Título: Sistema Automatizado de Verificação de Argumentos Lógicos

Descrição Geral:

Desenvolva um sistema em Python que receba argumentos em lógica proposicional ou de predicados e determine se são válidos ou inválidos. O sistema deve implementar diferentes técnicas de prova e fornecer justificativas para suas conclusões.

ESPECIFICAÇÃO DETALHADA

Parte 1: Lógica Proposicional

Entrada: Argumentos em lógica proposicional usando notação textual

Formato de entrada:

```
Premissas:
P -> Q
P
---
Conclusão:
Q
```

Requisitos:

1. **Parser de Fórmulas**
    - Interpretar operadores: ~ (negação), ∧ ou & (conjunção), ∨ ou | (disjunção), -> (implicação), <-> (bicondicional)
    - Tratar parênteses corretamente
    - Validar sintaxe da entrada
2. **Verificador por Tabela Verdade**
    - Gerar todas as combinações possíveis de valores
    - Verificar se: $(P_1 \land P_2 \land \ldots \land P_n) \rightarrow C$ é tautologia
    - Exibir a tabela verdade completa
3. **Identificador de Forma de Argumento**
    - Reconhecer formas válidas: Modus Ponens, Modus Tollens, Silogismo Hipotético, etc.
    - Identificar falácias comuns: Afirmação do Consequente, Negação do Antecedente

Exemplo de execução:

```
Entrada:
Premissa 1: P -> Q
Premissa 2: Q -> R
Premissa 3: P
Conclusão: R

Saída:
✓ ARGUMENTO VÁLIDO
Método: Tabela Verdade
Forma: Silogismo Hipotético + Modus Ponens
Justificativa: Em todas as linhas onde as premissas são verdadeiras,
a conclusão também é verdadeira.

Tabela Verdade:
P | Q | R | P->Q | Q->R | P | (P∧Q∧R) | R | Válido?
V | V | V |  V   |  V   | V |    V    | V |   ✓
V | V | F |  V   |  F   | V |    F    | F |   -
...
```

Parte 2: Lógica de Predicados - Nível Básico (Obrigatória - 30%)

Entrada: Argumentos simples em lógica de predicados

Formato:

```
Domínio: {a, b, c}
Premissas:
(∀x)P(x)
Conclusão:
P(a)
```

Parte 2: Lógica de Predicados - Nível Básico

Entrada: Argumentos simples em lógica de predicados

Formato:

Domínio: {a, b, c}

Premissas:

(∀x)P(x)

Conclusão:

P(a)

Requisitos:

1. **Parser de Quantificadores**
    - Interpretar (∀x), (∃x)

- Identificar predicados: P(x), Q(x, y), etc.
- Tratar variáveis e constantes

2. **Verificador com Domínio Finito**
   - Para domínios pequenos, enumerar todas as interpretações
   - Verificar validade por força bruta
   - Exibir contraexemplos quando inválido
3. **Aplicador de Regras Básicas**
   - Particularização Universal
   - Generalização Universal
   - Particularização Existencial
   - Generalização Existencial

Exemplo:

```
Entrada:
Domínio: {1, 2, 3}
Premissa 1: (∀x)P(x)
Conclusão: (∃x)P(x)

Saída:
✓ ARGUMENTO VÁLIDO
Método: Enumeração em domínio finito
Regra aplicada: Se (∀x)P(x) então P(c) para qualquer c,
logo (∃x)P(x)

Verificação:
Se P(1)=V, P(2)=V, P(3)=V (para satisfazer ∀xP(x))
Então existe pelo menos um x com P(x)=V ✓
```

Parte 3: Técnicas Avançadas (Opcional - 30% extra)

Escolha pelo menos UMA das opções:

Opção A: Método de Resolução

1. Converter para Forma Clausal
2. Aplicar regra de resolução
3. Detectar cláusula vazia (prova por contradição)

```
Entrada:
Premissa: (∀x)[P(x) -> Q(x)]
Premissa: P(a)
Conclusão: Q(a)

Saída:
Conversão para cláusulas:
1. ¬P(x) v Q(x)  [de ∀x(P(x)->Q(x))]
2. P(a)          [premissa]
3. ¬Q(a)         [negação da conclusão]

Aplicando Resolução:
4. Q(a)          [resolução de 1 e 2, com x=a]
5. □             [resolução de 3 e 4 - CONTRADIÇÃO]

✓ Argumento VÁLIDO (prova por contradição)
```

Opção B: Sistema de Dedução Natural

1. Implementar regras de inferência
2. Buscar sequência de aplicações
3. Construir árvore de prova

```
Entrada:
Premissas: P -> Q, Q -> R, P
Conclusão: R

Saída - Árvore de Prova:
1. P -> Q        [Premissa]
2. Q -> R        [Premissa]
3. P             [Premissa]
4. Q             [Modus Ponens: 3, 1]
5. R             [Modus Ponens: 4, 2]
✓ Q.E.D.
```

Opção C: Forma Normal Prenex e Skolemização

1. Converter para FNP
2. Aplicar Skolemização
3. Verificar equivalência

```
Entrada:
(∀x)P(x) -> (∃y)Q(y)

Saída:
Passo 1 - Eliminar implicação:
¬(∀x)P(x) v (∃y)Q(y)

Passo 2 - Mover negação:
(∃x)¬P(x) v (∃y)Q(y)

Passo 3 - Forma Normal Prenex:
(∃x)(∃y)[¬P(x) v Q(y)]

Passo 4 - Skolemização:
¬P(c₁) v Q(c₂)
onde c₁, c₂ são constantes de Skolem
```

CASOS DE TESTE OBRIGATÓRIOS

Lógica Proposicional:

1. **Modus Ponens**

P -> Q, P |- Q (VÁLIDO)

2. **Falácia da Afirmação do Consequente**

P -> Q, Q |- P (INVÁLIDO)

3. **Silogismo Disjuntivo**

P ∨ Q, ~P |- Q (VÁLIDO)

4. **Dilema Construtivo**

(P->Q) ∧ (R->S), P ∨ R |- Q ∨ S (VÁLIDO)

Lógica de Predicados:

5. **Particularização Universal**

(∀x)P(x) |- P(a) (VÁLIDO)

6. **Generalização Existencial**

P(a) |- (∃x)P(x) (VÁLIDO)

7. **Silogismo de Aristóteles**

(∀x)[H(x)->M(x)], H(s) |- M(s) (VÁLIDO)

8. **Argumento Inválido**

(∃x)P(x) ∧ (∃x)Q(x) |- (∃x)[P(x)∧Q(x)] (INVÁLIDO)

9. **Quantificadores Aninhados**

(∀x)(∃y)P(x,y) |- (∃y)(∀x)P(x,y) (INVÁLIDO)

10. **Equivalência de De Morgan**

~(∀x)P(x) |- (∃x)~P(x) (VÁLIDO)

CRITÉRIOS DE AVALIAÇÃO

Funcionalidades Básicas (70%):

- [ ] Parser funcional para lógica proposicional (10%)
- [ ] Validação por tabela verdade (15%)
- [ ] Identificação de formas de argumento (10%)
- [ ] Parser para lógica de predicados (10%)
- [ ] Validação em domínio finito (15%)
- [ ] Aplicação de regras básicas (10%)

Qualidade do Código (15%):

- [ ] Código organizado em módulos (5%)
- [ ] Documentação adequada (5%)
- [ ] Tratamento de erros (5%)

Interface e Usabilidade (15%):

- [ ] Interface clara (terminal ou GUI) (5%)
- [ ] Mensagens de erro informativas (5%)
- [ ] Exemplos e ajuda integrados (5%)

Extras (até 30%):

- [ ] Implementação de método de resolução (15%)
- [ ] Sistema de dedução natural (15%)
- [ ] Conversão para FNP e Skolemização (10%)
- [ ] Interface gráfica (10%)
- [ ] Exportação de provas em LaTeX (5%)
- [ ] Testes unitários abrangentes (5%)

RECURSOS PARA PESQUISA

Lógica:

1. "Logic for Computer Science" - Huth & Ryan
2. "Mathematical Logic for Computer Science" - Ben-Ari
3. Stanford Encyclopedia of Philosophy - Logic

Python:

1. Biblioteca sympy.logic para inspiração
2. pyparsing para construir parser
3. networkx para árvores de prova
4. tkinter ou PyQt para GUI

Algoritmos:

1. DPLL algorithm (satisfatibilidade)
2. Resolution algorithm
3. Tableau method
4. Natural deduction systems

EXEMPLO DE INTERAÇÃO COMPLETA

=== VERIFICADOR DE ARGUMENTOS LÓGICOS ===

Escolha o tipo de lógica:

1. Proposicional

2. Predicados

> 2

Entre com o domínio (separado por vírgulas) ou deixe vazio para domínio infinito:

> 1, 2, 3

Quantas premissas?

> 2

Premissa 1: (∀x)[P(x) -> Q(x)]

Premissa 2: P(1)

Conclusão: Q(1)

Analisando...

✓ ARGUMENTO VÁLIDO

Método utilizado: Particularização Universal + Modus Ponens

Demonstração:

1. (∀x)[P(x) -> Q(x)] [Premissa 1]

2. P(1) [Premissa 2]

3. P(1) -> Q(1) [Particularização Universal de 1 com x=1]

4. Q(1) [Modus Ponens: 2, 3]

Q.E.D.

Verificação em domínio {1,2,3}:

Testando todas as interpretações possíveis...

✓ Em todas as interpretações onde as premissas são verdadeiras,

a conclusão também é verdadeira.

Deseja:

1. Ver tabela de interpretações

2. Exportar prova em LaTeX

3. Tentar outro argumento

4. Sair

>

DICAS DE IMPLEMENTAÇÃO

Fase 1:

1. Comece com lógica proposicional simples
2. Implemente parser recursivo descendente
3. Use força bruta para validação inicial

Fase 2:

1. Adicione predicados com 1 argumento
2. Implemente domínios finitos pequenos
3. Adicione regras de inferência básicas

Fase 3:

1. Suporte a predicados com múltiplos argumentos

2. Implemente algoritmos mais eficientes
3. Adicione técnicas avançadas (resolução, etc.)

ENTREGÁVEIS

1. **Código fonte** (todos os arquivos .py)
2. **Documentação** (README.md com instruções)
3. **Relatório técnico** (3-5 páginas explicando decisões de design)
4. **Casos de teste** (arquivo com exemplos e resultados esperados)
5. **Vídeo demonstração** (5-10 minutos mostrando funcionamento)