-prolog/stable

- Depois, atualize as informações do pacote:

sudo apt-get update

Instalação Prolog

LINUX:

Instale o SWI-Prolog através do gerenciador de pacotes:

- Abra um terminal (Ctrl+Alt+T) e digite:

`sudo apt-get install swi-prolog`

Instalação SWI Prolog

Windows:

- Acesse o site oficial do SWI-Prolog:
- Navegue até a seção de Downloads:
- No menu do site, clique em "Download for windows"
- Selecione a versão mais recente.



SWI-Prolog downloads

SWI Prolog

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY COMMERCIAL WIKI

- Development release ★
- Stable release
- Daily builds for Windows
- Browse GIT repository



Download daily builds for Windows

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY COMMERCIAL WIKI

The table below provides access to the most recent 7 daily builds of SWI-Prolog for Windows, both the 32- and 64-bit versions. The build is done automatically from the [GIT sources](#). The files use the following naming convention:

- `swipl-wbits-date.exe`

Please note that these versions **may be unstable!** It is advised to follow current discussions on the [mailing list](#) and/or the git commit messages at [GitHub](#). The primary purpose of the daily builds is to quickly provide binaries after a bug report.

Name	Last modified	Size
Up	-	-
swipl-w64-2024-05-31.exe	Fri May 31 04:34:55 2024	13,750,393
swipl-w32-2024-05-31.exe	Fri May 31 04:24:53 2024	13,797,840
swipl-w64-2024-05-30.exe	Thu May 30 04:35:27 2024	13,749,966
swipl-w32-2024-05-30.exe	Thu May 30 04:24:43 2024	13,797,466
swipl-w64-2024-05-29.exe	Wed May 29 04:36:56 2024	13,750,501
swipl-w32-2024-05-29.exe	Wed May 29 04:26:12 2024	13,797,903
swipl-w64-2024-05-28.exe	Tue May 28 04:35:20 2024	13,746,787
swipl-w32-2024-05-28.exe	Tue May 28 04:24:59 2024	13,793,626
swipl-w64-2024-05-27.exe	Mon May 27 15:51:40 2024	13,748,023
swipl-w32-2024-05-27.exe	Mon May 27 15:41:23 2024	13,793,529

Instalação SWI Prolog

Windows:

- Aceite o termo de antivirus e clique na instalação:

 Windows antivirus software works using *signatures* and *heuristics*. Using the huge amount of viruses and malware known today, arbitrary executables are often [falsely classified as malicious](#). [Google Safe Browsing](#), used by most modern browsers, therefore often classifies our Windows binaries as malware. You can use e.g., [virustotal](#) to verify files with a large number of antivirus programs.

Our Windows binaries are cross-compiled on an isolated Linux container. The integrity of the binaries on the server is regularly verified by validating its SHA256 fingerprint.

Please select the checkbox below to enable the actual download link.

I understand

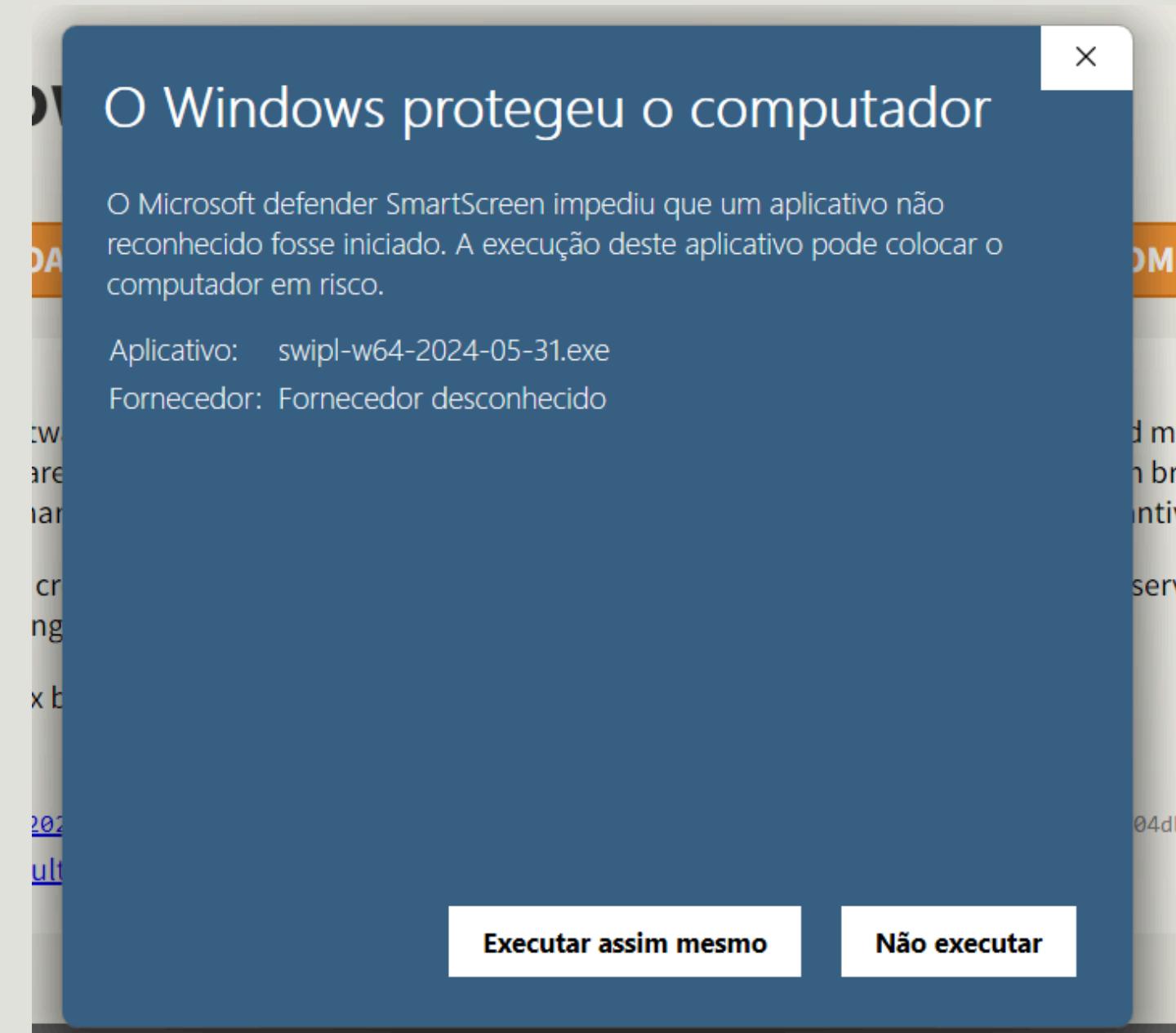
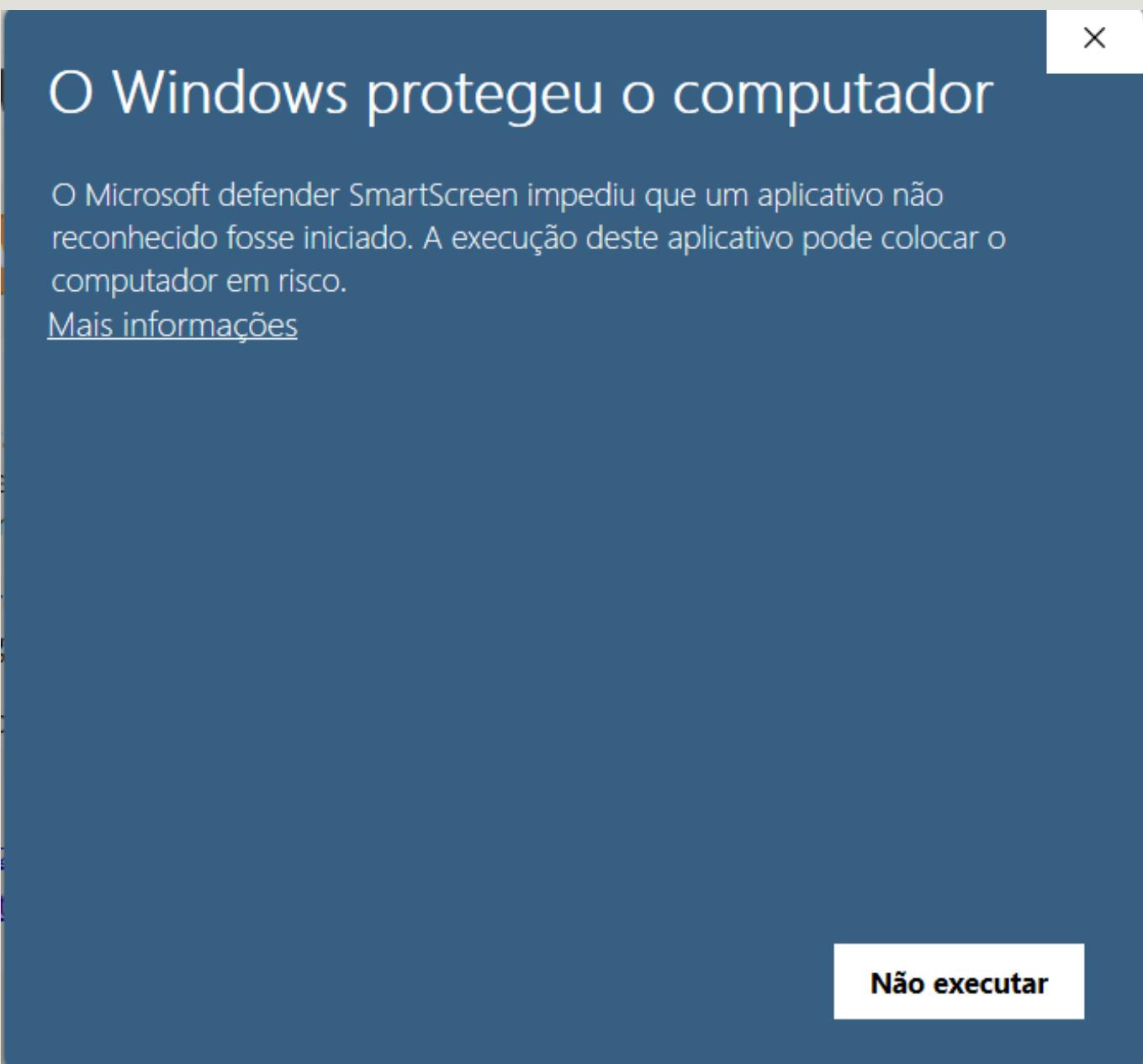
[Download swipl-w64-2024-05-31.exe](#) (SHA256: d1fc412900383734bfb4a7e8f3d39262266177bee0322936f9850ec7f95f04db)

[VIRUSTOTAL Scan Result](#)

Instalação SWI Prolog

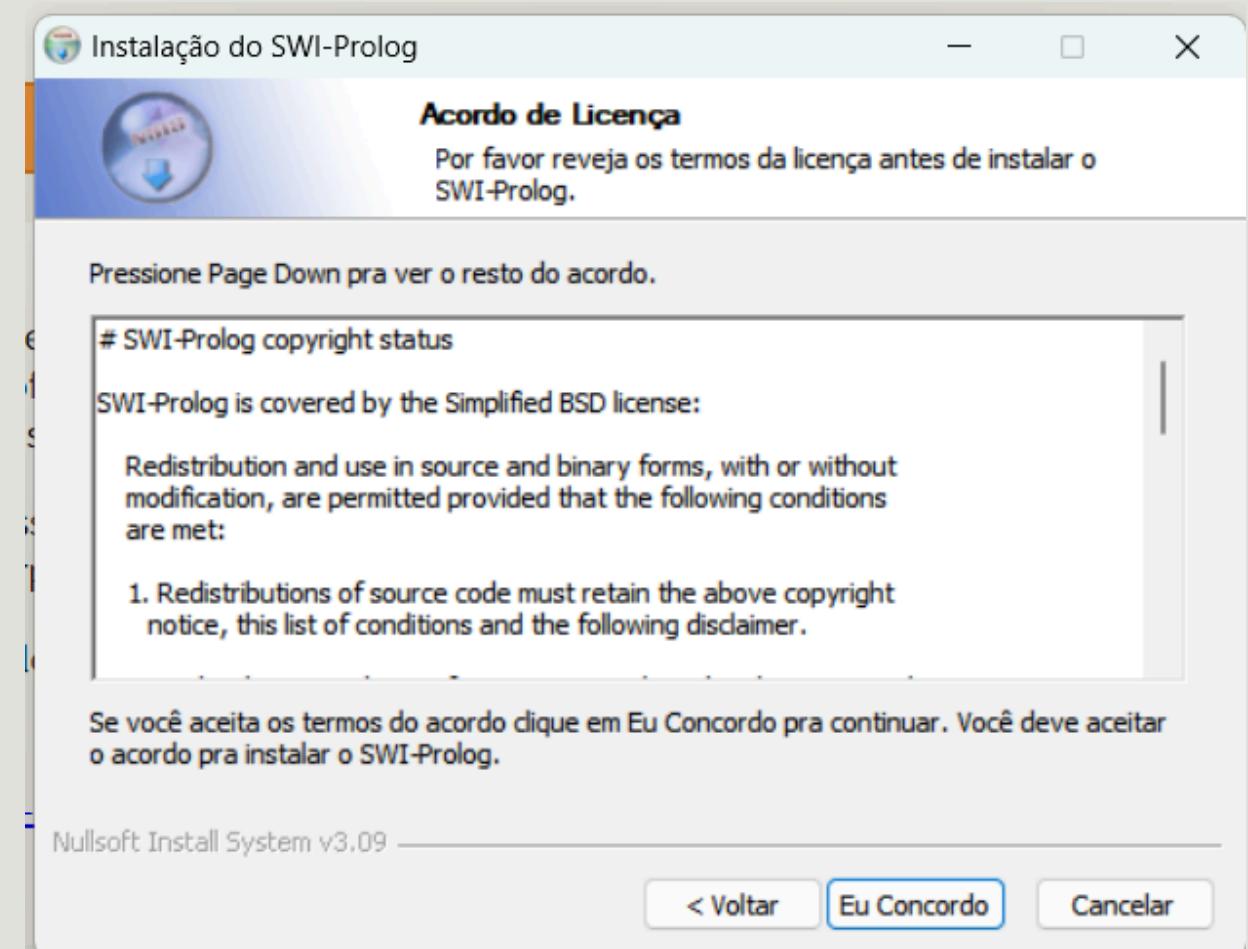
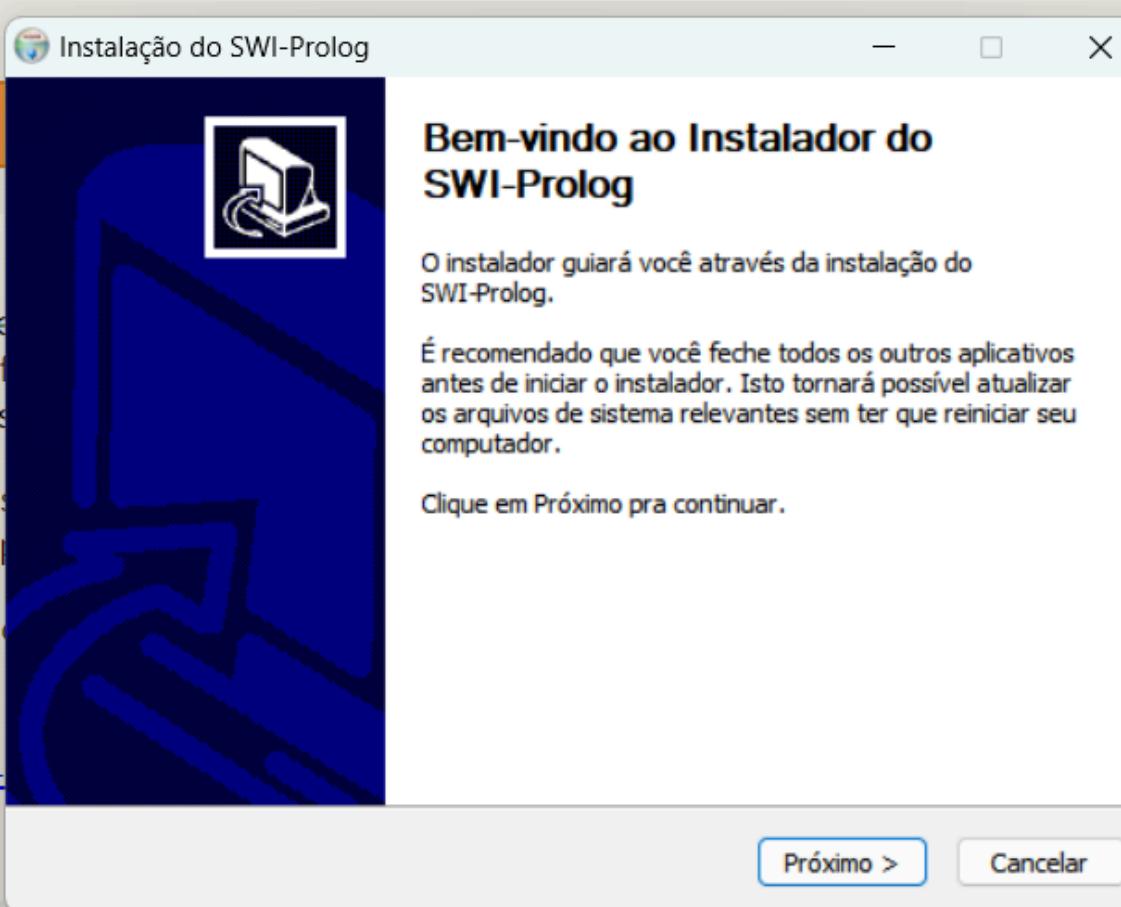
Windows:

Aceite e execute o arquivo mesmo que o Windows comunique

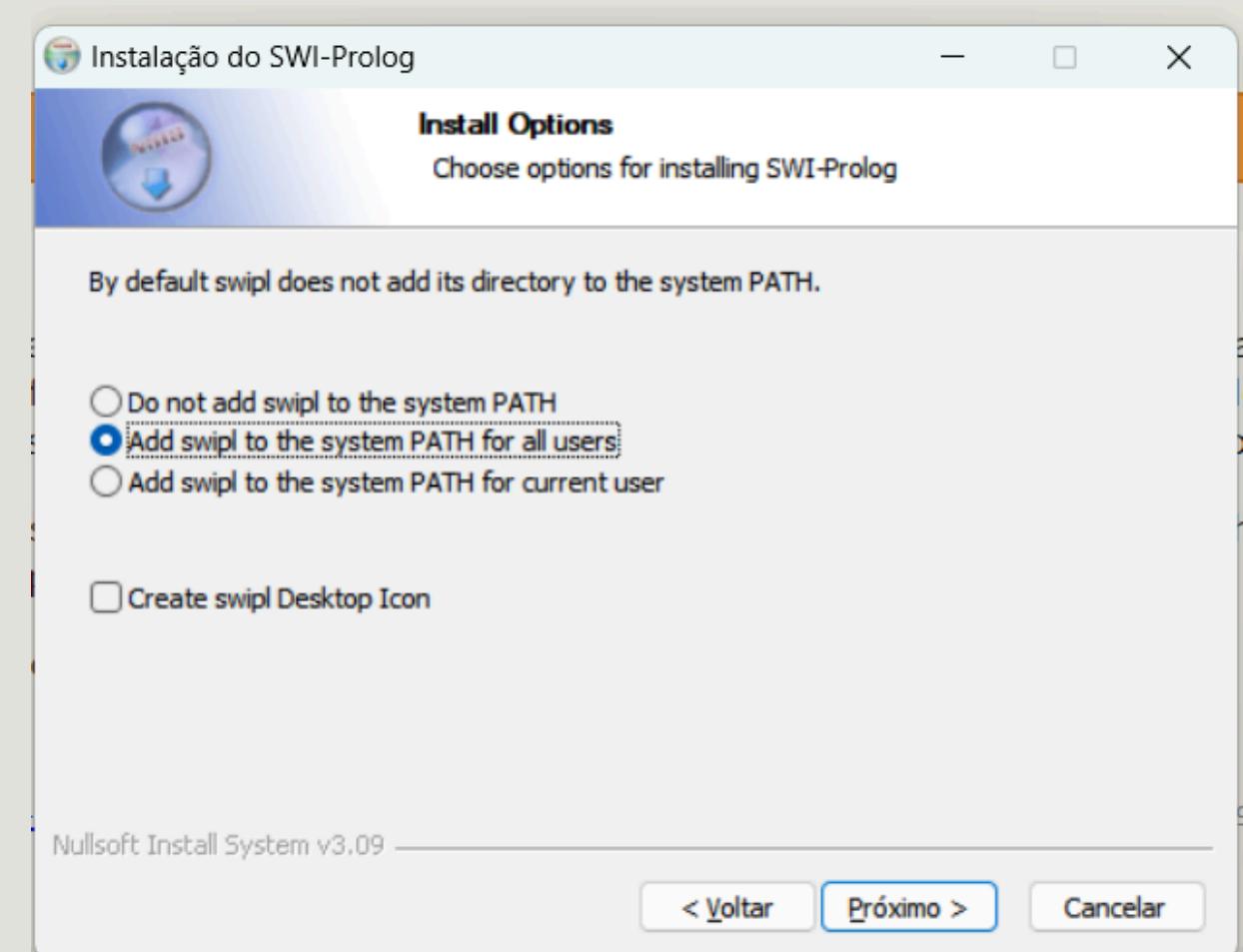


Instalação SWI Prolog

Windows:



Aceite os termos de uso



Add o swipl no sistema

Sintaxe

- **Variáveis:** Iniciadas com letra maiúscula ou underscore (_), sendo anônimas as variáveis que começam com underscore. Ex.: Joao, _joao, _
- **Átomos:** são constantes, começam com letra minúscula ou entre aspas simples. Ex.: joao, 'João'.
- **Inteiros:** Sequência numérica sem (.) ou caracteres ASCII entre aspas, que serão tratados como lista de inteiros. Ex.: 1, 4, "b".
- **Floats:** números com um ponto (.) e pelo menos uma casa decimal. Ex.: 6.1
- **Listas:** Sequência de elementos entre [] separados por vírgula. Ex.: [a, b, c]

Sintaxe

Termo	Classificação
vINCENT	átomo
23	íntero
variable23	átomo
aulas de lógica	erro
'Joao'	átomo
[1, [2, 3], 4]	lista
Footmassage	variável
65.	erro
23.0	float
-	variável anônima
'aulas de lógica'	átomo
"a"	lista
[]	lista

Sintaxe

Comandos

- **Write:** utilizado para escrever:

`write('Teste de impressão').` Correto

`write(Teste de impressão).` Incorreto

`write(joao).` Correto

Sintaxe

Comandos

- **Read:** utilizado para ler variáveis:

`read(X).` Correto

`read(x).` Incorreto

`read(Joao).`

`read(joao).`

Fatos

Fatos são sempre verdadeiros.

Exemplos:

- **homem(x).** = significa que x é um homem
- **genitor(x, y).** = significa que “x é genitor de y” ou “y é gerado de x”

Fatos

homem(tom).

homem(bob).

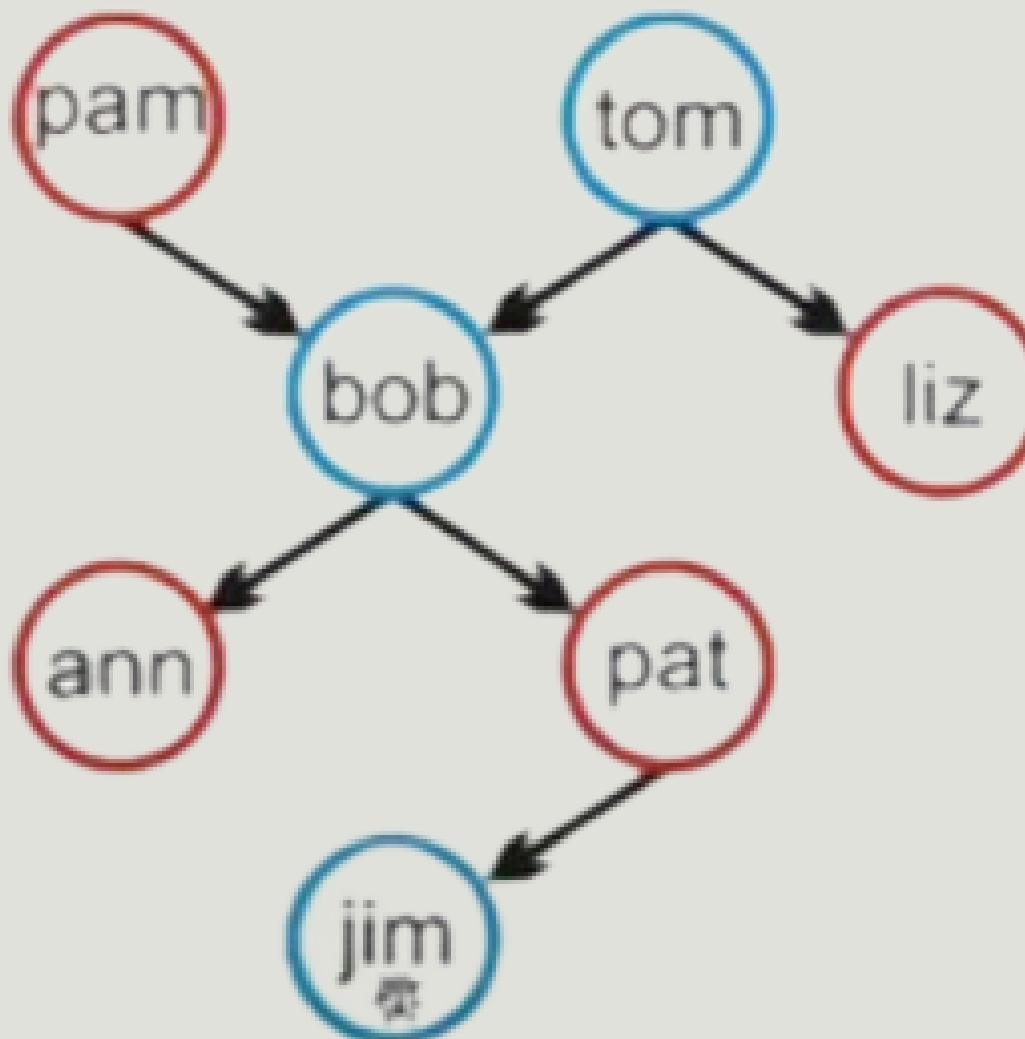
homem(jim).

mulher(pam).

mulher(liz).

mulher(ann).

mulher(pat).



genitor(pam, bob).

genitor(tom, bob).

genitor(tom, liz).

genitor(bob, ann).

genitor(bob, pat).

genitor(pat, jim).

Regras

Facilitam as consultas e tornam o programa mais expressivo.

$A(x) \rightarrow B(x) \vee (C(x) \wedge D(x))$, seria escrita em Prolog como: `a(X) :- b(X); (c(X), d(X)).`

Exemplos:

- **Regra se X é filho de Y:** `filho(X, Y) :- genitor(Y, X).`
- **Regra se X é mãe de Y:** `mae(X, Y) :- genitor(X, Y), mulher(X).`
- **Regra se X é avo de Z:** `avo(X, Z) :- genitor(X, Y), genitor(Y, Z).`

Consultas

Não é necessário programar os retornos, apenas são feitas consultas.

Exemplos:

- **mulher(pam).** / Retorno = True
- **genitor(bob, X).** / Retorno = X = ann; X = pat; false.

Consultas

Como são feitas as consultas?

- **Matching:** Compara a estrutura das expressões (**pai (joao, maria) = pai(X,Y)**), ou seja, verifica se há um padrão.
- **Unificação:** substitui o valor das variáveis para determinar se a consulta satisfaz os fatos e as regras do programa.
- **Resolução:** inferência lógica dos fatos e regras
- **Recursão:** Regras que chamam a si mesmas. Ex.: Fatorial de algum número
- **Backtracking:** Checagem de todas as possibilidades. Ex.: Se João tem mais de um filho, a consulta retorna todos.

Aritmética

É possível utilizar duas notações para representar expressões em Prolog:

- **Infixa:** $(2 + 3)$.

```
soma_infixa(X, Y, Resultado) :-  
    Resultado is (X + Y).
```

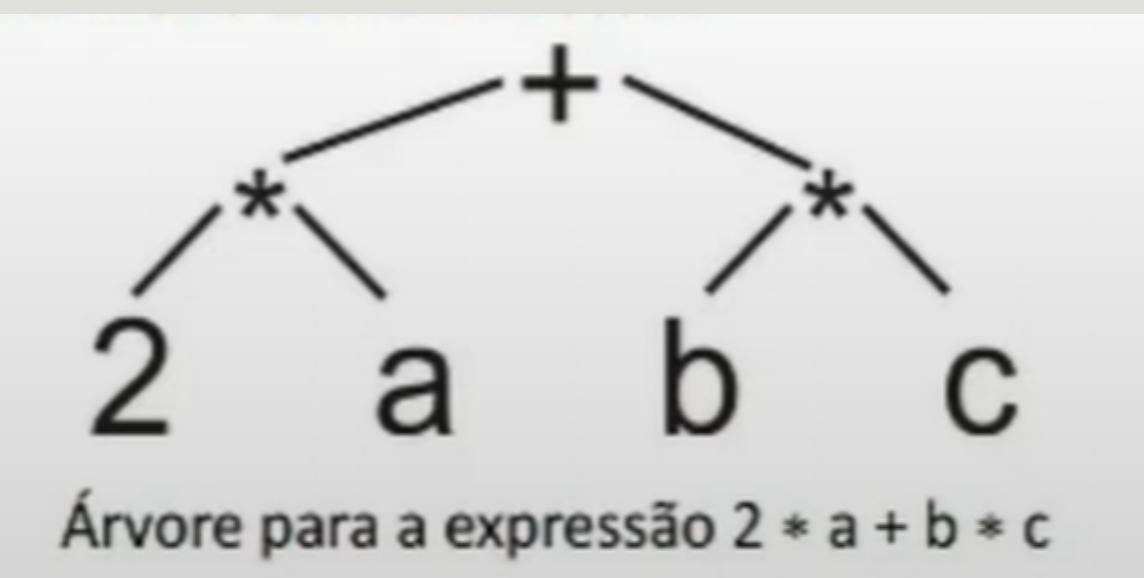
```
?- soma_infixa(2,3,Z).  
Z = 5.
```

- **Prefixa:** $+(2,3)$.

```
soma_prefixa(X, Y, Resultado) :-  
    Resultado is +(X, Y).
```

```
?- soma_prefixa(2,3,Z).  
Z = 5.
```

Prolog trata as representações de forma equivalente, pois, internamente utiliza árvores para representar expressões. Assim, basta mudar a ordem de caminhamento para obter uma ou outra forma.



Aritmética

São oferecidos diversos operadores para cálculos aritméticos:

Operador	Significado
+ , - , * , /	Realizar soma, subtração, multiplicação e divisão, respectivamente
is	Atribui uma expressão numérica à uma variável
mod	Obter o resto da divisão
^	Calcular potenciação
cos, sis, tan	Função cosseno, seno e tangente, respectivamente
exp	Exponenciação
ln, log	Logaritmo natural e logaritmo
sqrt	Raiz

Aritmética

Operadores aritméticos:

Operador	Significado
>	Maior que
<	Menor que
\geq	Maior ou igual a
\leq	Menor ou igual a
$=, =$	igual
\neq	diferente
$\text{\textbackslash}+$	Negação – retorna sucesso se o predicado for falso e vice-versa.

Os operadores $=$ e $=, =$ realizam diferentes tipos de comparação

- $=$ atribui valores para as variáveis (unificação de termos)
- $=, =$ avalia se os valores são iguais

Vantagens

1- Expressividade e Clareza

O paradigma lógico permite que problemas sejam descritos em lógica matemática, tornando os programas claros de se entender. As regras e os fatos podem ser expressos de forma declarativa, descrevendo o que se deseja alcançar, em vez de como alcançar.

Assim, entender o caminho para alcançar a resposta se torna mais claro e de fácil compreensão, já que as respostas do programa já estão expostas, basta saber quais delas são as coerentes para o caso em questão.

Vantagens

2- Inferência automática

Uma das características mais poderosas das linguagens lógicas é a capacidade de realizar inferências automáticas. Com um conjunto de fatos e regras, o mecanismo de inferência do sistema lógico pode derivar novas informações, encontrar soluções para problemas e responder a consultas de forma automática.

Isto facilita encontrar soluções e consultas, já que terá um procedimento/regra pré-definida que por meio da inferência irá direcionar o código para o objetivo correto, facilitando o entendimento do código.

Vantagens

3- Implementação de Algoritmos de Busca e Backtracking

O paradigma lógico é particularmente adequado para a implementação de algoritmos que envolvem busca e backtracking, comuns em problemas de inteligência artificial, como resolução de quebra-cabeças, planejamento e resolução de problemas de satisfação de restrições.

A utilização do backtracking em primeira vista aparenta ser um método ruim, pois o programa teria que analisar várias vezes para saber o caminho correto, porém é excelente, pois as respostas corretas teram um maior nível de clareza, já que o programa sabe que as outras respostas são as erradas. Isso é ótimo para a inteligência artificial.

Vantagens

4 - Prototipação Rápida

Devido à natureza declarativa e à capacidade de inferência automática, o desenvolvimento de protótipos para problemas complexos pode ser mais rápido em linguagens de programação lógica. Isso facilita a exploração e a validação rápida de soluções potenciais.

Vantagens

5 - Facilidade de Manipulação de Conhecimento

Sistemas baseados em lógica são eficazes na representação e manipulação de conhecimento. É fácil adicionar novos fatos e regras ao sistema e atualizar o conhecimento existente sem a necessidade de reescrever grandes partes do código.

Atualizar o código fica mais fácil, já que é utilizada a linguagem lógica, faz com que se assemelhe a linguagem humana, tornando a compreensão do código mais fácil.

Desvantagens

1- Desempenho e Eficiência:

Embora algumas das vantagens desse paradigma estejam relacionadas na melhora do desempenho do código, seu desempenho é considerado ainda bem prejudicado, tais vantagens amenizam essa perda de eficiência em comparação com os outros paradigmas.

Os sistemas baseados em lógica, especialmente aqueles que utilizam mecanismos de inferência automática como o Prolog, podem ser menos eficientes em termos de tempo de execução e uso de recursos comparados a linguagens imperativas ou orientadas a objetos. O processo de unificação e a busca exaustiva podem ser computacionalmente intensivos, especialmente em problemas grandes e complexos.

Desvantagens

2- Escalabilidade:

Escalabilidade refere-se à capacidade de um sistema de lidar com um aumento na carga de trabalho ou no volume de dados sem sofrer degradação significativa no desempenho. No contexto do paradigma lógico, a escalabilidade pode ser desafiadora devido a vários fatores inerentes à forma como esses sistemas operam.

À medida que o conjunto de regras e fatos cresce, a complexidade e o tempo necessário para a inferência podem aumentar significativamente, dificultando a escalabilidade de sistemas lógicos para aplicações muito grandes ou dinâmicas.

Desvantagens

3- Dificuldade de Depuração:

Embora o paradigma lógico ofereça benefícios em termos de clareza e estrutura, a depuração de programas lógicos pode ser mais complicada. Isso ocorre porque:

Rastreamento de Erros: Encontrar a origem de um erro em um conjunto complexo de regras e fatos pode ser desafiador. A depuração exige entender não apenas onde o erro ocorreu, mas como a sequência de inferências levou a esse ponto.

Desvantagens

4- Limitado para Aplicações Numéricas:

O paradigma lógico não é naturalmente adequado para aplicações que envolvem cálculos numéricos intensivos ou manipulação de grandes matrizes de dados.

Embora seja possível realizar cálculos numéricos em linguagens lógicas, isso geralmente é menos eficiente e mais complexo do que em linguagens projetadas para esse fim.

O paradigma foca mais em uma linguagem declarativa, isso faz com que cálculos numéricos não sejam um destaque neste tipo de paradigma em comparação com os outros.

Desvantagens

5- Curva de Aprendizado:

Para programadores acostumados com paradigmas imperativos ou orientados a objetos, a transição para o paradigma lógico pode ser difícil. A abordagem declarativa requer um modo de pensamento diferente, e dominar a lógica formal e as técnicas de inferência pode exigir um esforço considerável.

Por ser uma maneira de se construir um código e de realizar objetivo por meio dessas linguagens, utilizar linguagens como prolog podem ser difíceis à primeira vista já que causam essa estranheza no primeiro contato e consequentemente dificulta o aprendizado do programador à esse paradigma.

Desvantagens

6- Complexidade na Expressão de Algoritmos:

Como há uma maior dificuldade de aprendizado, embora o paradigma lógico seja excelente para problemas de inferência e busca, expressar algoritmos complexos ou procedimentos específicos pode ser menos intuitivo e mais complicado. A falta de controle explícito sobre o fluxo de execução pode tornar a implementação de certos tipos de algoritmos mais difícil.

Comparação dos paradigmas:

Paradigma Lógico:

- Utiliza lógica matemática
- Primeira linguagem: Planner
- Baseado em fatos e cláusulas verdadeiras
- Diferente de outros paradigmas por não usar instruções explícitas
- Dados imutáveis
- **Declaração de Conhecimento:**
 - Programas são coleções de fatos e regras
- **Resolução Automática:**
 - Soluções deduzidas automaticamente por inferência lógica
- **Não Procedural:**
 - Foca no objetivo a ser alcançado, não em como alcançá-lo

Comparação dos paradigmas:

Paradigma Lógico x Imperativo:

- **Base:** Alteração sequencial do estado do programa
- **Programação:** Usa comandos de atribuição e estruturas de controle (loops, condicionais)
- **Foco:** Como resolver o problema (passo a passo)
- **Resolução:** Controle explícito do fluxo de execução pelo programador
- **Mutabilidade:** Trabalha com dados mutáveis, alterando o estado do programa
- **Uso Ideal:** Problemas de manipulação de estado e controle sequencial

Comparação dos paradigmas:

Paradigma Lógico x Funcional:

- **Base:** Cálculo lambda e teoria dos conjuntos
- **Programação:** Baseada em funções puras e imutáveis
- **Foco:** O que deve ser calculado, sem efeitos colaterais
- **Resolução:** Avaliação de expressões e aplicação de funções
- **Mutabilidade:** Trabalha principalmente com dados imutáveis, evita efeitos colaterais
- **Uso Ideal:** Problemas de transformação de dados, processamento de lista, programação concorrente

Comparação dos paradigmas:

Paradigma Lógico x Orientado a objetos:

- **Base:** Modelagem de objetos e suas interações
- **Programação:** Organiza o código em classes e objetos
- **Foco:** Como os objetos se comportam e interagem
- **Resolução:** Métodos e mensagens entre objetos
- **Mutabilidade:** Trabalha com estado mutável dos objetos
- **Uso Ideal:** Modelagem do mundo real, sistemas complexos com muitas interações

EXERCÍCIOS

1. Qual das seguintes linguagens de programação é mais associada ao paradigma de programação lógica?

- A) Python**
- B) Java**
- C) Prolog**
- D) C++**

1. Qual das seguintes linguagens de programação é mais associada ao paradigma de programação lógica?

- A) Python
- B) Java
- C) Prolog
- D) C++

2. Em programação lógica, o processo de retroceder para um ponto anterior e tentar um caminho alternativo quando uma meta não pode ser provada é chamado de:

- A) Recursão**
- B) Iteração**
- C) Backtracking**
- D) Herança**

2. Em programação lógica, o processo de retroceder para um ponto anterior e tentar um caminho alternativo quando uma meta não pode ser provada é chamado de:

- A) Recursão
- B) Iteração
- C) Backtracking
- D) Herança

3. Em Prolog, a operação que combina metas com fatos e regras para provar uma declaração é conhecida como:

- A) Atribuição
- B) Unificação
- C) Compilação
- D) Interpretação

3. Em Prolog, a operação que combina metas com fatos e regras para provar uma declaração é conhecida como:

- A) Atribuição
- B) Unificação
- C) Compilação
- D) Interpretação

4. Ainda em Prolog, qual tipo de dado possui as seguintes características abaixo: “São constantes, devem ser iniciadas com minúsculas seguidas de qualquer caractere alfanumérico ou qualquer sequência entre ‘ ’ (aspas simples)”

- A) Variáveis
- B) Átomos
- C) Inteiros
- D) Listas

4. Ainda em Prolog, qual tipo de dado possui as seguintes características abaixo: “São constantes, devem ser iniciadas com minúsculas seguidas de qualquer caractere alfanumérico ou qualquer sequência entre ‘ ’ (aspas simples)”

- A) Variáveis
- B) Átomos
- C) Inteiros
- D) Listas

5. Em uma consulta, qual é o nome do processo que checa se determinado padrão está presente, para saber quais fatos e regras podem ser utilizados?

- A) Matching**
- B) Unificação**
- C) Resolução**
- D) Recursão**

5. Em uma consulta, qual é o nome do processo que checa se determinado padrão está presente, para saber quais fatos e regras podem ser utilizados?

- A) Matching**
- B) Unificação**
- C) Resolução**
- D) Recursão**

6. Dada a base de fatos abaixo:

```
animal(urso).  
animal(peixe).  
animal(raposa).  
animal(coelho).  
animal(veado).  
planta(alga).  
planta(grama).  
come(urso, peixe).  
come(urso, raposa).  
come(urso, veado).  
come(peixe, alga).  
come(raposa, coelho).  
come(coelho, grama).  
come(veado, grama).
```

Quais são as respostas para as consultas?

- a) ?- planta(_).
- b) ?- come(raposa,_).
- c) ?- come(_,urso).

6. Dada a base de fatos abaixo:

```
animal(urso).  
animal(peixe).  
animal(raposa).  
animal(coelho).  
animal(veado).  
planta(alga).  
planta(grama).  
come(urso, peixe).  
come(urso, raposa).  
come(urso, veado).  
come(peixe, alga).  
come(raposa, coelho).  
come(coelho, grama).  
come(veado, grama).
```

Quais são as respostas para as consultas?

- a) True.
- b) True.
- c) False.

7. Considere a base de fatos e regras em Prolog:

```
gosta(mary,vinho) .  
gosta(roberto,cerveja) .  
gosta(ana,cereja) .  
gosta(augusto,vinho) .  
  
gosta(alberto,X) :- gosta(X,vinho) .
```

7. Qual a resposta correta para a consulta:

?- **gosta(alberto,X).**

a) X = mary ;

X = augusto ;

d) true.

b) X = mary ;

false.

e) X = mary ;

X = augusto ;

false.

c) X = mary ;

X = augusto ;

true.

gosta (mary, vinho) .

gosta (roberto, cerveja) .

gosta (ana, cereja) .

gosta (augusto, vinho) .

gosta (alberto, X) :- gosta (X, vinho) .

7. Qual a resposta correta para a consulta:

?- **gosta(alberto,X).**

a) X = mary ;

X = augusto ;

d) true.

b) X = mary ;

false.

e) X = mary ;

X = augusto ;

false.

c) X = mary ;

X = augusto ;

true.

gosta (mary, vinho) .

gosta (roberto, cerveja) .

gosta (ana, cereja) .

gosta (augusto, vinho) .

gosta (alberto, X) :- gosta (X, vinho) .

8. Considere a seguinte base de fatos em Prolog:

aluno (joao, calculo) .	professor (andre, calculo) .
aluno (maria, calculo) .	professor (julia, estrutura) .
aluno (joel, programacao) .	professor (pedro, programacao) .
aluno (joel, estrutura) .	funcionario (pedro, uves) .
frequenta (joao, uvv) .	funcionario (julia, uves) .
frequenta (maria, uvv) .	funcionario (andre, uvv) .
frequenta (joel, uves) .	

Escreva uma regra que mostre quem são os alunos de um professor X.

sao_alunos_do_professor(A, X) :- professor(X, Materias),
aluno(A, Materias).

9. Suponha os seguintes fatos:

```
nota(joao, 5.0).  
nota(joaquim, 4.5).  
nota(maria, 4.0).  
nota(mary, 10.0).  
nota(jose, 6.5).  
nota(joana, 9.0).  
nota(cleuza, 6.0).
```

Considerando que:

- nota de 7.0 a 10.0 = aprovado
- nota de 5.0 a 6.9 = recuperação
- nota de 0.0 a 4.9 = reprovado

Identifique o erro da regra abaixo:

```
situacao(A) :- nota(A, Nota), (  
    (Nota>=7.0, Nota=<10.0 , write('Aprovado'))  
,  
    (Nota>=5.0, Nota=<6.9 , write('Recuperação'))  
,  
    (Nota>=0.0, Nota=<4.9 , write('Reprovado'))  
).
```

Foi utilizado um conectivo lógico errado, deveria ter sido o conectivo OU (;)

```
situacao(A) :- nota(A, Nota), (           → 0
    (Nota>=7.0, Nota<10.0 , write('Aprovado')) )
    (Nota>=5.0, Nota<6.9 , write('Recuperação')) )
    (Nota>=0.0, Nota<4.9 , write('Reprovado')) )
.
```

10. Crie uma regra em Prolog que peça no console um número inteiro e imprima na tela se o número é maior do que 100 ou se é menor ou igual a 100.

```
maiorQueCem() :- write('Entre como o numero: '),
    read(X),
    (
        (X>100 , write('O numero é maior que cem'))
        ;
        (X=<100 , write('O numero não é maior que cem'))
    ) .
```

Bibliografia:

<https://www.youtube.com/playlist?list=PLZ-Bk6jzsb-OScKa7vhpcQXoU2uxYGaFx>

https://awari.com.br/paradigmas-de-programacao-2/?utm_source=blog&utm_campaign=projeto+blog&utm_medium=Entenda%20os%20principais%20paradigmas%20de%20programa%C3%A7%C3%A3o%20e%20suas%20diferen%C3%A7as#:~:text=O%20paradigma%20l%C3%B3gico%20%C3%A9%20baseado,intelig%C3%A1ncia%20artificial%20e%20sistemas%20especialistas.

<https://www.treinaweb.com.br/blog/linguagens-e-paradigmas-de-programacao>

OBRIGADO!

