

LÓGICA APLICADA À COMPUTAÇÃO

Mini Projeto AB2 - Elaboração de um tradutor de Linguagem Natural para Lógica Proposicional e verificador de validade de sentenças.

Link do projeto no GitHub: <https://github.com/martanascimento1/logic-project>

Discentes: Giovanna Alves Barbosa de Oliveira, Layza Nunes Faria Costa, Vitória Maria Santana Bigi, Marta Mirely Nascimento dos Santos

Plataforma utilizada: Microsoft Copilot e GitHub Copilot

A princípio, foi pedido que a IA implementasse um código em python que verificasse a validade de premissas dadas em linguagem de lógica proposicional, com base nas regras de inferência dadas: Modus Ponens, Modus Tollens, Silogismo Hipotético, Silogismo Disjuntivo, e operações lógicas (AND, OR, NOT)

```
project.py > operacoes_logicas
You, 18 minutes ago | 1 author (You)
1  def modus_ponens(premissas, conclusao):
2      if "P -> Q" in premissas and "P" in premissas:
3          return True
4      return False
5
6  def modus_tollens(premissas, conclusao):
7      if "P -> Q" in premissas and "/Q" in premissas:
8          return True
9      return False
10
11 def silogismo_hipotetico(premissas, conclusao):
12     if "A -> B" in premissas and "B -> C" in premissas:
13         return True
14     return False
15
16 def silogismo_disjuntivo(premissas, conclusao):
17     if "P * Q" in premissas and "/P" in premissas:
18         return True
19     return False
20
21 def operacoes_logicas(premissas, conclusao):
22     if "P & Q" in premissas and "P" in premissas:
23         if "Q" in premissas:
24             return True
25         return False
26     elif "P & Q" in premissas and "Q" in premissas:
27         if "P" in premissas:
28             return True
29     return False
30
31 """FORMATO DE ENTRADA EX.:
32 PREMISA = ["A -> B", "B -> C", "A"]
33 CONCLUSÃO = "C" """
34
```

```

35 premissas_exemplo = input('Digite as premissas (usando letras de A a Z): ')
36 conclusao_exemplo = input("Digite a conclusão (usando letras de A a Z): ")
37
38 regras = [modus_ponens, modus_tollens, silogismo_hipotetico, silogismo_disjuntivo]
39
40 valido = False
41 for regra in regras:
42     resultado = regra(premissas_exemplo, conclusao_exemplo)
43     if resultado:
44         valido = True
45         break
46
47 if valido:
48     print("O argumento é válido.")
49 else:
50     print("O argumento é inválido.")
51

```

Como pode ser observado no print a seguir, o programa não estava retornando a saída correta para uma sentença dada relativamente simples:

```

project.py > operacoes_logicas
32 PREMISSA = ["A -> B", "B -> C", "A"]
33 CONCLUSÃO = "C"
34
35 premissas_exemplo = input('Digite as premissas (usando letras de A a Z): ')
36 conclusao_exemplo = input("Digite a conclusão (usando letras de A a Z): ")
37
38 regras = [modus_ponens, modus_tollens, silogismo_hipotetico, silogismo_disjuntivo]
39
40 valido = False
41 for regra in regras:
42     resultado = regra(premissas_exemplo, conclusao_exemplo)
43     if resultado:
44         valido = True
45         break
46
47 if valido:
48     print("O argumento é válido.")
49 else:
50     print("O argumento é inválido.")
51

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2 GITLENS COMMENTS

```

giovanna@GIOVANNA-DESKTOP:~/ufal/logica/logic-project$ /bin/python3 /home/giovanna/ufal/logica/logic-project/project.py
Digite as premissas (usando letras de A a Z): ["P -> Q", "/P"]
Digite a conclusão (usando letras de A a Z): "Q"
O argumento é válido.
giovanna@GIOVANNA-DESKTOP:~/ufal/logica/logic-project$

```

Dada a sentenças: "P -> Q" (se P, então Q), e "/P" (NOT P) e a conclusão "Q", a saída retornada deveria informar que o argumento é inválido, tendo em vista que a

conclusão correta seria “/Q” (NOT Q). No entanto, o programa retornou que o argumento era válido, fazendo assim com que fosse identificada a existência de um erro no mesmo.

Para corrigir o código, copiamos o mesmo e colamos no GitHub Copilot, informando que haviam erros para alguns casos de saída, e pedindo que este erro fosse corrigido:

```
o código a seguir cumpre seu papel corretamente em boa parte dos testes, mas em alguns casos específicos ele dá um resultado equivocado, tente achar onde está o erro que causa isso e me sugira uma modificação adequada: def modus_ponens(premissas, conclusao): if "P -> Q" in premissas and "P" in premissas: return True return False
```

GitHub Copilot

Referência 1 usada >

O código parece estar verificando corretamente as regras de inferência lógica, mas há um problema: ele não está considerando a conclusão em nenhuma das funções de regra. Em lógica, a conclusão é uma parte crucial do argumento e deve ser verificada em relação às premissas.

Por exemplo, na função `modus_ponens`, a conclusão deve ser "Q" se as premissas forem "P -> Q" e "P". Da mesma forma, para as outras funções.

Aqui está uma versão modificada do código que leva em consideração a conclusão:

```
def modus_ponens(premissas, conclusao):
    if "P -> Q" in premissas and "P" in premissas and conclusao == "Q":
        return True
    return False

def modus_tollens(premissas, conclusao):
    if "P -> Q" in premissas and "/Q" in premissas and conclusao == "/P":
        return True
    return False

def silogismo_hipotetico(premissas, conclusao):
    if "A -> B" in premissas and "B -> C" in premissas and conclusao == "A -> C":
        return True
    return False

def silogismo_disjuntivo(premissas, conclusao):
    if "P * Q" in premissas and "/P" in premissas and conclusao == "Q":
        return True
    return False

def operacoes_logicas(premissas, conclusao):
    if "P & Q" in premissas and "P" in premissas and conclusao == "Q":
        return True
    elif "P & Q" in premissas and "Q" in premissas and conclusao == "P":
        return True
    return False
```

Após as correções apontadas pela IA, o código foi testado com as mesmas sentenças, e desta vez, a saída foi correta:




```
interface.py > ...
27
28
29 """FORMATO DE ENTRADA EX.:
30 PREMISSA = ["A -> B", "B -> C", "A"]
31 CONCLUSÃO = "C" """
32
33 premissas_exemplo = input('Digite as premissas (usando letras de A a Z): ')
34 conclusao_exemplo = input("Digite a conclusão (usando letras de A a Z): ")
35
36 regras = [modus_ponens, modus_tollens, silogismo_hipotetico, silogismo_disjuntivo]
37
38 valido = False
39 for regra in regras:
40     resultado = regra(premissas_exemplo, conclusao_exemplo)
41     if resultado:
42         valido = True
43         break
44
45 if valido:
46     print("O argumento é válido.")
47 else:
48     print("O argumento é inválido.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 5

● giovanna@GIOVANNA-DESKTOP:~/ufal/logica/projetinho$ /bin/python3 /home/giovanna/ufal/logica/projetinho/interface.py
Digite as premissas (usando letras de A a Z): ["P -> Q", "/P"]
Digite a conclusão (usando letras de A a Z): "Q"
O argumento é inválido.
○ giovanna@GIOVANNA-DESKTOP:~/ufal/logica/projetinho$
```

Mesmo após esta correção, identificamos que alguns casos de teste ainda estavam retornando a saída errada, e ao consultar a IA, fomos informados de que uma possível causa para os erros, seria o formato em que as premissas estavam sendo digitadas.

o código está retornando que o argumento é inválido, mesmo quando as premissas são válidas, qual o problema?

 GitHub Copilot  

Used 1 reference >

Para identificar o problema, precisamos de mais informações sobre as funções `modus_ponens`, `modus_tollens`, `silogismo_hipotetico` e `silogismo_disjuntivo`. Essas funções são usadas para avaliar as premissas e a conclusão. Se houver algum problema com essas funções, isso pode levar a um resultado incorreto.

No entanto, uma possível causa do problema pode ser a maneira como as premissas estão sendo inseridas. No código atual, as premissas são lidas como uma única string do usuário. Se as funções esperam uma lista de premissas, isso pode estar causando o problema.


Você poderia tentar modificar a entrada de premissas para ser uma lista de strings, em vez de uma única string. Aqui está um exemplo de como você pode fazer isso:



```
premissas_exemplo = input('Digite as premissas (usando letras de A a Z): ')
premissas_exemplo = premissas_exemplo.split(', ')
```

Neste exemplo, as premissas devem ser inseridas separadas por vírgulas e um espaço, como "A -> B, B -> C, A".

Com as correções sugeridas pelo programa, o modo em que as premissas deveriam ser digitadas, tornou-se mais fácil para o usuário. Além disso, a IA também nos informou que que o programa tinha sido construído para trabalhar com base em

strings exatas, ao invés de qualquer premissa válida, assim, sugerindo uma implementação mais geral.

 GitHub Copilot

Used 1 reference >

O problema está na maneira como as funções `modus_ponens`, `modus_tollens`, `silogismo_hipotetico` e `silogismo_disjuntivo` estão verificando as premissas. Elas estão procurando por strings exatas como "P -> Q", "A -> B", etc., mas as premissas que você está fornecendo são diferentes ("A -> B", "B -> C", "A").

Uma solução seria modificar essas funções para aceitar qualquer premissa válida, em vez de procurar por strings exatas. Aqui está um exemplo de como você pode fazer isso para a função `silogismo_hipotetico`:

```
def silogismo_hipotetico(premissas, conclusao):  
    # Procurar por premissas do tipo "X -> Y" e "Y -> Z"  
    premissa1 = next((p for p in premissas if "->" in p), None)  
    premissa2 = next((p for p in premissas if "->" in p and p.split(" -> ")[1] == conclusao), None)  
  
    if premissa1 and premissa2 and premissa1.split(" -> ")[1] == premissa2.split(" -> ")[0]:  
        return True  
    return False
```

Essa função agora procura por duas premissas que formam um silogismo hipotético válido com a conclusão fornecida, em vez de procurar por strings exatas. Você pode fazer alterações semelhantes nas outras funções para corrigir o problema.

Continuamos percebendo inconsistências durante as execuções do programa com diferentes quantidades de premissas e formatos de conclusão, e após análise, foi observado que para a execução correta do programa, as premissas e conclusão precisam estar digitadas corretamente do ponto de vista lógico. Assim, adicionamos um pequeno guia de entradas esperadas para que o verificador possa analisar corretamente:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 7
giovanna@GIOVANNA-DESKTOP:~/ufal/logica/projetinho$ /bin/python3 /home/giovanna/ufal/logica/projetinho/interface.py
FORMATOS DE ENTRADA E SAÍDA ESPERADOS:

MODUS PONENS:
premissas = P -> Q, P
conclusao = Q

MODUS TOLLENS:
premissas = P -> Q, /Q
conclusao = /P

SILOGISMO HIPOTÉTICO:
premissas = P -> Q, Q -> R
conclusao = P -> R

SILOGISMO DISJUNTIVO:
premissas = P * Q, /P
conclusao = Q

OPERAÇÕES LÓGICAS:
premissas = P & Q
conclusao = P

Digite as premissas (usando letras de A a Z):
```

Em relação a conversão das sentenças em linguagem natural para sentenças em linguagem lógica, nenhuma das IAs utilizadas conseguiu nos fornecer um algoritmo que fosse coerente na hora de cumprir essa tarefa. Tendo em vista que esse é um processo um pouco mais dificultoso, talvez elas ainda não possuam recursos adequados e suficientes.

A imagem abaixo representa um código fornecido pela IA generativa Github Copilot. Inicialmente a ideia era ter um algoritmo simples que converte uma sentença específica para linguagem lógica, e posteriormente seriam adicionadas funcionalidades que permitissem a entrada do próprio usuário com a linguagem natural, dentre outras, entretanto, nem com as sentenças estando já definidas no próprio código, não foi possível converter.

```
conversao.py - logic-project - Visual Studio Code
Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda
conversao.py X
conversao.py > ...
You, há 22 horas | 1 author (You)
1 def traduzir_para_logica(sentenca):
2     # Substituições para as palavras-chave
3     sentenca = sentenca.replace("E", "and")
4     sentenca = sentenca.replace("OU", "or")
5     sentenca = sentenca.replace("ENTÃO", "->")
6     sentenca = sentenca.replace("É falso que", "~")
7
8     # Imprime a sentença traduzida
9     return sentenca
10
11 # Exemplos de sentenças em linguagem natural
12 p = "Chove em Curitiba"
13 q = "uso guarda-chuva"
14
15 sentenca1 = f"{p} E {q}"
16 sentenca2 = f"Tweety é um pássaro OU Tweety é um mamífero"
17 sentenca3 = f"Se {p} ENTÃO {q}"
18 sentenca4 = f"É falso que Tweety é um mamífero"
19
20 # Traduzindo as sentenças
21 print(traduzir_para_logica(sentenca1)) # Output: p and q
22 print(traduzir_para_logica(sentenca2)) # Output: r or s
23 print(traduzir_para_logica(sentenca3)) # Output: r -> u
24 print(traduzir_para_logica(sentenca4)) # Output: ~u
25
26
```

A saída esperada para as sentenças colocadas no código acima seriam:

P and Q

R or S

$R \rightarrow U$

$\sim U$

No entanto, como a conversão não obteve sucesso, a saída foi a seguinte:

```
marta@marta-Aspire-A315-53:~/logic-project/logic-project$ /bin/python3 /home/marta/logic-project/logic-project/conversao.py
Chove em Curitiba and uso guarda-chuva
Tweety é um pássaro or Tweety é um mamífero
Se Chove em Curitiba andNTÃO uso guarda-chuva
~ Tweety é um mamífero
marta@marta-Aspire-A315-53:~/logic-project/logic-project$
```