INE5416 Paradigmas de Programação

Prof. A. G. Silva

25 de agosto de 2016

Prova de teoremas (I)

- Cálculo de predicados como método para expressar coleções de proposições
- Determinação se quaisquer fatos interessantes ou úteis podem ser inferidos a partir delas
- Análogo ao trabalho de matemáticos na busca de novos teroremas, inferidos a partir de axiomas e teoremas conhecidos
- Os primeiros dias da ciência da computação (anos 50 e início dos 60) deslumbraram a automatização do processo de prova de teoremas

Prova de teoremas (II)

- A partir da introdução, por Robinson e Smullyan (1960), de procedimentos eficientes para demonstração automática de teoremas por computador, a lógica passou a ser estudada também como método computacional para a solução de problemas (Padronização das fórmulas lógicas em Formas Normais Canônicas)

- Grande avanço com a introdução do método da resolução por Alan Robinson (1965) da Universidade de Syracuse
  - Método de **refutação** - para provar um teorema a partir de um conjunto de hipóteses, nega-se o teorema e prova-se que o conjunto de expressões formado pelas hipóteses e pelo teorema negado é contraditório
  - Parte da transformação da expressão a ser provada para a **forma normal conjuntiva** ou **forma clausal**
  - Utiliza uma regra de inferência única, chamada **regra de resolução** e um algoritmo de casamento de padrões chamado **algoritmo de unificação**

Prova de teoremas (III)

- A **resolução** é uma regra de inferência que permite às proposições inferidas serem computadas a partir de proposições dadas
  - Método com aplicação potencial para a prova automática de teoremas
  - Desenvolvida para proposições na forma clausal

- Suponha que existam duas proposições com as formas
  $P_1 \leftarrow P_2$
  $Q_1 \leftarrow Q_2$

  Suponha que P_1 é idêntica a Q_2, renomeados como T então:
  $T \leftarrow P_2$
  $Q_1 \leftarrow T$

  Processo de inferência ou resolução:
  $Q_1 \leftarrow P_2$

Formalização de argumentos

- Um argumento é uma sequência de premissas seguida de uma conclusão

- Exemplo:
  - *Se neva, então faz frio.*
  - *Está nevando.*
  - **Logo**, *está fazendo frio.*

- Vocabulário:
  - p : *"neve"*
  - q : *"frio"*

- Formalização ($\models$ é a consequência lógica):
  - $\{ \ p \rightarrow q, \ p \ \} \quad \models \quad q$

Validação de argumentos (I)

- Nem todo argumento é válido!
- Exemplo: Intuitivamente, qual dos argumentos a seguir é válido?
- Argumento 1:
  - *Se eu fosse artista, então eu seria famoso.*
  - *Não sou famoso.*
  - *Logo, não sou artista.*
- Argumento 2:
  - *Se eu fosse artista, então eu seria famoso.*
  - *Sou famoso.*
  - *Logo, sou artista.*

Validação de argumentos (II)

- Um argumento é **válido** se a sua conclusão é uma **consequência lógica** de suas premissas, ou seja, a veracidade da conclusão está implícita na veracidade das premissas.

- Três métodos de validação de argumentos:
  - **Tabela-verdade** (semântico)
  - **Prova por dedução** (sintático)
  - **Prova por refutação** (sintático)

- Métodos semânticos são baseados em interpretações

- Métodos sintáticos são baseados em regras de inferência (raciocínio)

Validação de argumentos – Tabela-verdade (I)

- Um argumento da forma $\{\alpha_1, \ldots, \alpha_n\} \models \beta$ é válido se e somente se a fórmula correspondente $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta$ é válida (tautológica).

- **Exemplo 1** – Argumento:
  - *Se eu fosse artista, seria famoso.*
  - *Não sou famoso.*
  - *Logo, não sou artista.*

- Vocabulário:
  - p : *"artista"*
  - q : *"famoso"*

- Formalização:
  - $\{\ p \rightarrow q,\ \neg q\ \} \models \neg p$

p	q	(p	→	q)	∧	¬	q	→	¬	p
F	F	F	V	F	V	V	F	V	V	F
F	V	F	V	V	F	F	V	V	V	F
V	F	V	F	F	F	V	F	V	F	V
V	V	V	V	V	F	F	V	V	F	V

o argumento é válido

Validação de argumentos – Tabela-verdade (II)

- **Exemplo 2** – Argumento:
  - *Se eu fosse artista, seria famoso.*
  - *Sou famoso.*
  - *Logo, sou artista.*

- Vocabulário:
  - p : *"artista"*
  - q : *"famoso"*

- Formalização:
  - $\{ p \to q, \ q \} \models p$

p	q	(p	→	q)	∧	q	→	p
F	F	F	V	F	F	F	V	F
F	V	F	V	V	V	V	F	F
V	F	V	F	F	F	F	V	V
V	V	V	V	V	V	V	V	V

o argumento **não** é válido

Validação de argumentos – Tabela-verdade (III)

- **Exercício** – Sócrates está disposto a visitar Platão ou não?
  - *Se Platão está disposto a visitar Sócrates, então Sócrates está disposto a visitar Platão. Por outro lado, se Sócrates está disposto a visitar Platão, então Platão não está disposto a visitar Sócrates; mas se Sócrates não está disposto a visitar Platão, então Platão está disposto a visitar Sócrates.*
- Vocabulário:
  - p : *"Platão está disposto a visitar Sócrates"*
  - s : *"Sócrates está disposto a visitar Platão"*
- Formalização:
  - $\{\ p \rightarrow s,\ (s \rightarrow \neg p) \wedge (\neg s \rightarrow p)\ \}$

Validação de argumentos – Tabela-verdade (IV)

- Consequência lógica é o elo entre o que um agente "acredita" e aquilo que é explicitamente representado em sua base de conhecimento
- A tabela-verdade é um método semântico que permite verificar consequências lógicas.
- Este método tem a vantagem de ser conceitualmente simples; porém, como o número de linhas na tabela-verdade cresce exponencialmente em função do número de símbolos proposicionais na fórmula, na prática, seu uso nem sempre é viável.
- Assim, adota-se **raciocínio automatizado** como uma alternativa mais eficiente para verificação de consequência lógica (isto é, validação de argumentos)

Prova por dedução (I)

- Uma **prova por dedução** de uma fórmula φ, a partir de uma base de conhecimento Δ, é uma sequência finita de fórmulas $\gamma_1, \ldots \gamma_k$ tal que:
  - $\gamma_k = \varphi$;
  - para $1 \leq i \leq k$, ou $\gamma_i \in \Delta$, ou então γ_i é **derivada** de fórmulas em $\Delta \cup \{\gamma_1, \ldots, \gamma_{i-1}\}$, pela aplicação de uma **regra de inferência**.

- Regra de inferência é um padrão de manipulação sintática que define como uma fórmula (*conclusão*) pode ser derivada de outras fórmulas (*premissas*).

Prova por dedução (II)

- Regras de inferência clássicas:
  - **Modus ponens (MP):** $\{\ \alpha \rightarrow \beta,\ \alpha\ \} \vdash\ \beta$
  - **Modus tollens (MT):** $\{\ \alpha \rightarrow \beta,\ \neg\beta\ \} \vdash\ \neg\alpha$
  - **Silogismo hipotético (SH):** $\{\ \alpha \rightarrow \beta,\ \beta \rightarrow \gamma\ \} \vdash\ \alpha \rightarrow \gamma$
    - Representam "esquemas de raciocínio" válidos
    - Podemos validar estes esquemas usando tabela-verdade
    - Podem ser usadas para derivar conclusões que são consequências lógicas de suas premissas

Prova por dedução (III)

- **Exemplo:** validar o argumento $\{ j \rightarrow g, \; \neg j \rightarrow t, \; g \rightarrow c, \; \neg c \} \models t$

1. $j \rightarrow g$ $\quad \Delta$
2. $\neg j \rightarrow t$ $\quad \Delta$
3. $g \rightarrow c$ $\quad \Delta$
4. $\neg c$

---

5. $j \rightarrow c$ $\quad$ SH(1,3)
6. $\neg j$ $\quad$ MT(5,4)
7. t $\quad$ MP(2,6)

- **Conclusão:** o argumento é válido, pois a fórmula t pode ser derivada de Δ.

Prova por refutação (I)

- Embora a prova por dedução seja um método mais prático que a tabela-verdade, ainda é muito difícil obter algoritmos eficientes para validação de argumentos com base neste método.
- **Refutação** é um processo em que se demonstra que uma determinada hipótese contradiz uma base de conhecimento.
- Uma base de conhecimento $\Delta = \{\alpha_1, \ldots, \alpha_n\}$ é consistente se a fórmula correspondente $\alpha_1 \wedge \cdots \wedge \alpha_n$ é satisfatível.
- Se $\Delta = \{\alpha_1, \ldots, \alpha_n\}$ é consistente, provar $\Delta \models \gamma$ equivale a mostrar que o conjunto de fórmulas $\{\alpha_1 \wedge \cdots \wedge \alpha_n, \neg\gamma \}$ é inconsistente.

Prova por refutação (II)

- **Argumento**
  1. Se o time joga bem, então ganha o campeonato.
  2. Se o time não joga bem, então o técnico é culpado.
  3. Se o time ganha o campeonato, então os torcedores ficam contentes.
  4. Os torcedores não estão contentes.
  5. Logo, o técnico é culpado.

- **Refutação**

(a)	O técnico não é culpado	hipótese
(b)	O time joga bem	MT(a,2)
(c)	O time ganha o campeonato	MP(b,1)
(d)	Os torcedores ficam contentes	MP(c,3)
(e)	Contradição!	Confrontando (d) e (4)

Prova por refutação (III)

- **Exemplo:** validar o argumento $\{\ j \to g,\ \neg j \to t,\ g \to c,\ \neg c\ \} \models t$

1. $j \to g$ Δ
2. $\neg j \to t$ Δ
3. $g \to c$ Δ
4. $\neg c$ Δ

---

5. $\neg t$ Hipótese
6. j MT(5,2)
7. g MP(6,1)
8. c MP(7,3)
9. $\square$ Contradição!

- **Conclusão:** como $\Delta \cup \{\neg t\}$ é inconsistente, segue que $\Delta \models t$.

Forma normal conjuntiva (I)

- Para simplificar a automatização do processo de refutação, vamos usar **fórmulas normais** (Forma Normal Conjuntiva – FNC).

- Passos para conversão para FNC:
  - Elimine a implicação:
    $\alpha \rightarrow \beta \;\equiv\; \neg\alpha \vee \beta$
  - Reduza o escopo da negação:
    $\neg(\alpha \wedge \beta) \;\equiv\; \neg\alpha \vee \neg\beta$
    $\neg(\alpha \vee \beta) \;\equiv\; \neg\alpha \wedge \neg\beta$
  - Reduza o escopo da disjunção:
    $\alpha \vee (\beta \wedge \gamma) \;\equiv\; (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Forma normal conjuntiva (II)

- Exemplo de conversão para FNC:
  - $p \vee q \;\rightarrow\; r \wedge s$
  - $\equiv\; \neg(p \vee q) \;\vee\; (r \wedge s)$
  - $\equiv\; (\neg p \wedge \neg q) \;\vee\; (r \wedge s)$
  - $\equiv\; ((\neg p \wedge \neg q) \vee r) \;\wedge\; ((\neg p \wedge \neg q) \vee s)$
  - $\equiv\; (\neg p \vee r) \;\wedge\; (\neg q \vee r) \;\wedge\; (\neg p \vee s) \;\wedge\; (\neg q \vee s)$ **(FNC)**
  - **Fórmulas normais:** $\{\; \neg p \vee r,\; \neg q \vee r,\; \neg p \vee s,\; \neg q \vee s \;\}$

Inferência por resolução (I)

- FNC permite usar inferência por resolução:
- A idéia da resolução é:
  - $RES(\alpha \vee \beta,\ \neg\beta \vee \gamma) \ = \ \alpha \vee \gamma$
  - $RES(\alpha,\ \neg\alpha) \ = \ \square$
- Equivalência entre resolução e regras de inferência clássicas:

$MP(\alpha \rightarrow \beta,\ \alpha) \ = \ \beta$	$RES(\neg\alpha \vee \beta,\ \alpha) \ = \ \beta$
$MT(\alpha \rightarrow \beta,\ \neg\beta) \ = \ \neg\alpha$	$RES(\neg\alpha \vee \beta,\ \neg\beta) \ = \ \neg\alpha$
$SH(\alpha \rightarrow \beta,\ \beta \rightarrow \gamma) \ = \ \alpha \rightarrow \gamma$	$RES(\neg\alpha \vee \beta,\ \neg\beta \vee \gamma) \ = \ \neg\alpha \vee \gamma$

Inferência por resolução (II)

- **Exemplo:** validar o argumento $\{\ j \rightarrow g,\ \neg j \rightarrow t,\ g \rightarrow c,\ \neg c\ \} \models t$

1. $\neg j \lor g$ Δ
2. $j \lor t$ Δ
3. $\neg g \lor c$ Δ
4. $\neg c$ Δ

---

5. $\neg t$ Hipótese
6. j RES(5,2)
7. g RES(6,1)
8. c RES(7,3)
9. $\square$ RES(8,4)

- **Conclusão:** como $\Delta \cup \{\neg t\}$ é inconsistente, segue que $\Delta \models t$.

Inferência por resolução (III)

- **Exercício:** Prove usando refutação e inferência por resolução.
  - Se o programa possui erros de sintaxe, sua compilação produz uma mensagem de erro.
  - Se o programa não possui erros de sintaxe, sua compilação produz um programa executável.
  - Se tivermos um programa executável, podemos executá-lo para obter um resultado.
  - Não temos como executar o programa para obter um resultado.
  - Logo, a compilação do programa produz uma mensagem de erro.

Cálculo de predicados – Prova de teoremas (I)

- O processo de determinação de valores úteis para variáveis é a **unificação**
- A atribuição temporária de valores a variáveis é chamada de **instanciação**
- É comum instanciar uma variável com um valor, falhar em completar o casamento e então ser necessária uma volta – rastreamento para trás ou *backtracking* – para instanciá-la com valor diferente

Cálculo de predicados – Prova de teoremas (II)

- Propriedade criticamente importante da resolução: habilidade de detectar inconsistência em um conjunto de proposições
- Essa propriedade permite que a resolução seja usada para provar teoremas:
  - Em termos de cálculo de predicados como um conjunto de proposições pertinentes
  - Com a negação do próprio teorema iniciando como a nova proposição (por contradição)
  - Proposições originais são **hipóteses**
  - A negação do teorema é o **objetivo**
- Teoricamente válido e útil; finito quanto ao conjunto de proposições finito; porém o tempo para encontrar um inconsistência em uma grande base de proposições pode ser imenso

Cálculo de predicados – Prova de teoremas (III)

- Proposições usadas para resolução exigem apenas um tipo restrito de forma clausal
- Tipos especiais de proposições, chamados de **cláusulas de Horn** – de Alfred Horn (1951) –, podem ser de apenas duas formas:
  - Uma única proposição atômica do lado esquerdo (com cabeça ou *head*)
  - Um lado esquerdo vazio (sem cabeça)
- *Cláusulas de Horn com cabeça* – definição de **relacionamentos**:
  likes(bob,trout) $\leftarrow$ likes(bob,fish) $\wedge$ fish(trout)
- *Cláusulas de Horn sem cabeça* – definição de **fatos**:
  father(bob,jake)

Variáveis I

- As variáveis em Prolog servem para responder questões mais elaboradas
- Exemplos:
  - "Do que Maria gosta?"
  - "Quem mora em Atenas?"
- Variáveis distinguem-se dos objetos por terem nomes iniciados com letra maiúscula
- Considere o seguinte banco de dados:

```
gosta(maria, flores).
gosta(maria, pedro).
gosta(paulo, maria).
```

Se fizermos a pergunta:

```
?- gosta(maria, X).
```

Variáveis II

- Perguntamos “Do que Maria gosta?”. Prolog responde

  `X = flores`

  e fica esperando por mais instruções
- Novamento, buscam-se fatos que unifiquem com o fato da pergunta
- A diferença é que uma variável está presente agora
- Variáveis não instanciadas, como é o caso de `X` inicialmente, unificam com qualquer termo
- Prolog examina os fatos na ordem em que aparecem no banco de dados
- Portanto, `gosta(maria, flores)` é econtrado primeiro
- A variável `X` unifica com `flores` e, a partir deste momento, passa a ser <u>instanciada</u> com este temo
- Prolog também <u>marca</u> a posição no banco de dados onde a unificação ocorreu

Variáveis III

- Quando um fato que unifica com uma pergunta é encontrado, Prolog mostra os objetos que as variáveis guardam
- No exemplo, só há uma variável que foi unificada
- Se teclarmos ENTER, isto será interpretado que estamos satisfeitos com a resposta e a busca é interrompida
- Se teclarmos um ponto-e-virgula (;) seguido de ENTER, Prolog continua sua busca do ponto onde tinha parado, tentando outra resposta à pergunta
- Em vez de começar do início do banco de dados, está tentando ressatisfazer a pergunta

Conjunções I

- Questões mais complexas podem ser feitas com conjunções
- Problema com metas separadas que Prolog deve tentar satisfazer uma a uma

  ```
  gosta(maria, chocolate).
  gosta(maria, vinho).
  gosta(pedro, vinho).
  gosta(pedro, maria).
  ```

  Segue uma pergunta que significa "Pedro gosta de Maria <u>e</u> Maria gosta de Pedro?":

  ```
  ?- gosta(pedro, maria), gosta(maria, pedro).
  ```

  A vírgula é pronunciada "e" e serve separar um número qualquer de metas diferentes
- No exemplo, a resposta será "não". Embora a primeira meta seja um fato, a segunda não pode ser provada

Conjunções II

- Conjunções combinadas com variáveis
- Exemplo: "Há algo de que ambos Maria e Pedro gostam?"

```
?- gosta(maria, X), gosta(pedro, X).
```

- Caso Prolog consiga satisfazer a primeira meta, manterá uma marca no banco de dados no ponto em que houve unificação e tenta a segunda meta
- Se a segunda meta também for satisfeita, Prolog coloca uma outra marca para a segunda meta
- Cada meta tem sua própria marca no banco de dados
- Se a segunda meta falhar, Prolog tentará ressatisfazer a meta anterior (neste caso, a primeira) a partir da marca esta meta.

Backtracking I

1. A primeira meta encontra o fato unificador `gosta(maria, chocolate).` A variável `X` é instanciada a `chocolate` em todas as ocorrência de `X` na pergunta. Prolog marca esta posição para a primeira meta e a instanciação de `X`

2. A segunda meta, que virou `gosta(pedro, chocolate)`, devido à instanciação de `X`, não unifica com nada no banco de dados, e por isso a meta falha. Prolog então tentará ressatisfazer a meta anterior do ponto onde esta parou no banco de dados e desfazeno instanciações associadas

3. Na ressatisfação da primeira meta, o próximo fato unificador é `gosta(maria, vinho).` Prolog move a marca para a nova posição e instancia `X` a `vinho`

Backtracking II

4. Prolog tenta a próxima meta, que agora é `gosta(pedro, vinho).` Esta não é uma ressatisfação, mas sim uma meta inteiramente nova, e portanto a busca é feita a partir do início do banco de dados. Esta nova meta é satisfeita e Prolog coloca a marca desta meta no fato unificador.

5. Todas as metas da pergunta estão satisfeitas agora. Prolog imprime as instanciações de variáveis:

   ```
   X = vinho
   ```

   e aguarda instruções

Regras I

- Uma regra é uma afirmação geral sobre objetos e seus relacionamentos
- Por exemplo, suponho que queremos representar a seguinte dependência entre fatos:
  Pedro gosta de todo mundo que gosta de vinho
  o que pode ser reescrito como:
  Pedro gosta de `X` **se** `X` **gosta de vinho.**
- Em Prolog, regras consistem de uma cabeça e um corpo.
- A cabeça e o corpo são conectado pelo símbolo ":-" formado por dois pontos e hífen
- O ":-" pronuncia-se "se"
- O exemplo seria escrito:

```
gosta(pedro, X) :- gosta(X, vinho).
```

Regras II

- Podemos tornar Pedro mais exigente sobre o que ele gosta adicionando mais metas ao corpo da regra. Exemplo, “Pedro gosta de qualquer um que goste de vinho e de chocolate”:

  ```
  gosta(pedro, X) :- gosta(X, vinho), gosta(X, chocolate).
  ```

- Ou então, supondo que Pedro gosta de mulheres que gostam de vinho:

  ```
  gosta(pedro, X) :- mulher(X), gosta(X, vinho).
  ```

  Note que a mesma variável `X` ocorre três vez na regra. Dizemos que o escopo de `X` é a regra toda. Isto significa que, quando `X` for instanciada, as três ocorrências terão o mesmo valor

Regras III

- Uma meta unifica com uma regra se ela unifica com a cabeça da regra.
- Agora, para verificar a veracidade da regra, o corpo é usado
- Diferentemente dos fatos, onde basta haver unificação para que a meta seja sarisfeita, no caso de uma regra a unificação na verdade transfere a verificação da satisfação para a conjunção de metas que formam o corpo da regra
- Vamos ilustrar este procedimento com nosso próximo exemplo, que envolve a família da rainha Vitória. Usaremos o predicado pais com três argumentos tal que `pais(X, Y, Z)` significa que "os pais de `X` são `Y` e `Z`".
- O segundo argumento é a mãe e o terceiro é o pai de `X`. Usaremos também os predicados mulher e homem para indicar o sexo das pessoas.

Regras IV

- Banco de dados

```
homem(alberto).
homem(eduardo).
mulher(alice).
mulher(vitoria).
pais(eduardo, vitoria, alberto).
pais(alice, vitoria, alberto).
```

- Definiremos agora o predicado `irma_de` tal que `irma_de(X, Y)` seja satisfeito quando `X` for irmã de `Y`. Dizemos que `X` é irmã de `Y` quando
  - `X` é mulher
  - `X` tem mãe `M` e pai `P`, e
  - `Y` tem os mesmos pais de `X`

Regras V

- Em Prolog:

```
irma_de(X, Y) :-
        mulher(X),
        pais(X, M, P),
        pais(Y, M, P).
```

- Se perguntarmos:

```
?- irma_de(alice, eduardo).
```

Regras VI

Processamento da pergunta: `?- irma_de(alice, eduardo).`

N	Meta	Marca	Variáveis
1	irma_de(alice, eduardo)	irma_de regra 1	X_1 = alice, Y_1 = eduardo
1.1	mulher(alice)	mulher fato 1	-
1.2	pais(alice, M_1, P_1)	pais fato 2	M_1 = vitoria, P_1 = alberto
1.3	pais(eduardo, vitoria, alberto)	pais fato 1	-

Processamento da meta: `irma_de(alice, X).`

N	Meta	Marca	Variáveis
1	irma_de(alice, X)	irma_de regra 1	X_1 = alice, Y_1 = X
1.1	mulher(alice)	mulher fato 1	-
1.2	pais(alice, M_3, P_3)	pais fato 2	M_1 = vitoria, P_1 = alberto
1.3	pais(Y_1, vitoria, alberto)	pais fato 1	Y_1 = eduardo

Exercícios

- Suponha que queiramos saber de quem Alice é irmã
- A pergunta adequada seria

  ```
  ?- irma_de(alice, X).
  ```

- O que ocorre então na segunda tabela, onde `X` sem índice indicará a varíavel da pergunta.
- Observe que $X=Y_1=eduardo$ e portanto Prolog responde:

  ```
  X = eduardo
  ```

  e fica aguardando novas instruções. O que acontecerá se pedirmos respostas alternativas?
- Descreva o que acontece se forem pedidas respostas alternativas no exemplo envolvendo o predicado `irma_de` acima. Este é o comportamento esperado? Como consertar a regra, supondo que existe um predicado `dif(X, Y)` que é satisfeito quando `X` e `X` são diferentes?

Trabalho 1 – Parte B (entrega pelo Moodle)

- Enunciado:

  https://www.inf.ufsc.br/~alexandre.silva/courses/16s2/ine5416/exercicios/t1B.pdf

Referências

- SEBESTA, Robert W. Conceitos de Linguagens de Programação. 5a. Ed. Porto Alegre: Bookman, 2003.
- Notas dos Prof. Joao Meidanis (IC/UNICAMP), Prof. Silvio do Lago Pereira (DTI/FATEC) e Profa. Jerusa Marchi (INE/UFSC)