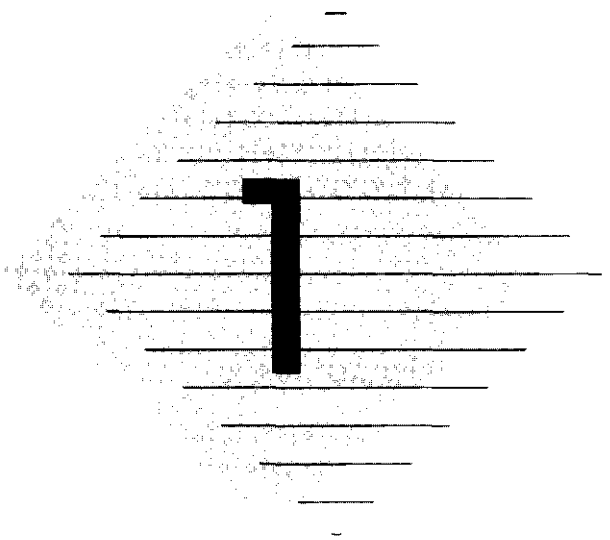


# Lógica Formal

## Objetivos do Capítulo

Após estudar este capítulo, o leitor deve ser capaz de:

- Reconhecer e trabalhar com os símbolos formais que são usados nas lógicas proposicional e de predicados
- Achar o valor-verdade de uma expressão na lógica proposicional
- Achar o valor-verdade de alguma interpretação de uma expressão na lógica de predicados
- Usar a lógica de predicados para representar sentenças da língua portuguesa
- Construir demonstrações formais nas lógicas proposicional e de predicados, e usá-las para determinar a validade de um argumento da língua portuguesa
- Entender como a linguagem de programação Prolog é constituída em função da lógica de predicados
- Provar matematicamente a correção de programas que usam comandos de atribuição e comandos condicionais

A exemplo de qualquer outra ciência, a Ciência da Computação depende da Matemática para obter um vocabulário preciso, uma notação poderosa, abstrações úteis e um raciocínio rigoroso. O objetivo deste livro é melhorar nosso entendimento da linguagem, das ferramentas e dos processos de raciocínio da Matemática que são usados na Ciência da Computação.

Este capítulo introduz a Lógica Formal, que delineia o método organizado e cuidadoso de pensar que caracteriza qualquer investigação científica ou qualquer outra atividade de raciocínio. Além disso, a

lógica formal tem aplicações diretas na Ciência da Computação. A última seção deste capítulo explora uma linguagem baseada na lógica e no uso da Lógica Formal objetivando verificar a correção de programas de computadores. Ainda, a lógica de circuitos (a lógica que rege os circuitos de computadores) é um análogo direto da lógica de sentenças deste capítulo. Estudaremos este tipo de lógica no Cap. 7.

## Seção 1.1 Sentenças, Representação Simbólica e Tautologias

Geralmente nos expressamos, em português, através de interrogações e exclamações, mas, para comunicar fatos ou informações, usamos sentenças. Tecnicamente, uma **sentença** (ou **proposição)** é uma frase que pode ser apenas verdadeira ou falsa.

**EXEMPLO 1** Considere o seguinte:

a. Dez é menor do que sete.
b. Como vai você?
c. Ela é muito talentosa.
d. Existem formas de vida em outros planetas do universo.

A frase (a) é uma sentença porque é falsa. Como o item (b) é uma pergunta, não pode ser considerado nem verdadeiro nem falso. Não tem valor-verdade e, portanto, não é uma sentença. Na frase (c) a palavra *ela* é uma variável e a frase não é verdadeira nem falsa, pois *ela* não está especificada; portanto, (c) não é uma sentença. A frase (d) é uma sentença porque é verdadeira ou falsa; independentemente de sermos capazes de decidir qual dos dois.

### Conectivos e Valores-Verdade

Para enriquecermos nossas conversas não nos limitamos ao uso de simples sentenças. Ao contrário, as combinamos com o uso de conectivos a fim de criarmos sentenças compostas, cujo valor-verdade depende dos valores-verdade de cada sentença que o compõe e dos conectivos usados. Um conectivo comum é a palavra *e.* (Palavras como *mas* e *também* expressam diferentes significados, mas têm o mesmo efeito sobre os valores-verdade.) Se combinarmos as duas sentenças verdadeiras "Elefantes são grandes" e "Bolas são redondas" devemos considerar a sentença resultante "Elefantes são grandes e bolas são redondas" como verdadeira. Na lógica, usamos o símbolo A para denotar o conectivo lógico *e* e letras maiúsculas para denotar as sentenças (ao que chamaremos de símbolos proposicionais). Valores-verdade são atribuídos aos símbolos proposicionais. Concordamos, então, que se A e B são verdadeiras, *A* A *B* (leia-se *"A* e B") deve ser considerada verdadeira.

PRATICA 1
a. Se *A* é verdadeira e *B* é falsa, que valor você atribuiria a *A* ^ *B?*
b. Se *A* é falsa e *B* é verdadeira, que valor você atribuiria a *A* ^ *B?*
c. Se *A* e *B* são ambas falsas, que valor você atribuiria a *A* ^ *B?*

*(As respostas aos exercícios práticos estão no fim do livro.)* •

A expressão *A* A *B* é chamada a **conjunção** de *A* e *B;* e *A* e *B* são chamados os **fatores** da expressão. Podemos resumir os efeitos das conjunções através do uso da **tabela-verdade** apresentada na Tabela 1.1. Em cada linha da tabela-verdade os valores-verdade são atribuídos aos símbolos proposicionais e o valor-verdade resultante da composição da expressão é, então, mostrado.

| *A* | *B* | *A^B* |
|---|---|---|
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

| *A* | *B* | *A^B* |
|---|---|---|
| V | V | V |
| V | F | |
| F | V | |
| F | F | |

**Tabela 1.1**

Outro conectivo é a palavra *ou*, denotada pelo símbolo $\vee$. A expressão $A \vee B$ (leia-se "A ou B") é chamada **disjunção** de A e B e Ae B são chamados de **parcelas** da expressão. *Se A e B* forem ambos verdadeiros, $A \vee B$ deverá ser considerada verdadeira, nos dando a primeira linha da tabela-verdade para a disjunção (veja a Tabela **1.2).**

**PRATICA 2** Use o que entendeu da palavra *ou* para completar a tabela-verdade para a disjunção, isto é, a Tabela 1.2 •

As sentenças podem ainda ser combinadas na forma "se sentença 1, então sentença 2". Se *A* denota a sentença 1 e *B* denota a sentença 2, a sentença composta deve ser denotada por $A \rightarrow B$ (leia-se "A implica *B").* O conectivo lógico aqui é a **implicação** e indica que a verdade de *A* implica ou leva à verdade de *B.* Existem outras maneiras de expressar $A \rightarrow B$ na linguagem quotidiana, tal como "A é condição suficiente para B", "A somente se *B", "B* é conseqüência de A". Na expressão $A \rightarrow B$, *A* constitui a sentença **antecedente** e *B* a sentença **conseqüente.**

**EXEMPLO 2** A sentença "Fogo é uma condição necessária para fumaça" pode ser reformulada como "Se há fumaça, então há fogo". O antecedente é "há fumaça", e o conseqüente é "há fogo". •

**PRATICA 3** Indique o antecedente e o conseqüente em cada uma das seguintes sentenças. *(Dica:* reescreva as frases na forma se-então.) •

a. Se a chuva continuar, o rio vai transbordar.
b. Uma condição suficiente para a falha de uma rede é que a chave geral pare de funcionar.
c. Os abacates só estão maduros quando estão escuros e macios.
d. Uma boa dieta é uma condição necessária para um gato saudável. •

A tabela-verdade para a implicação é menos óbvia do que para a conjunção e disjunção. Para compreendermos sua definição, vamos supor que seu colega de quarto diga "Se eu me formar nesta primavera, vou tirar férias na Flórida." Se ele, de fato, se formar na primavera e tirar suas férias na Flórida, a sentença foi verdadeira. Se A e *B* forem ambas verdadeiras, consideraremos a implicação $A \rightarrow B$ verdadeira. Se o seu colega se formar e não tirar as férias na Flórida, seu comentário consistiu em uma sentença falsa. Quando A é verdadeira e *B* é falsa, consideramos $A \rightarrow B$ falsa. Suponhamos agora que seu colega não se formou. Independentemente de ele tirar ou não férias na Flórida, não poderemos acusá-lo de ter formulado uma sentença falsa e lhe daremos o benefício da dúvida. Por convenção, aceitamos $A \rightarrow B$ como verdadeira se A for falsa, independentemente do valor-verdade de *B.*

PRATICA 4 Resuma a explanação acima, escrevendo a tabela-verdade de $A \rightarrow B$. •

| A | *B* | $A \rightarrow B$ | $B \rightarrow A$ | $(A \rightarrow B) \wedge (B \rightarrow A)$ |
|---|---|---|---|---|
| V | V | V | V | V |
| V | F | F | V | F |
| F | V | V | F | F |
| F | F | V | V | V |

**Tabela 1.3**

Suponhamos que $A \rightarrow B$ seja verdadeira. Então, de acordo com a tabela-verdade da implicação, *B* pode ser verdadeira ou A pode ser falsa. Portanto, a despeito do fato de A ser verdadeira implicar que *B* também o seja, *B* ser verdadeira não implica que A o seja. A frase "S é uma condição suficiente para A" para denotar $A \rightarrow B$ apenas significa que, se A for verdadeira, *B* também o será.

O conectivo de **equivalência** é denotado pelo símbolo $\leftrightarrow$. A expressão $A \leftrightarrow B$ é a abreviação de $(A \rightarrow B)$ $A$ $(B \rightarrow A)$. Podemos escrever a tabela-verdade para a equivalência construindo, parte por parte, a tabela para $(A \rightarrow B)$ $A$ $(B \rightarrow A)$, como na Tabela 1.3. Perceba, nesta tabela, que $A \leftrightarrow B$ é verdadeira exatamente quando A e *B* têm o mesmo valor-verdade. A expressão $A \leftrightarrow B$ *é* normalmente lida como "A se, e somente se. B".

Os conectivos que vimos até agora são chamados de **conectivos binários** pois eles unem duas expressões a fim de produzir uma terceira. Vamos agora considerar um **conectivo unário,** isto é, um conectivo que atua em uma única expressão para produzir uma outra. A **negação** é um conectivo unário. A negação de A, A ' é lida como "não A", "A é falsa" ou "A não é verdade". Isto não quer dizer que A' sempre tenha um valor-verdade falso, mas que o valor-verdade de A' é o contrário do de A. (Alguns textos denotam A ' como A.)

PRATICA 5 Escreva a tabela-verdade para *A'*. (Serão necessárias apenas duas linhas.) •

**EXEMPLO 3** Se A é a sentença "Vai chover amanhã", a sentença A' é "Não é verdade que vai chover amanhã", que pode ser reescrita como "Não vai chover amanhã". •

Achar a negativa de uma sentença composta pode exigir algum esforço. Se *P* for a sentença "Peter é alto e magro", então a sentença *P'* será "É falso que Peter seja alto e magro", que pode ser reformulada como "Peter não é alto ou não é magro". Perceba que esta sentença *não* é a mesma que "Peter é baixo e gordo". Se *P* for a sentença "O rio é raso ou poluído", então *P'* é a sentença "É falso que o rio seja raso ou poluído", que pode ser reescrita como "O rio nem é raso nem é poluído" ou ainda "O rio é profundo e despoluído". No entanto, *P' não* é equivalente a "O rio não é raso ou não é poluído".

PRATICA 6 Qual das frases a seguir representa *A'* se *A* é a sentença "Julie adora manteiga mas detesta nata"?

a. Julie detesta manteiga e nata.
b. Julie não gosta de manteiga ou nata.
c. Julie não gosta de manteiga mas adora nata.
d. Julie detesta manteiga ou adora nata. •

Podemos encadear sentenças, seus conectivos e os parênteses (ou colchetes) para obtermos novas expressões, tal como em

***(A → B) ^ (B →A)***

Naturalmente, como nas linguagens de programação, regem algumas *regras de sintaxe* (regras sob as quais as cadeias são válidas); por exemplo,

***A)) ^ ^ →BC***

não deve ser considerada uma cadeia válida. Expressões que formam cadeias válidas são chamadas de **fórmulas bem-formuladas** ou **wffs** (de *well-formed formulas).* A fim de reduzir o número de parênteses necessários em uma wff, estipulamos uma ordem na qual os conectivos são aplicados. Esta "ordem de precedência" é:

1. Conectivos dentro de parênteses, dos mais internos para os mais externos
2. '
3. ^, ∨
4. →
5. ↔

Isto significa que a expressão A ∨ *B'* significa *A* ∨ (B')e não(A∨ *B)'*. Analogamente, *A* ∨ *B* → *C* significa *(A* ∨ *B)* → C e não A∨ *(B* → *C).*

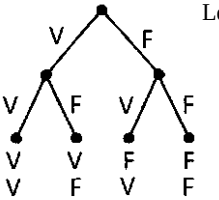
Wffs compostas de letras representativas de sentenças e conectivos têm valores-verdade que dependem dos valores-verdade atribuídos aos símbolos proposicionais. Escrevemos as tabelas-verdade para qualquer wff montando as partes que as compõem, como fizemos para (A → *B)* ∧ *(B* → A).

**EXEMPLO 4** A tabela-verdade para a wff A ∨ *B'* →(A ∨ *B)'* é dada na Tabela **1**.4. •

| *A* | *B* | *B'* | *A* ∨ *B'* | *A* ∨ *B* | *(A* ∨ *B)'* | *A* ∨ *B'* → *(A* ∨ *B)'* |
|---|---|---|---|---|---|---|
| V | V | F | V | V | F | F |
| V | F | V | V | V | F | F |
| F | V | F | F | V | F | V |
| F | F | V | V | F | V | V |

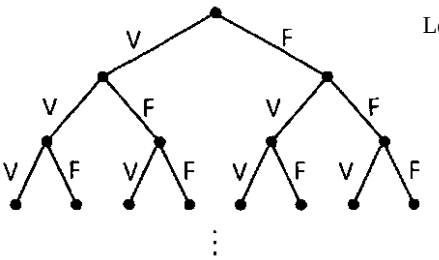
**Tabela 1.4**

Se estivermos montando uma tabela-verdade para uma wff que contenha *n* símbolos proposicionais diferentes, quantas linhas terá a tabela? Das tabelas feitas até agora, sabemos que uma wff com apenas um símbolo proposicional tem duas linhas em sua tabela-verdade e uma wff com dois símbolos proposicionais tem quatro linhas. O número de linhas é igual ao número de combinações verdadeiro-falso possíveis entre as letras




| Letras de afirmações | Possibilidades |
|---|---|
| 1 | $2 = 2^1$ nós |
| 2 | $4 = 2^2$ nós |

(a)




| Letras de afirmações | Possibilidades |
|---|---|
| 1 | $2 = 2^1$ nós |
| 2 | $4 = 2^2$ nós |
| 3 | $8 = 2^3$ nós |
| *n* | ($2^n$ nós) |

(b)

**Figura 1.1**

das combinações. O primeiro símbolo proposicional tem dois valores possíveis, V e F. Para cada um desses valores, o segundo símbolo proposicional também tem dois valores possíveis. A Fig. 1.1a ilustra isto na forma de uma "árvore" de dois níveis com quatro folhas, que mostram as quatro combinações possíveis de V e F para dois símbolos proposicionais. Para *n* símbolos proposicionais estendemos a árvore a *n* níveis, como na Fig. 1.1 .b. O número total de folhas será igual, portanto, a $2^n$ O número total de linhas em uma tabela-verdade para *n* símbolos proposicionais será também 2".

Essa estrutura de árvore nos mostra como enumerar todas as combinações V-F dentre os *n* símbolos proposicionais quando precisamos montar uma tabela-verdade. Se lermos cada nível da árvore de baixo para cima, veremos que os valores V-F para o símbolo proposicional *n* (a que compõe a última coluna da tabela-verdade) desdobra cada valor da linha *n* — 1 em dois valores e os da linha *n* — *2* em quatro valores e assim por diante. Portanto, uma tabela-verdade para três símbolos proposicionais começaria como mostrado na Tabela 1.5. Os valores para a sentença *C* variam, para os valores da sentença *B* em grupos de dois e para os valores da sentença *A* em grupos de quatro, resultando em uma versão horizontal de uma árvore. (Lendo as linhas de cima para baixo e usando 1 para os V e 0 para os F veremos que estamos apenas contando a partir de zero em binário.)

| ***A*** | ***B*** | ***C*** |
|---|---|---|
| V | V | V |
| V | V | F |
| V | F | V |
| V | F | F |
| F | V | V |
| F | V | F |
| F | F | V |
| F | F | F |

**Tabela 1.5**

PRATICA 7 Construa as tabelas-verdade para as seguintes wffs.

**a.** $(A \to B) \leftrightarrow (B \to A)$ (lembre-se que $C \leftrightarrow$ D só é verdadeira quando *C e D* têm o mesmo valor- verdade)

**b.** $(A \vee A') \to (B \wedge B')$

**c.** $[(A \wedge B') \to C']'$

**d.** $(A \to B) \leftrightarrow (B' \to A')$

•

## Tautologias

Uma wff como a (d) da Prática 7, cujos valores-verdade são sempre verdadeiros, é chamada uma **tautologia.** Uma wff como a do item (b), cujos valores-verdade são sempre falsos, é chamada uma **contradição.** Quando uma wff composta na forma $P \leftrightarrow Q$ *é* uma tautologia, como na Prática 7(d), os valores-verdade de *P* e *Q* conferem para todas as colunas da tabela-verdade. Neste caso, *P* e *Q* são chamadas de **wffs equivalentes** e denotadas por $P \Leftrightarrow Q$. Desta forma, $P \Leftrightarrow Q$ indica um fato, a saber: a wff particular $P \leftrightarrow Q$ *é* uma tautologia.

Vamos listar algumas equivalências básicas, provar uma ou duas delas através da construção das tabelas-verdade e deixar as demais como exercícios. Representaremos qualquer contradiçãcrpor 0 e qualquer tautologia por 1.

**Algumas Equivalências Tautológicas**

| | | |
|---|---|---|
| **1a.** $A \vee B \Leftrightarrow B \vee A$ | **1b.** $A \wedge B \Leftrightarrow B \wedge A$ | (propriedades comutativas) |
| **2a.** $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$ | **2b.** $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$ | (propriedades associativas) |
| **3a.** $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$ | **3b.** $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ | (propriedades distributivas) |
| **4a.** $A \vee 0 \Leftrightarrow A$ | **4b.** $A \wedge 1 \Leftrightarrow A$ | (propriedades de identidade) |
| **5a.** $A \vee A' \Leftrightarrow 1$ | **5b.** $A \wedge A' \Leftrightarrow 0$ | (propriedades complementativas) |

(Perceba que 2a nos leva a escrever $A \vee B \vee C$ sem a necessidade de parênteses; analogamente, 2b nos leva a escrever $A \wedge B \wedge C$.)

**EXEMPLO 5** A tabela-verdade da Tabela 1.6 verifica a equivalência 1 a, a propriedade comutativa da disjunção, e a da Tabela 1.7 verifica a 4b, a propriedade da identidade para a conjunção. Perceba que foram necessárias apenas duas linhas para a Tabela 1.7 porque 1 (uma tautologia) não pode assumir valores falsos. •

| $A$ | $B$ | $A \vee B$ | $B \vee A$ | $A \vee B \leftrightarrow$5 v A |
|---|---|---|---|---|
| V | V | V | V | V |
| V | F | V | V | V |
| F | V | V | V | V |
| F | F | F | F | V |

**Tabela 1.6**

| $A$ | 1 | $A \wedge 1$ | $A \wedge 1 \leftrightarrow$ A |
|---|---|---|---|
| V | V | V | V |
| F | V | F | V |

**Tabela 1.7**

PRATICA 8 Verifique a equivalência 5a. •

As equivalências em nossa lista são grupadas em cinco pares. Em cada par, uma equivalência pode ser obtida da outra substituindo-se os $\wedge$ por $\vee$, $\vee$ por $\wedge$, 0 por 1 e 1 por 0. Cada equivalência em um par é chamada a **dual** da outra. Esta lista de equivalência aparece de uma forma mais geral no Cap. 7.

Duas outras equivalências muito úteis são as **leis de De Morgan,** assim chamadas devido ao matemático inglês do século XIX Augustus De Morgan, que primeiro as formulou. Essas leis são:

$$(A \vee B)' \Leftrightarrow A' \wedge B' \quad \text{e} \quad (A \wedge B)' \Leftrightarrow A' \vee B'$$

Cada qual é a dual da outra. As leis de De Morgan nos ajudam a representar a negação de sentenças compostas, como na Prática 6.

Suponha que *P* e *Q* são equivalentes e que *P* aparece como uma componente de uma wff *R* grande. O que acontece se substituirmos *Q* por *P?* Por mais que os valores-verdade mudem, não haverá qualquer alteração. Denotemos por $R_Q$ a wff obtida pela substituição de *P* por *Q* em *R.* Se construirmos as tabelas-verdade para *R* e para $R_Q$ , a cada linha os valores de *Q* e *P* coincidem; logo, a cada linha, os valores de *R* e $R_Q$ também coincidem. Portanto. $R_Q$ *é* equivalente à wff original *R.*

**EXEMPLO 6** Seja $R\ (A \rightarrow B) \rightarrow B$ e seja $P\ A \rightarrow B$. Da Prática 7(d), sabemos que Pé equivalente a $Q = B' \rightarrow A'$. Substituindo *P* por *Q.* obtemos $R_{,1} = (B' \rightarrow A') \rightarrow B$. As tabelas-verdade para *R* e $R_Q$ podem ser vistas nas Tabelas 1.8 e 1.9. Os valores-verdade de $A \rightarrow B$ e $B' \rightarrow A'$ coincidem para todas as linhas, portanto os valores-verdade de *R* e $R_Q$ coincidem para todas as linhas. Logo *R* e $R_Q$ são equivalentes. •

| $A$ | $B$ | $A \rightarrow B$ | $(A \rightarrow B) \rightarrow B$ |
|---|---|---|---|
| V | V | V | V |
| V | F | F | V |
| F | V | V | V |
| F | F | V | F |

**Tabela 1.8**

| $A$ | $B$ | $A'$ | $B'$ | $B' \rightarrow A'$ | $(B' \rightarrow A') \rightarrow B$ |
|---|---|---|---|---|---|
| V | V | F | F | V | V |
| V | F | F | V | F | V |
| F | V | V | F | V | V |
| F | F | V | V | V | F |

**Tabela 1.9**

**Equivalências Tautológicas**

| | | |
|---|---|---|
| Propriedades Comutativas: | $A \vee B \Leftrightarrow B \vee A$ | $A \wedge B \Leftrightarrow B \wedge A$ |
| Propriedades Associativas: | $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$ | $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$ |
| Propriedades Distributivas: | $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$ | $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ |
| Propriedades de Identidade: | $A \vee 0 \Leftrightarrow A$ | $A \wedge 1 \Leftrightarrow A$ |
| Propriedades Complementativas: | $A \vee A' \Leftrightarrow 1$ | $A \wedge A' \Leftrightarrow 0$ |
| Leis de De Morgan: | $(A \vee B)' \Leftrightarrow A' \wedge B'$ | $(A \wedge B)' \Leftrightarrow A' \vee B'$ |
| Propriedades Idempotentes: | $A \vee A \Leftrightarrow A$ | $A \wedge A \Leftrightarrow A$ |
| Dupla Negativa: | $(A')' \Leftrightarrow A$ | |
| Reescrevendo a Implicação: | $(A \rightarrow B) \Leftrightarrow A' \vee B$ | |
| Contraposição: | $(A \rightarrow B) \Leftrightarrow (B' \rightarrow A')$ | |
| Prova Condicional | $A \rightarrow (B \rightarrow C) \Leftrightarrow (A \wedge B) \rightarrow C$ | |

Apesar de existirem infinitas tautologias, o quadro anterior resume as equivalências mais úteis (que expressam certas tautologias), incluindo aquelas que já havíamos listado anteriormente. Essas equivalências podem ser usadas, como vimos, para reescrever wffs. Como antes, 0 é qualquer contradição e 1 é qualquer tautologia.

## Conectivos Lógicos e Programação

Os conectivos lógicos E, OU e NÃO (ou, mais comumente seus equivalentes em inglês AND, OR e NOT) são oferecidos pela maioria das linguagens de programação. Esses conectivos, de acordo com as tabelas-verdade que definimos, agem sobre combinações de expressões verdadeiras e falsas a fim de produzir um valor-verdade final. Desses valores provém a capacidade de tomada de decisão fundamental ao controle do fluxo de programas de computadores. Desta forma, em um desvio condicional de um programa, se o valor-verdade de uma determinada expressão for verdadeiro, o programa irá executar um trecho de seu código; se o valor for falso, o programa executa, em seguida, outro trecho de seu código. Se a expressão condicional for substituída por uma expressão mais simples equivalente, o valor-verdade da expressão e, portanto, o controle do fluxo do programa não serão afetados, mas o novo código torna-se mais simples de ser entendido e poderá ser executado mais rapidamente.

EXEMPLO 7 Vejamos o seguinte comando na linguagem de programação Pascal:

```
if (fluxoext > fluxoint)
      and not ((fluxoext > fluxoint) and (pressão < 1000)) then
            UmProcedimento(lista de parâmetros)
else
            OutroProcedimento(lista de parâmetros);
```

A expressão condicional aqui tem a seguinte forma

$$A \wedge (A \wedge B)'$$

onde *A é fluxoext > fluxoint* e *B é pressão* < 1000. Esta expressão pode ser simplificada substituindo-se algumas subexpressões por suas expressões equivalentes.

$$
\begin{aligned}
A \wedge (A \wedge B)' &\Leftrightarrow A \wedge (A' \vee B') && \text{(lei de De Morgan)}\\
&\Leftrightarrow (A \wedge A') \vee (A \wedge B') && \text{(propriedade distributiva)}\\
&\Leftrightarrow 0 \vee (A \wedge B') && \text{(propriedade complementativa)}\\
&\Leftrightarrow (A \wedge B') \vee 0 && \text{(propriedade comutativa)}\\
&\Leftrightarrow A \wedge B' && \text{(propriedade de identidade)}
\end{aligned}
$$

O comando pode então ser reescrito como

**if** *(fluxoext > fluxoint)* **and not** *(pressão* < **1**000) **then**
    *UmProcedimento(lista* de parâmetros)
**else**
    *OutroProcedimento(lista* de parâmetros);

## Um Algoritmo

Se tivermos uma wff da forma $P \rightarrow Q$, onde *P e Q* são wffs compostas, podemos usar um procedimento mais rápido para construir uma tabela-verdade para determinar quando $P \rightarrow Q$ *é* uma tautologia. Assumimos que $P \rightarrow Q$ *não* é uma tautologia — isto é, que possa assumir valores falsos — e vejamos se isto nos leva a alguma situação impossível. $P \rightarrow Q$ é falsa apenas quando *P* é verdadeira e *Q* é falsa. Se atribuirmos verdadeira a *P* e falsa a *Q*, determinamos valores-verdade para os wffs reunindo *P* e *Q*. Continuamos atribuindo os valores-verdade até que todas as ocorrências de símbolos proposicionais tenham um valor-verdade. Se tivermos atribuído a alguma letra ambos os valores verdadeiro e falso, temos uma situação impossível, logo $P \rightarrow Q$ *é* uma tautologia. Caso contrário, temos uma maneira de tornar $P \rightarrow Q$ falso, então não é uma tautologia.

O que descrevemos foi um conjunto de instruções, um procedimento, para se encarregar da determinação de se $P \rightarrow Q$ é ou não uma tautologia. Este procedimento pode ser executado através da execução mecânica dessas instruções; em um período finito de tempo teremos a resposta. Em termos da ciência da computação, o procedimento é um *algoritmo.*

**Definição: Algoritmo**
Um **algoritmo** é um conjunto de instruções que pode ser executado mecanicamente em uma quantidade finita de tempo e que resolve algum problema.

Os algoritmos constituem o coração da Ciência da Computação, e temos mais a dizer sobre eles através deste livro. Você provavelmente já sabe que a principal tarefa ao se escrever um programa para computador consiste em arquitetar um algoritmo (procedimento) para produzir a solução.

Os algoritmos são normalmente descritos em uma forma intermediária entre uma descrição puramente verbal na forma de parágrafos (tal como a que demos para decidir se $P \rightarrow Q$ é ou não uma tautologia) e um programa de computador (que, se executado, iria realizar os passos do algoritmo) escrito em uma linguagem de programação. Esta forma intermediária para descrição de algoritmos é chamada de **pseudocódigo.** Um algoritmo escrito em pseudocódigo não deve ser difícil de entender, mesmo que você não saiba nada a respeito de programação de computadores. (Os algoritmos apresentados neste livro vão desde de pseudocódigos descritivos até códigos executáveis em Pascal.) Apresentamos a seguir um pseudocódigo na forma do de Pascal para determinar se $P \rightarrow Q$ é ou não uma tautologia.

**ALGORITMO** ***TestaTautologia***

**procedure** *TestaTautologia (P,Q:* wffs compostas);
{ determina se a wff $P \rightarrow Q$ é ou não uma tautologia }

**begin**

j atribui valores-verdade a *P* e *Q }*
*P := true;*
*Q: = false;*

**repeat**
    **for** cada wff que já tenha um valor-verdade atribuído, atribua os valores-verdade determinados para suas componentes
**until** todas as ocorrências de símbolos proposicionais têm valores-verdade;

```
if alguma letra tem dois valores-verdade then
      write ('Verdadeiro') { P - > Q é uma tautologia }
else
      write ('Falso');              { P - > Q não é uma tautologia }

end;
```

Este algoritmo inclui um laço de repetição, dentro do qual as atribuições dos valores-verdade são realizadas para componentes cada vez menores dos *P* e *Q* originais, até que a condição de *saída do laço* — todas as ocorrências de símbolos proposicionais tenham valores-verdade atribuídos — seja satisfeita. Em seguida o algoritmo procede com o comando condicional, onde é decidido se $P \rightarrow Q$ constitui ou não uma tautologia.

**EXEMPLO 8** Considere uma wff da forma $(A \rightarrow B) \rightarrow (B' \rightarrow A')$. Aqui P é $A \rightarrow B$ e $Q$ é $B' \rightarrow A'$. Seguindo o algoritmo *TestaTautologia*, atribuímos valores

$A \rightarrow B$ verdadeira e $B' \rightarrow A'$ falsa

o que significa que $(A \rightarrow B) \rightarrow (B' \rightarrow A')$ não é uma tautologia. Caminhando no laço, a atribuição de falso à wff composta $B' \rightarrow A'$ requer as atribuições futuras:

*B*' verdadeira e *A*' falsa

ou

*B* falsa e *A* verdadeira

A combinação de A verdadeira e $A \rightarrow B$ verdadeira requer a atribuição

*B* verdadeira

Neste ponto todas as ocorrências de símbolos proposicionais têm valores-verdade como mostrado a seguir:

$$\underbrace{A\text{-V}\ (A \rightarrow B)\ B\text{-V}}_{V} \rightarrow \quad \underbrace{B\text{-F}\ (B' \rightarrow A')\ A\text{-F}}_{F}\ \text{V}$$

Isto termina o laço. No passo final do algoritmo, *B* tem atribuições tanto de V como de F. Esta situação impossível indica que nossa suposição estava errada e, portanto, $(A \rightarrow B) \rightarrow (B' \rightarrow A')$ é uma tautologia. Podemos verificar isso construindo a tabela-verdade. •

O algoritmo *TestaTautologia* determina se wffs de determinadas formas são ou não tautologias. No entanto, o processo de construção da tabela-verdade e o exame dos valores-verdade em sua última coluna constituem um algoritmo que determina se uma wff arbitrária é ou não uma tautologia. Este segundo algoritmo é, portanto, mais poderoso, uma vez que resolve problemas mais gerais.

## Revisão da Seção 1.1

### Técnicas

- Construção de tabelas-verdade para wffs compostas
- Reconhecimento de tautologias e contradições

### Principais Conceitos

As wffs são representações simbólicas de sentenças.
Os valores-verdade de wffs compostas dependem dos valores-verdade de seus componentes e do tipo dos conectivos usados.

## Exercícios 1.1

As respostas dos itens com estrelas ao lado são dadas ao final do livro.

★1. Quais das frases a seguir são sentenças?
   a. A lua é feita de queijo verde.
   b. Ele é um homem alto.
   c. Dois é um número primo.
   d. O jogo terminará logo?
   e. As taxas do ano que vem serão maiores.
   f. As taxas do ano que vem serão menores.
   g. $x- - 4 = 0$

2. Dados os valores-verdade $A$ verdadeiro, $B$ falso e $C$ verdadeiro, qual o valor-verdade de cada uma das seguintes wffs?
   a $A \wedge (B \vee C)$ b $(A \wedge B) \vee C$
   c. $(A \wedge B)' \vee C$ d. $A' \vee (B' \wedge C)'$

3. Qual os valores-verdade das seguintes sentenças?
   a. 8 é par ou 6 é ímpar.
   b. 8 é par e 6 é ímpar.
   c. 8 é ímpar ou 6 é ímpar.
   d. 8 é ímpar e 6 é ímpar.
   e. Se 8 é ímpar, então 6 é ímpar.
   f. Se 8 é par, então 6 é ímpar.
   g. Se 8 é ímpar, então 6 é par.
   h. Se 8 é ímpar e 6 é par, então $8 < 6$.

★4. Indique o antecedente e o conseqüente de cada uma das seguintes sentenças:
   a. O crescimento sadio das plantas é conseqüência de quantidade suficiente de água.
   b. O crescimento da oferta de computadores é uma condição necessária para o desenvolvimento científico.
   c. Haverá novos erros apenas se o programa for alterado.
   d. A economia de combustível implica um bom isolamento, ou todas as janelas são janelas para tempestades.

5. Diversas negativas são dadas para cada uma das seguintes afirmações. Quais são as certas?
   a. A resposta é 2 ou 3.
      1. Nem 2 nem 3 são a resposta.
      2. A resposta não é 2 ou não é 3.
      3. A resposta não é 2 e não é 3.
   b. Pepinos são verdes e têm sementes.
      1. Pepinos não são verdes e não têm sementes.
      2. Pepinos não são verdes ou não têm sementes.
      3. Pepinos são verdes e não têm sementes.
   c. $2 < 7$ e 3 é ímpar.
      1. $2 > 7$ e 3 é par.
      2. $2 \geq 7$ e 3 é par.
      3. $2 \geq 7$ ou 3 é ímpar.
      4. $2 \geq 7$ ou 3 é par.

6. Sejam $A$, $B$ e $C$ as seguintes sentenças:

   *A:* Rosas são vermelhas.
   *B:* Violetas são azuis.
   *C:* Açúcar é doce.

   Traduza as seguintes sentenças compostas para notação simbólica.
   a. Rosas são vermelhas e violetas são azuis.
   b. Rosas são vermelhas e, ou bem violetas são azuis ou bem açúcar é doce.
   c. Sempre que violetas são azuis, as rosas são vermelhas e o açúcar é doce.
   d. Rosas são vermelhas apenas se as violetas não forem azuis e se o açúcar for azedo.
   e. Rosas são vermelhas e, se o açúcar for azedo, então as violetas não são azuis ou o açúcar é doce.

7. Com $A$, $B$ e $C$ como os definidos no Exercício 6. traduza as seguintes wffs para o português:
   **a.** $B \vee C'$
   **b.** $B' \vee (A \rightarrow C)$
   **c.** $(C \wedge A') \leftrightarrow B$
   **d.** $C \wedge (A' \leftrightarrow B)$
   **e.** $(B \wedge C')' \rightarrow A$
   **f.** $A \vee (B \wedge C')$
   **g.** $(A \vee B) \wedge C'$

★**8.** Com o uso de letras para denotar as sentenças componentes, traduza as seguintes sentenças compostas para notação simbólica:
   a. Se os preços subirem, as construções ficarão mais caras, mas se as construções não forem caras, elas serão muitas.
   b. Tanto ir para cama como nadar é condição suficiente para trocar de roupa; no entanto, trocar de roupa não significa que se vai nadar.
   c. Ou vai chover ou vai nevar, mas não ambos.
   d. Se Janet vencer ou perder, ela estará cansada.
   e. Ou Janet irá vencer ou, se perder, ficará cansada.

9. Construa as tabelas-verdade para as seguinteswffs. Indique as tautologias e as contradições.
   ★**a.** $(A \rightarrow B) \leftrightarrow A' \vee B$ ★**b.** $(A \wedge B) \vee C \rightarrow A \wedge (B \vee C)$
   **c.** $A \wedge (A' \vee B')'$ **d.** $A \wedge B \rightarrow A'$
   **e.** $(A \rightarrow B) \rightarrow [(A \vee C) \rightarrow (B \vee C)]$ **f.** $A \rightarrow (B \rightarrow A)$
   **g.** $A \wedge B \leftrightarrow B' \vee A'$ **h.** $(A \vee B') \wedge (A \wedge B)'$
   **i.** $[(A \vee B) \wedge C'] \rightarrow A' \vee C$

★10. Um chip da memória de um microcomputador tem $2^4$ elementos de dois estados (LIGA-DESLIGA). Qual o número total de configurações LIGA-DESLIGA?

11. Considere o seguinte fragmento de um programa Pascal:

```
for contador : = 1 to 5 do
  begin
    read(a);
    if ((a < 5.0)and(2*a < 10.7)) or (sqrt(5.0*a) > 5.1) then writeln(a);
 end;
```

Os valores de entrada para *a* são 1.0, 5.1, 2.4, 7.2 e 5.3. Quais são os valores de saída?

12. Verifique as equivalências na lista da página 8 através da construção de suas tabelas-verdade. (Já verificamos as l.a, 4b e 5a).

13. Verifique, através da construção das tabelas-verdade que as wffs a seguir são tautologias:
   ★**a.** $A \vee A'$ **b.** $(A')' \leftrightarrow A$
   ★**c.** $A \wedge \mathbf{B} \rightarrow B$ **d.** $A \rightarrow A \vee B$
   **e.** $(A \vee B)' \leftrightarrow A' \wedge B'$ **f.** $(A \wedge B)' \leftrightarrow A' \vee B'$

14. Suponha que *A, B* e *C* representam condições que serão verdadeiras e falsas quando um programa é executado. Suponha ainda que desejamos que este programa realize alguma tarefa somente quando *A* ou *B* forem verdadeiras (mas não ambas) e C for falsa. Usando *A, B e* C e os conectivos E, OU e NOT, escreva uma sentença que será verdadeira apenas nestas condições.

15. Reescreva o programa Pascal a seguir com uma expressão condicional simplificada:

```
if not ((Valor1 < Valor2) or odd(Numero))
or {not(Valor1 < Valor2) and odd(Numero)) then
   comando 1
else
   comando2;
```

**16. a.** Verifique que $A \rightarrow B$ é equivalente a $A' \vee B$.
   b. Usando a parte (a) e outras equivalências, escreva a negação da sentença "Se Sam passar em seu curso de Física, então ele se formará."

17. Use o algoritmo *TestaTautologia* para provar que as expressões a seguir são tautologias:
   ★**a.** $[B' \wedge (A \rightarrow B)] \rightarrow A'$ **b.** $[(A \rightarrow B) \wedge A] \rightarrow B$
   **c.** $(A \vee B) \wedge A' \rightarrow B$ **d.** $(A \wedge B) \wedge B' \rightarrow A$

18. Em cada caso, construa as wffs compostas *P e Q* a fim de que a sentença dada seja uma tautologia.
   **a.** $P \wedge Q$
   **b.** $P \rightarrow P'$
   **c.** $P \wedge (Q \rightarrow P')$

19. A tabela-verdade de $A \vee B$ mostra que o valor de A $\vee$ *B* é verdadeiro se A for verdadeira, se *B* for verdadeira ou ainda se ambos o forem. O uso da palavra *ou* quando o resultado é verdadeiro, se ambos os componentes forem verdadeiros, é chamado de *ou inclusivo.* É este *ou inclusivo* que é entendido na frase "Teremos chuva ou garoa amanhã." Outro uso da palavra *ou* na língua portuguesa é o *ou exclusivo,* algumas vezes chamado de XOU (ou **XOR,** em inglês) no qual o resultado é falso quando ambos os componentes são verdadeiros. O ou exclusivo é entendido na frase "No cruzamento, devemos seguir para o norte ou para o sul". O ou exclusivo é simbolizado por A $\oplus$ *B.*
    a. Escreva a tabela-verdade para o ou exclusivo.
    b. Mostre que $A \oplus B \leftrightarrow (A \leftrightarrow B)'$ é uma tautologia.

20. Toda wff é equivalente a uma sentença que use apenas os conectivos da conjunção e negação. Para garantirmos isto devemos achar wffs equivalentes para A $\vee$ B e $A \to B$ que usem apenas $\wedge$ e '. Estas novas wffs poderão substituir, respectivamente, quaisquer ocorrências de A $\vee$ *B e A* $\to$ *B.* (O conectivo $\leftrightarrow$ já foi definido em termos dos outros conectivos, portanto já sabemos que pode ser substituído em uma wff.)
    a. Mostre que A $\vee$ *B é* equivalente a (A' $\wedge$ *B*')'.
    b. Mostre que A $\to$ *B* é equivalente a (A $\wedge$ *B')'.*

21. Mostre que qualquer wff composta é equivalente a uma wff que só contenha os conectivos de:
    a. $\vee$ e ' b. $\to$ e '.
    *(Dica:* Veja o exercício 20.)

22. Prove que há algumas wffs compostas que não são equivalentes a qualquer wff que use apenas os conectivos $\to$ e $\vee$.

★23. O conectivo binário I é definido pela tabela-verdade a seguir. Mostre que qualquer wff composta é equivalente a uma wff que use apenas o conectivo |. (*Dica:* Use o Exercício 20 e ache wffs equivalentes para A $\wedge$ B e A ' em termos de .)

| A | *B* | *A*\|*B* |
|---|---|---|
| *V* | *V* | *F* |
| *V* | *F* | *V* |
| *F* | *V* | *V* |
| *F* | *F* | *V* |

**Exercício 23**

| *A* | *B* | *A*↓*B* |
|---|---|---|
| *V* | *V* | *F* |
| *V* | *F* | *F* |
| *F* | *V* | *F* |
| *F* | *F* | *V* |

**Exercício 24**

**24.** O conectivo binário ↓ é definido pela tabela-verdade dada. Mostre que qualquer wff composta é equivalente a uma wff que use apenas o conectivo ↓. *(Dica:* Veja o Exercício 23).

25. Em um determinado país, todos os habitantes são ou um contador de verdade que sempre fala a verdade ou mentirosos que sempre mentem. Viajando neste país, você encontra dois habitantes, Percival e Llewellyn. Percival diz "Se eu for um contador de verdades, Llewellyn também é um contador de verdades". Percival é um mentiroso ou um contador de verdade? E Llewellyn?

# Seção 1.2 Quantificadores, Predicados e Validade

## Quantificadores e Predicados

O que podemos expressar através das wffs que vimos na Seção 1.1 é muito limitado. Por exemplo, consideraríamos a sentença "Para todo *x, x* > 0" como uma sentença verdadeira sobre inteiros positivos, no entanto ela não pode ser simbolizada adequadamente através de símbolos proposicionais, parênteses e conectivos lógicos. Ela contém dois novos elementos, um *quantificador* e um *predicado.* Os quantificadores são frases como "para todo", "para cada" ou "para algum", que indicam de alguma forma *quantos* objetos têm uma determinada propriedade. O **quantificador universal** é simbolizado por um A de cabeça-para-baixo, $\forall$, e é lido "para todo", "para todos", "para cada" ou para "para qualquer". Portanto, a sentença acima pode ser simbolizada como

$$(\forall x)(x > 0)$$

Um quantificador e sua variável são sempre colocados entre parênteses. O segundo par de parênteses indica que o quantificador age sobre a expressão interna, que no caso é "x > 0".

A frase *"x > 0"* descreve a propriedade da variável *x,* que é ser positiva. Uma propriedade também é chamada de **predicado;** usamos a notação *P(x)* para representar algum predicado não especificado ou propriedade que *x* possa ter. Portanto, a sentença original é um exemplo da forma mais geral

$$(\forall x)P(x)$$

O valor-verdade da expressão $(\forall x)(x > 0)$ depende do domínio dos objetos sob os quais estamos "interpretando" esta expressão. Isto é, a coleção de objetos dos quais *x* pode ser escolhido. Esta coleção de objetos é chamada de *domínio de interpretação.* Já havíamos visto que se o domínio de interpretação consistisse em inteiros positivos, a expressão teria o valor-verdade verdadeiro pois qualquer valor possível para *x* teria a propriedade necessária de ser maior que zero. Se o domínio de interpretação consistisse de todos os inteiros, a expressão teria o valor-verdade falso, pois nem todo *x* teria a propriedade necessária. Impomos a condição de que o domínio de interpretação tenha pelo menos um elemento a fim de que não nos preocupemos com o caso trivial.

Uma interpretação para a expressão $(\forall x)P(x)$ poderia consistir não apenas na coleção de objetos dos quais *x* pode tirar seus valores mas também da propriedade que *P(x)* representa em seu domínio. Portanto, uma interpretação para $(\forall x)P(x)$ poderia ser a seguinte: O domínio consiste em todos os livros em sua biblioteca local, e *P(x)* é a propriedade de que *x* deve ter capa vermelha. $(\forall x)P(x)$, nesta interpretação, diz que todo livro em nossa biblioteca local tem capa vermelha. O valor-verdade desta expressão nesta interpretação é, sem dúvida, falso.

PRÁTICA 9 Qual o valor-verdade da expressão $(\forall x)P(x)$ em cada uma das seguintes interpretações?

a. *P(x)* é a propriedade de que *x* seja amarelo e o domínio de interpretação é o conjunto de todos os canários-da-terra.
b. *P(x)* é a propriedade de que *x* seja amarelo e o domínio de interpretação é o conjunto de todos os pássaros.
c. *P(x)* é a propriedade de que *x* seja uma ave e o domínio de interpretação é o conjunto de todos os pássaros.
d. *P(x)* é a propriedade de que *x* seja positivo ou negativo e o domínio de interpretação consiste em todos os inteiros. •

O **quantificador existencial** é simbolizado por um E espelhado, ∃, e é lido como "existe um", "para pelo menos um" ou "para algum". Portanto, a expressão

$$(\exists x)(x > 0)$$

deve ser lida como "existe um *x* tal que *x* é maior que zero."

Novamente, o valor-verdade desta expressão depende da interpretação. Se o domínio de interpretação contiver um número positivo, a expressão terá valor-verdadeiro; caso contrário, ela terá valor falso. O valor-verdade de $(\exists x)P(x)$, se o domínio for todos os livros de nossa biblioteca local e *P(x)* for a propriedade de *x* ter a capa vermelha, será verdadeiro, desde que tenhamos pelo menos um livro na biblioteca com a capa vermelha.

PRÁTICA 10
a. Construa uma interpretação (i.e., dê o domínio e o significado de *P(x))* na qual $(\forall x)P(x)$ tenha o valor verdadeiro.
b. Construa uma interpretação na qual $(\forall x)P(x)$ tenha o valor falso.
c. É possível achar uma interpretação na qual tanto $(\forall x)P(x)$ seja verdadeiro e $(\exists x)P(x)$ seja falso?
d. É possível achar uma interpretação na qual tanto $(\forall x)P(x)$ quanto $(\exists x)P(x)$ sejam verdadeiros? •

Os predicados que vimos até agora, envolvendo propriedades de uma única variável, são **predicados unários.** Eles podem ser **binários,** quando envolvem propriedades de duas variáveis; **ternários,** quando envolvem propriedades de três variáveis; ou, de forma mais geral, **n-ários,** quando envolvem propriedades de *n* variáveis. Quantificadores universais podem ser incluídos a expressões com esses predicados de ordens altas.

EXEMPLO 9 A expressão $(\forall x)(\exists y)Q(x, y)$ é lida como "para todo *x* existe um *y* tal que *Q(x, y)".* Na interpretação onde o domínio consiste em inteiros e *Q(x, y)* é a propriedade de $x < y$, isto apenas indica que, para qualquer inteiro, existe um inteiro ainda maior. O valor-verdade desta expressão é verdadeiro. Na mesma interpretação, a expressão $(\exists y)(\forall x)Q(x, y)$ indica que existe um inteiro *y* que é maior que qualquer inteiro *x.* Seu valor-verdade é falso.

•

O Exemplo 9 mostra que a ordem na qual os quantificadores aparecem é importante.

Em expressões como $(\forall x)P(x)$ ou $(\exists x)P(x)$, *x* é uma *variável muda;* isto é, os valores-verdade das expressões permanecem os mesmos em uma dada interpretação, mesmo que escrevamos a expressão, digamos

como $(\forall y)P(y)$ ou $(\exists z)P(z)$, respectivamente. Analogamente, o valor-verdade de $(\forall x)(\exists y)Q(x, y)$ é o mesmo de $(\forall z)(\exists w)Q(z, w)$ para qualquer interpretação. No entanto, $(\forall x)(\exists x)Q(x, x)$ indica algo bem diferente. Na interpretação do Exemplo 9, por exemplo, $(\forall x)(\exists x)Q(x, x)$ indicaria que para todo inteiro *x*, existe um inteiro *x* tal que $x < x$. Esta sentença é falsa, a despeito de $(\forall x)(\exists y)Q(x, y)$ ser verdadeiro para a interpretação. Não podemos misturar variáveis diferentes sem mudar a natureza da expressão obtida.

Também são permitidas constantes nas expressões. Um símbolo de constante (a, *b*, c, etc.) é interpretado como um objeto específico no domínio. Esta especificação é parte da interpretação. Por exemplo, a expressão $(\forall x)Q(x,a)$ é falsa na interpretação na qual o domínio consista em inteiros, $Q(x, y)$ é a propriedade $x <$ y, e *a* contém o valor 7; não é verdade que todo inteiro seja menor que 7.

Agora temos tudo o que é necessário em uma interpretação.

**Definição: Interpretação**

Uma **interpretação** de uma expressão envolvendo predicados consiste no seguinte:

a. um conjunto de objetos chamados o **domínio** da interpretação, que deve conter pelo menos um elemento;
b. a atribuição de uma propriedade dos objetos do domínio para cada predicado na expressão; e
c. a atribuição de um objeto particular no domínio a cada símbolo constante na expressão.

As expressões podem ser obtidas da combinação de predicados, quantificadores, símbolos de agrupamento (parênteses ou colchetes) e dos conectivos lógicos da Seção 1.1. Como antes, uma expressão precisa obedecer regras sintáticas a fim de ser considerada uma fórmula bem-formulada ou wff. A expressão $P(x)(\forall x) \wedge)\exists y$ não é uma fórmula bem-formulada. Exemplos de fórmulas bem-formuladas são:

$$P(x) \vee Q(y) \tag{1}$$
$$(\forall x)[P(x) \rightarrow Q(x)] \tag{2}$$
$$(\forall x)((\exists y)[P(x, y) \wedge Q(x, y)] \rightarrow R(x)) \tag{3}$$

e

$$(\exists x)S(x) \vee (\forall y)T(y) \tag{4}$$

Os símbolos de agrupamento ajudam a identificar o escopo de um quantificador, a seção da wff a qual os quantificadores se aplicam. (Isto é análogo ao escopo de um identificador de um programa como a seção do programa na qual o identificador tem significado.) Não há quantificadores na wff (1). Na (2), o escopo do quantificador $(\forall x)$ é $P(x) \rightarrow Q(x)$. Na (3), o escopo do $(\exists y)$ é $P(x, y) \wedge Q(x, y)$, enquanto que o escopo de $(\forall x)$ é toda a expressão nos parênteses que o segue. Na (4), o escopo de $(\exists x)$ é $S(x)$ e o escopo de $(\forall y)$ é $T(y)$.

Sabemos que o valor-verdade de uma wff é determinado em relação a uma dada interpretação. Em alguns casos, como veremos em breve, uma wff pode não ter valor-verdade para uma particular interpretação. De forma que, para considerarmos o valor-verdade de uma wff, precisamos considerar o escopo de seus quantificadores.

**EXEMPLO 10** Considere a wff

$$(\forall x)(\exists y)[P(x, y) \wedge Q(x, y)]$$

Aqui o escopo de $(\exists y)$ é todo o $P(x, y) \wedge Q\{x,$ v). O escopo de $(\forall x)$ é $(\exists y)[P(x, y) \wedge Q(x, y)]$; os parênteses e os colchetes podem ser eliminados quando o escopo estiver claro. Na interpretação onde o domínio consista em inteiros positivos, $P(x, y)$ seja a propriedade "$x \leq y$" e $Q(x, y)$ seja a propriedade "x divide y", a wff é verdadeira. Para qualquer *x* que seja um inteiro positivo, existe um inteiro positivo y, por exemplo, $y = 2x$, tal que $x \leq y$ e *x* divide v.

Suponhamos agora que a wff é

$$(\forall x)[(\exists y)P(x, y) \wedge Q(x, y)]$$

Agora o escopo de $(\exists y)$ é apenas $P(x,$ v). Se usarmos a mesma interpretação de antes, não haverá valor-verdade determinado para esta wff. Dado qualquer *x* podemos escolher um y (tal como $y = 2x$) de forma que $(\exists y)P(x,$ v) seja verdadeira, mas em $Q(x,$ y), y está livre para tomar qualquer valor no domínio de interpretação, exceto os que escolhemos para *y* satisfazer $(\exists y)P(x, y)$. De fato, lembremos que y em $(\exists y)P(x, y)$ é uma variável muda. Para valores de y que sejam múltiplos de *x*, $Q(x,$ y) é verdadeiro e, portanto, toda a expressão também o é; para outros valores de y, $Q(x,$ y) é falso e, portanto, toda a expressão também o é. •

Na segunda wff do Exemplo 10, a variável *y* ocorre três vezes:

$$(\forall x)[(\exists y)P(x, y) \wedge Q(x, y)]$$
$$\uparrow \quad \uparrow \qquad \uparrow$$

A ocorrência de uma variável em uma wff é chamada de uma **ocorrência ligada** em duas condições:

1. Se tratar de uma variável identificando o que o quantificador quantifica, ou
2. estar fora do escopo de um quantificador envolvendo esta variável.

Portanto, a primeira ocorrência de v no exemplo acima é uma ocorrência ligada do primeiro tipo e a segunda ocorrência é uma ocorrência ligada do segundo tipo. A terceira ocorrência é uma **ocorrência livre** — que não é ligada. Ela não é uma variável que seja parte direta de um quantificador, nem é uma variável dentro do escopo de um quantificador que a envolva. Uma variável que tenha pelo menos uma ocorrência limite em uma wff é uma **variável ligada** na wff, e uma variável com pelo menos uma ocorrência livre é uma **variável livre** na wff. (Perceba que pode ocorrer a situação um tanto o quanto estranha de uma variável ser ao mesmo tempo ligada e livre na mesma expressão.)

Como no Exemplo 10, uma wff com variáveis livres freqüentemente — mas nem sempre — não terão valores-verdade para uma dada interpretação. De fato, a wff será verdadeira para alguns valores das variáveis livres e falsa para outros. Portanto, as wffs com predicados diferem das wffs da Seção 1.1, que eram compostas apenas por símbolos proposicionais e conectivos lógicos, e que sempre tinham valores-verdade. No entanto, uma wff com predicados e sem variáveis livres tem valor-verdade em qualquer interpretação, conquanto este valor possa alterar-se de uma interpretação para outra.

PRÁTICA 11 Qual o valor-verdade da wff

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

na interpretação onde o domínio consiste em todos os inteiros, *A(x) é "x* > 0", *B(x,y*v) é "*x* > *y*" e C(y) é "y ≤ 0"? Construa outra interpretação com o mesmo domínio, no qual a sentença tenha o valor-verdade oposto. •

Muitas sentenças em português podem ser expressas como wffs contendo predicados e quantificadores. Por exemplo, a sentença "todo papagaio é feio" está, na verdade, dizendo que qualquer coisa que seja um papagaio é feia. Fazendo *P(x)* denotar *"x* é um papagaio" e *U(x)* denotar *"x* é feio" vemos que a sentença pode ser simbolizada como

$$(\forall x)[P(x) \rightarrow U(x)]$$

Outras variantes com o mesmo significado na língua portuguesa são "Qualquer papagaio é feio" e "Cada papagaio é feio".

Da mesma forma, "Existe um papagaio feio" é denotado como

$$(\exists x)[P(x) \wedge U(x)]$$

As variantes aqui são "Alguns papagaios são feios" e "Existem papagaios feios".

Ao representar essas sentenças da língua portuguesa como wffs usamos $(\forall x)$ para a implicação e $(\exists x)$ para a conjunção. As duas outras combinações possíveis quase nunca expressam o que se deseja dizer. A wff $(\forall x)[P(x) \wedge U(x)]$ indica que todos os elementos no domínio — entendido aqui como todo o mundo — são um papagaio feio; a wff $(\exists x)[P(x) \rightarrow U(x)]$ é verdadeira na medida em que haja algum elemento no domínio, chamado *x,* que não seja um papagaio, pois neste caso, *P(x)* assume falso e a implicação é verdadeira.

PRATICA 1 2 Usando os símbolos predicados *S(x), I(x)* e *M(x),* escreva wffs que expressem o pedido. (O domínio é a coleção de todas as pessoas.)

a. Todos os estudantes são inteligentes.
b. Alguns estudantes inteligentes gostam de música.
c. Todos que gostam de música são estudantes estúpidos. •

Negar sentenças com quantificadores, assim como negar sentenças compostas, requer um certo cuidado. A negação da sentença "Tudo é bonito" é "Não é verdade que tudo é bonito" ou "Algo não é bonito". Simbolicamente,

$$[(\forall x)A(x)]' \leftrightarrow (\exists x)[A(x)]'$$

é válido. Perceba que "Tudo não é bonito", ou $(\forall x)[A(x)]'$, diz algo *mais forte* que a negação da sentença original.

A negação de "Algo é bonito" é "Nada é bonito" ou "Tudo não é bonito". Simbolicamente,

$$[(\exists x)A(x)]' \leftrightarrow (\forall x)[A(x)]'$$

é válido. Em inglês, a sentença "Algo não é bonito" pode ser erroneamente interpretada como "Nem tudo é bonito" ou "Há algo que não seja bonito". No entanto, esta interpretação errada, simbolizada por $(\exists x)[A(x)]'$ *não é tão forte* quanto a negação da sentença original.

## Validade

A fim de distinguirmos as wffs que contêm apenas símbolos proposicionais e conectivos lógicos (descritas na Seção 1.1) das que contêm predicados e variáveis, chamaremos as primeiras de **wffs proposicionais** e as últimas de **wffs predicativas.** Como vimos antes, uma wff proposicional sempre tem valor-verdade, enquanto que uma wff predicativa pode não ter valor-verdade.

O valor-verdade de uma wff proposicional depende dos valores-verdade atribuídos aos símbolos proposicionais. O valor-verdade (ou falta dele) de uma wff predicativa depende da interpretação. Escolher uma interpretação para uma wff predicativa é análogo a escolher valores-verdade para wffs proposicionais, exceto por haver um número infinito de interpretações possíveis para as wffs predicativas e apenas 2" linhas possíveis para wffs proposicionais com *n* símbolos proposicionais.

Uma tautologia é uma wff proposicional que é verdadeira em todas as linhas da tabela-verdade. O análogo à tautologia para as wffs predicativas é a *validade* — uma wff predicativa é **válida** se for verdadeira para qualquer interpretação possível. A validade de uma wff deve ser obtida apenas da forma da wff, uma vez que deve ser independente de qualquer interpretação em particular.

Sabemos que ao construirmos a tabela-verdade para uma wff proposicional e examinarmos todas as atribuições aos símbolos proposicionais, temos um algoritmo para determinar um "tautologismo". No entanto, como obviamente não podemos testar todas as interpretações possíveis, como podemos determinar a validade de uma wff predicativa? Como veremos, não existe algoritmo para determinar uma validade. (Isto não significa simplesmente que ainda não foi encontrado um algoritmo — o que estamos dizendo é que está *provado* que não existe um algoritmo deste tipo.) Precisamos usar o raciocínio para determinar quando a forma de uma wff a torna verdadeira para qualquer interpretação. Naturalmente, podemos provar que uma wff é inválida ao encontramos uma única interpretação na qual a wff tenha valor-verdade falso ou não tenha valor-verdade.

A tabela a seguir compara as wffs proposicionais e predicativas:

| **Wffs Proposicionais** | **Wffs Predicativas** |
|---|---|
| 1. Verdadeira ou falsa, de acordo com os valores atribuídos aos símbolos proposicionais | 1. Verdadeira, falsa ou sem valor-verdade, dependendo da interpretação |
| 2. Tautologia — verdadeira para todas as atribuições de valores-verdade | 2. Wff válida — verdadeira para todas as interpretações |
| 3. Algoritmo (tabela-verdade) para determinar se uma wff é ou não uma tautologia | 3. Não há algoritmo para determinar se uma wff é ou não válida |

Agora vamos pôr mãos à obra para determinar a validade.

EXEMPLO 11

a. A wff $(\forall x)P(x) \rightarrow (\exists x)P(x)$ é válida. Em qualquer interpretação, se qualquer elemento do domínio tiver uma certa propriedade, então existirá um elemento do domínio que tenha esta propriedade. (Usamos aqui o fato de que o domínio de qualquer interpretação tem que conter pelo menos um elemento.) Portanto, sempre que o antecedente for verdadeiro, o conseqüente também o será, e a implicação é, portanto, verdadeira.
b. A wff $(\forall x)P(x) \rightarrow P(a)$ é válida porque, em qualquer interpretação, *a* é um membro particular do domínio e, portanto, goza da propriedade que é compartilhada por todos os elementos do domínio.
c. A wff

$$(\forall x)[P(x) \wedge Q(x)] \leftrightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$$

é válida. Se tanto $P$ como $Q$ forem verdadeiras para todos os elementos do domínio, então $P$ será verdadeira para todos os elementos e $Q$ será verdadeira para todos os elementos e vice-versa.

**d** A wff $P(x) \rightarrow [Q(x) \rightarrow P(x)]$ é válida, apesar de conter uma variável livre. Para comprovarmos isto, consideremos qualquer interpretação, e seja $x$ qualquer membro do domínio. Então $x$ tem ou não tem a propriedade $P$. Se $x$ não a tiver, então $P(x)$ será falsa; como $P(x)$ é o antecedente da implicação, esta será verdadeira. Se, por outro lado, $x$ tiver a propriedade $P$, então $P(x)$ é verdadeira e, a despeito do valor-verdade de $Q(x)$, a implicação $Q(x) \rightarrow P(x)$ será verdadeira, e a implicação principal também será verdadeira. •

EXEMPLO 12 A wff $(\exists x)P(x) \rightarrow (\forall x)P(x)$ não é válida. Por exemplo, na interpretação onde o domínio consista em inteiros e $P(x)$ signifique que $x$ é par, é verdade que existe um inteiro par, mas é falso que todo inteiro seja par. O antecedente da implicação é verdadeiro e o conseqüente é falso, e portanto o valor da implicação é falso. •

Naturalmente não somos obrigados a usar um contexto matemático para construir uma interpretação na qual uma wff seja falsa, mas freqüentemente é mais simples fazê-lo pelo fato de as relações entre os objetos serem mais claras.

PRÁTICA 13 A wff

$$(\forall x)[P(x) \vee Q(x)] \rightarrow (\forall x)P(x) \vee (\forall x)Q(x)$$

é válida ou inválida? Justifique. •

## Revisão da Seção 1.2

### Técnicas

- Determinação do valor-verdade de uma wff predicativa em uma dada interpretação
- Tradução de sentenças na língua portuguesa para wffs e vice-versa
- Reconhecimento de uma wff válida e a justificação
- Reconhecimento de wffs não-válidas e a construção de uma interpretação na qual ela seja falsa ou não tenha valor-verdade

### Idéias Principais

O valor-verdade de wffs predicativas depende da interpretação considerada.
Wffs válidas são wffs predicativas que são verdadeiras para qualquer interpretação e, portanto, a validade é uma propriedade inerente à forma da wff propriamente dita.

## Exercícios 1.2

1. Qual o valor-verdade de cada uma das wffs a seguir na interpretação onde o domínio consiste em inteiros, $O(x)$ é "x é ímpar", $L(x)$ é "$x < 10$" e $G(x)$ é "$x > 9$"?
   a. $(\exists x)O(x)$ b. $(\forall x)[L(x) \rightarrow O(x)]$
   c. $(\exists x)[L(x) \wedge G(x)]$ d. $(\forall x)[L(x) \vee G(x)]$

2. Qual o valor-verdade de cada uma das wffs na interpretação onde o domínio consiste nos números inteiros?
   ★**a.** $(\forall x)(\exists y)(x + y = x)$ ★**b.** $(\exists y)(\forall x)(x + y = x)$
   ★**c.** $(\forall x)(\exists y)(x + y = 0)$ ★**d.** $(\exists y)(\forall x)(x + y = 0)$
   **e.** $(\forall x)(\forall y)(x < y \vee y < x)$ **f.** $(\forall x)[x < 0 \rightarrow (\exists y)(y > 0 \wedge x + y = 0)]$
   **g.** $(\exists x)(\exists y)(x^2 = y)$ **h.** $(\forall x)(x^2 > 0)$

3. Indique o valor-verdade de cada uma das wffs a seguir na interpretação onde o domínio consiste nos estados do Brasil, $Q(x, y)$ é " x é ao norte de y", $P(x)$ é "$x$ começa com a letra P" e $a$ é "Paraná".
   a. $(\forall x)P(x)$ b. $(\forall x)(\forall y)(\forall z)[Q(x, y) \wedge Q(y, z) \rightarrow Q(x, z)]$
   c. $(\exists y)(\exists x)Q(y, x)$ d. $(\forall x)(\exists y)[P(y) \wedge Q(x, y)]$
   e. $(\exists y)Q(a, y)$

4. Para as wffs a seguir, ache uma interpretação na qual sejam verdadeiras e outra na qual sejam falsas.
   ★**a.** $(\forall x)([A(x) \vee B(x)] \wedge [A(x) \wedge B(x)]$'
   **b.** $(\forall x)(\forall y)[P(x, y) \rightarrow P(y, x)]$
   **c.** $(\forall x)[P(x) \rightarrow (\exists y)Q(x, y)]$
   **d.** $(\exists x)[A(x) \wedge (\forall y)B(x, y)]$
   **e.** $[(\forall x)A(x) \rightarrow (\forall x)B(x)] \rightarrow (\forall x)[A(x) \rightarrow B(x)]$

5. Identifique o escopo de cada um dos quantificadores nas wffs a seguir e indique quaisquer variáveis livres.
   **a.** $(\forall x)[P(x) \rightarrow Q(y)]$
   **b.** $(\exists x)[A(x) \wedge (\forall y)B(y)]$
   **c.** $(\exists x)[(\forall y)P(x, y) \wedge Q(x, y)]$
   **d.** $(\exists x)(\exists y)[A(x, y) \wedge B(y, z) \rightarrow A(a, z)]$

6. Com o uso de símbolos predicados mostrados e os quantificadores apropriados, escreva cada sentença na língua portuguesa como uma wff predicativa. (O domínio é todo o mundo.)

   D(x) *é* "x é um dia." — *S* é "segunda-feira."
   *S(x)* é "x é ensolarado." — *T* é "terça-feira."
   *R(x)* é *"x é* chuvoso."

   ★a. Todos os dias são ensolarados.
   ★b. Alguns dias não são chuvosos.
   ★c. Todo dia que é ensolarado não é chuvoso.
   d. Alguns dias são ensolarados e chuvosos.
   e. Nenhum dia é ensolarado e chuvoso.
   f. Sempre é dia ensolarado se, e somente se, é um dia chuvoso.
   g. Nenhum dia é ensolarado.
   h. Segunda-feira foi ensolarada, portanto todo dia será ensolarado.
   i. Tanto segunda-feira quanto terça-feira foram chuvosos.
   j. Se algum dia for chuvoso, então todos os dias serão ensolarados.

7. Com o uso de símbolos predicados mostrados e os quantificadores apropriados, escreva cada sentença da língua portuguesa como uma wff predicativa. (O domínio é todo o mundo.)

   *J(x)* é "x é um juiz." — *Q(x) é "x é* um químico."
   *A(x) é* "x é um advogado." — *A(x, y) é "x* admira y."
   *M(x)* é "x é uma mulher."

   a. Existem algumas mulheres advogadas que são químicas.
   b. Nenhuma mulher é advogada e química.
   ★c. Alguns advogados só admiram juizes.
   d. Todos os juizes admiram apenas juizes.
   e. Apenas juizes admiram juizes.
   f. Todas as mulheres advogadas admiram algum juiz.
   g. Algumas mulheres não admiram advogados.

8. Usando os símbolos predicados mostrados e os quantificadores apropriados, escreva as sentenças na língua portuguesa como wffs predicativas. (O domínio é todo o mundo.)

   *C(x)* é "x é uma Corvette." — *P(x)* é "x é um Porsche."
   *F{x)* é "x é uma Ferrari." — *L(x, y)* é "x é mais lento que v."

   ★a. Nada é, ao mesmo tempo, uma Corvette e uma Ferrari.
   ★b. Alguns Porsches são apenas mais lentos que as Ferraris.
   c. Apenas Corvettes são mais lentas que Porsches.
   d. Todas as Ferraris são mais lentas que alguma Corvette.
   e. Nenhum Porsche é mais lento que a Corvette.
   f. Se existir uma Corvette que seja mais lenta que uma Ferrari, então todas as Corvettes serão mais lentas que todas as Ferraris.

9. Usando os símbolos predicados mostrados e os quantificadores apropriados, escreva as sentenças na língua portuguesa como wffs predicativas. (O domínio é todo o mundo.)

   *A(x) é* "x é uma abelha."
   *F(x)* é "x é uma flor."
   G(x) é "x gosta de y."

a. Todas as abelhas gostam de todas as flores.
b. Algumas abelhas gostam de todas as flores.
c. Todas as abelhas gostam de algumas flores
d. Toda abelha só odeia flores.
e. Apenas abelhas gostam de flores.
f. Toda abelha só gosta de flores.
g. Nenhuma abelha gosta só de flores.
h. Algumas abelhas gostam de algumas flores.
i. Algumas abelhas gostam apenas de flores.
j. Toda abelha odeia algumas flores.
k. Toda abelha odeia todas as flores.
1. Nenhuma abelha odeia todas as flores.

10. Se

*B(x)* for "x é bonito."
*E(x)* for "x é elegante."
*G(x,* y) for "*x* gosta de y."
*H(x)* for "*x* é um homem."
*M(x)* for "*x* é uma mulher."
*j* for "John."
*k* for "Kathy."

dê as traduções para a língua portuguesa das wffs a seguir:

★a. $E(j) \wedge G(k, j)$
★b. $(\forall x)[H(x) \rightarrow E(x)]$
c. $(\forall x)(M(x) \rightarrow (\forall y)[G(x, y) \rightarrow H(y) \wedge E(y)])$
d. $(\exists x)[H(x) \wedge E(x) \wedge G(x, k)]$
e. $(\exists x)(M(x) \wedge B(x) \wedge (\forall y)[G(x, y) \rightarrow E(y) \wedge H(y)])$
f. $(\forall x)[M(x) \wedge B(x) \rightarrow G(j, x)]$

11. Diversas formas de negação são apresentadas para cada uma das sentenças a seguir. Qual é a correta?
a. Algumas pessoas gostam de Matemática.
1. Algumas pessoas não gostam de Matemática.
2. Todo o mundo não gosta de Matemática.
3. Todo o mundo gosta de Matemática.
b. Todo o mundo gosta de sorvete.
1. Ninguém gosta de sorvete.
2. Todo o mundo não gosta de sorvete.
3. Alguém não gosta de sorvete.
c. Todo o mundo é alto e magro.
1. Alguém é baixo e gordo.
2. Ninguém é alto e magro.
3. Alguém é baixo ou gordo.
d. Alguns retratos estão velhos ou apagados.
1. Nenhum retrato está velho ou apagado.
2. Alguns retratos não estão velhos ou apagados.
3. Todos os retratos não estão velhos ou não estão apagados.

12. Explique por que as wffs são válidas.
a. $(\forall x)(\forall y)A(x, y) \leftrightarrow (\forall y)(\forall x)A(x, y)$
b. $(\exists x)(\exists y)A(x, y) \leftrightarrow (\exists y)(\exists x)A(x, y)$
c. $(\exists x)(\forall y)P(x, y) \rightarrow (\forall y)(\exists x)P(x, y)$
d. $A(a) \rightarrow (\exists x)A(x)$
e. $(\forall x)[A(x) \rightarrow B(x)] \rightarrow [(\forall x)A(x) \rightarrow (\forall x)B(x)]$

13. Forneça interpretações que provem que as wffs a seguir não são válidas:
★a. $(\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)[A(x) \wedge B(x)]$
b. $(\forall x)(\exists y)P(x, y) \rightarrow (\exists x)(\forall y)P(x, y)$
c. $(\forall x)[P(x) \rightarrow Q(x)] \rightarrow [(\exists x)P(x) \rightarrow (\forall x)Q(x)]$
d. $(\forall x)[A(x)]' \leftrightarrow [(\forall x)A(x)]'$

14. Determine quais wffs são válidas ou inválidas. Justifique sua resposta.
a. $(\exists x)A(x) \leftrightarrow ((\forall x)[A(x)]')'$
b. $(\forall x)P(x) \vee (\exists x)Q(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$
c. $(\forall x)A(x) \leftrightarrow ((\exists x)[A(x)]')'$
d. $(\forall x)[P(x) \vee Q(x)] \rightarrow (\forall x)P(x) \vee (\exists y)Q(y)$

# Seção 1.3 Lógica Proposicional

## Sistemas Formais

Resultados matemáticos são normalmente chamados teoremas. Os sistemas da lógica formal manipulam fórmulas como as das Seções 1.1 e 1.2 e atribuem um significado preciso à palavra teorema. Nestes sistemas, certas wffs são aceitas como **axiomas-wffs** que não precisam ser provadas. Um axioma deve, portanto, ser uma wff cuja "verdade" seja evidente. Então, pelo menos, um axioma deve ser uma tautologia ou, se envolve predicados, uma wff válida. Além dos axiomas, sistemas formais contêm regras de inferência. Uma **regra de inferência** é uma convenção que permite a uma nova wff de uma certa forma ser inferida, ou deduzida, de uma a duas outras wffs de uma certa forma. Uma seqüência de wffs na qual cada wff seja ou um axioma ou o resultado da aplicação de uma das regras de inferência às wffs anteriores na seqüência é chamada de **seqüência de prova. Um teorema** é o último componente desta seqüência; a seqüência é a **prova** do teorema.

O esboço a seguir é uma prova típica de um teorema

wff 1 (um axioma)
wff2 (um axioma)
wff3 (inferida da wff 1 e da wff2 por uma regra de inferência)
wff4 (um axioma)
wff5 (inferida da wff4 por uma regra de inferência)
wff6 (inferida da wff3 e wff5 por uma regra de inferência)

A última wff na seqüência, wff6, é o teorema, e a seqüência toda constitui sua prova. (Naturalmente as outras wffs também podem ser teoremas — bastaria termos parado a seqüência nelas.)

Outra condição que incluímos em nosso sistema, além da que os axiomas devam ser tautologias ou wffs válidas é que haja o mínimo de axiomas e de regras quanto for possível. Convém minimizar o número de sentenças que devemos aceitar sem prova, mesmo quando essas sentenças parecem óbvias.

A escolha do conjunto de axiomas e das regras de inferência deve ser cuidadosa. Se escolhermos poucos axiomas, poucas regras de inferência ou regras de inferências fracas, não seremos capazes de provar algumas wffs que são "verdadeiras" e que, portanto, podem ser teoremas. Por outro lado, se escolhermos muitos axiomas, muitas regras de inferência ou regras de inferência muito fortes, seremos capazes de chegar a praticamente qualquer wff e provar que é um teorema, inclusive wffs que não são "verdadeiras" e, portanto, não podem ser teoremas. Como conseguimos exatamente o necessário para os teoremas?

Para termos idéia da palavra intuitiva *verdadeiro,* consideramos dois sistemas lógicos formais — um para as wffs proposicionais e outro para as wffs predicativas. O sistema formal que usa as wffs proposicionais é chamado **lógica proposicional, lógica de sentenças** ou **cálculo proposicional.** O sistema formal que usa wffs predicadas é chamado **lógica predicada** ou **cálculo predicado.** (A palavra *cálculo* é usada aqui no sentido mais geral de "avaliação" ou "raciocínio", e não no sentido de "diferenciação" ou "integração".) Estudaremos lógica proposicional a seguir e lógica de predicados na Seção 1.4.

## Lógica Proposicional

Na lógica proposicional, as wffs são formadas de símbolos proposicionais, conectivos lógicos e símbolos de agrupamento. Neste sistema, uma wff "verdadeira" significa uma tautologia. Desejamos, portanto, os axiomas e as regras de inferência que nos permitirão provar todas as tautologias, e apenas as tautologias, como teoremas.

Tomemos as seguintes wffs como axiomas, onde *P, Q e R* representam wffs proposicionais:

1. $P \rightarrow (Q \rightarrow P)$
2. $[P \rightarrow (Q \rightarrow R)] \rightarrow [(P \rightarrow Q) \rightarrow (P \rightarrow R)]$
3. $(Q' \rightarrow P') \rightarrow (P \rightarrow Q)$

Podemos mostrar que cada uma delas é uma tautologia, propriedade que desejamos para nossos axiomas. Não está claro por que esses axiomas em particular devem ser escolhidos ou como devem ser usados; por enquanto, podemos ver que eles dizem algo sobre o processo de raciocínio. O axioma 1 diz que, dado *P,* qualquer coisa implica *P.* O axioma 2 diz que dada uma certa implicação que segue de *P,* então se o antecedente desta implicação também segue de *P,* o conseqüente também o fará. (Este é um tipo de regra de "transitividade", um pouco parecida com: Se $a > b > c$, então se $a > b$, segue que $a > c$.) O axioma 3 que se "não *Q*" implica "não *P*", então *P* implica *Q,* pois, do contrário, "não *Q*" valeria, e isto implicaria "não *P*".

Como *P, Q e R* podem ser wffs compostas, cada axioma dado acima é, na verdade, um padrão, ou esquema, para um número finito de wffs. Portanto,

$$(A \rightarrow B) \rightarrow [(C \wedge D) \rightarrow (A \rightarrow B)]$$

é um axioma, pois se encaixa no esquema do axioma 1, onde *P* é a wff $A \rightarrow B$ e *Q* é a wff $C \wedge D$. Mas ao contrário de um pequeno número de axiomas, isto não indica que temos um número infinito de axiomas? Sim, mas existem apenas três *formas* para o os axiomas.

Na lógica proposicional, existe apenas uma regra de inferência: Das wffs *P eP* $\rightarrow$ *Q*, podemos inferir a wff *Q*. (Esta regra de inferência é conhecida pelo seu nome latino de **modus ponens,** que significa "método de afirmação".)

A wff $A \rightarrow A$, onde *A é um* símbolo proposicional, é uma tautologia. Portanto, usando nossos axiomas e a regra de inferência, esperamos ser capazes de provar que ela é um teorema. O exemplo a seguir nos dá a seqüência de prova desta wff. Apesar de a wff ser bem simples, a prova é difícil; em particular, não fica absolutamente claro por onde começar. Não se aflija quanto a isto, pois logo teremos uma técnica que nos ajudará a contornar esta dificuldade. O objetivo do Exemplo 13 é tão-somente ilustrar o uso dos vários axiomas e da regra de inferência em uma seqüência de prova. A justificativa dada para cada passo elucida seu uso, mas elas não constituem parte da seqüência de prova propriamente dita.

**EXEMPLO 13** A wff $A \rightarrow A$ é um teorema. Uma seqüência de prova é a seguinte:

| | |
|---|---|
| **1.** $A \rightarrow [(A \rightarrow A) \rightarrow A)] \rightarrow$ $[A \rightarrow (A \rightarrow A)] \rightarrow (A \rightarrow A)]$ | (Axioma 2 com $P = A$, $Q = A \rightarrow A$, $R = A$) |
| **2.** $A \rightarrow [(A \rightarrow A) \rightarrow A]$ | (Axioma 1 com $P = A$, $Q = A \rightarrow A$) |
| **3.** $[A \rightarrow (A \rightarrow A)] \rightarrow (A \rightarrow A)$ | (de 1 e 2 pelo modus ponens) |
| **4.** $A \rightarrow (A \rightarrow A)$ | (Axioma 1 com $P = A$, $Q = A$) |
| **5.** $A \rightarrow A$ | (de 3 e 4 pelo modus ponens) • |

Uma seqüência de prova semelhante poderia mostrar que $P \rightarrow P$ *é* um teorema para qualquer wff *P;* portanto, o que temos é um esquema de teorema. Em geral, o que faremos é provar esquemas de teoremas ao invés de teoremas individuais.

Os axiomas que escolhemos envolvem apenas implicação e negação. Para wffs que contenham os conectivos de disjunção e conjunção, usamos as equivalências

$$A \vee B \Leftrightarrow A' \rightarrow B \qquad e \qquad A \wedge B \Leftrightarrow (A \rightarrow B')'$$

e nos contentamos em provar as wffs equivalentes que resultam. De fato, poderíamos ter definido a disjunção e conjunção em termos da implicação e negação. Então todas as nossas wffs teriam envolvido apenas os conectivos de implicação e negação.

Apesar de não provarmos isto aqui, este sistema de axiomas e uma regra de inferência fazem exatamente o que desejamos — toda tautologia é um teorema (i.e., tem uma prova), e vice-versa. Esta propriedade é descrita dizendo que nosso sistema formal é **completo** (tudo o que deveria ser um teorema, o é) e **correto** (nada que não deveria ser um teorema não o é).

Permitimos abreviações nas seqüências de provas através do uso de teoremas já provados. Uma vez que *T* já tenha sido provada por um teorema, então *T* pode servir como passo em outra seqüência de prova. Isto porque *T* tem sua própria seqüência de prova, que poderia ser substituída na seqüência de prova que estamos construindo.

## Deduções

Freqüentemente desejamos provar, como teoremas, wffs da forma $P \rightarrow Q$, onde *P* e *Q* são wffs compostas. *P* é chamada de **hipótese** do teorema e *Q,* de **tese.** $P \rightarrow Q$ *é* também uma implicação, onde, de acordo com a terminologia usada, *P* é o antecedente e *Q* o conseqüente. No entanto, nem toda implicação é um teorema. Para que o seja, o antecedente e o conseqüente precisam estar relacionados estruturalmente, de forma que a implicação como um todo seja uma tautologia. A implicação $A \rightarrow B$ para os símbolos proposicionais *A* e *B,* por exemplo, não é uma tautologia e portanto não é um teorema. Portanto A, neste caso, é um antecedente mas não uma hipótese; 6 é um conseqüente, mas não uma tese. No entanto, a implicação $A \wedge B \rightarrow B \wedge A$ é uma tautologia e, portanto, é um teorema. Neste caso, $A \wedge B$ *é* tanto um antecedente como uma hipótese e $B \wedge A$ é tanto um conseqüente quanto uma conclusão.

Se $P \rightarrow Q$ *é* um teorema, ele deve ser uma tautologia e sempre que *P* for verdadeira, *Q* também o deve ser. Intuitivamente, imaginamos ser possível deduzir *Q* a partir de *P.* Formalmente, definimos uma **dedução**

de $Q$ a partir de $P$ como uma seqüência de wffs terminando em $Q$ onde cada wff é um axioma ou é a wff $P$ ou ainda é derivada das wffs anteriores através das regras de inferência. De fato, esta é a prova de um teorema, onde aceitamos $P$ como um axioma. Podemos mostrar que $P \rightarrow Q$ *constitui um teorema se, e somente se, Q for dedutível a partir de P.* Nossa técnica para demonstrar teoremas da forma $P \rightarrow Q$ é, portanto, incluir a hipótese como uma das wffs na seqüência e concluir a seqüência com $Q$.

Para mostrar a potencialidade do método de dedução, refaremos a prova do teorema do Exemplo 13, usando uma wff $P$ arbitrária.

**EXEMPLO 14** Usando a lógica proposicional, prove o teorema $P \rightarrow P$.

1. $P$ (hipótese)

Como $P$ não é apenas a hipótese, mas também a tese, a prova está completa. Uma prova confusa de cinco linhas se resumiu a uma prova de uma única linha! •

Naturalmente, mesmo com o método dedutivo, a maioria das demonstrações não será tão trivial como no Exemplo 14, mas sempre temos um início ao incluirmos a hipótese.

**EXEMPLO 15** Usando a lógica proposicional, prove o teorema

$$[P \rightarrow (P \rightarrow Q)] \rightarrow (P \rightarrow Q)$$

A hipótese é $P \rightarrow (P \rightarrow Q)$, portanto, citá-la será o primeiro passo da demonstração.

**1.** $P \rightarrow (P \rightarrow Q)$ (hipótese)

Agora, consideremos o que precisamos fazer para sair deste ponto e chegar à tese desejada, $P \rightarrow Q$. Se introduzíssemos $P$ como um passo da demonstração, o modus ponens e o passo 1 nos permitiria concluir $P \rightarrow Q$. No entanto, neste ponto ainda não temos razão para este passo. Por isso procuraremos um axioma que nos permita usar o passo 1 e o modus ponens. O Axioma 2 tem uma hipótese que se assemelha ao passo 1, desde que façamos, no Axioma 2, $P = P$, $Q = P$ *e* $R = Q$. O segundo passo da demonstração é, portanto

**2.** $[P \rightarrow (P \rightarrow Q)] \rightarrow [(P \rightarrow P) \rightarrow (P \rightarrow Q)]$ (Axioma 2)

e, então

**3.** $(P \rightarrow P) \rightarrow (P \rightarrow Q)$ (1,2, modus ponens)

Agora, se pudéssemos introduzir $P \rightarrow P$ como um passo da demonstração, o modus ponens e o passo 3 nos dariam a conclusão $P \rightarrow Q$. M *Pa→sP* é o teorema provado no Exemplo 14 e, portanto, pode ser inserido aqui. A prova completa é a seguinte:

| | |
|---|---|
| **1.** $P \rightarrow (P \rightarrow Q)$ | (hipótese) |
| **2.** $[P \rightarrow (P \rightarrow Q)] \rightarrow [(P \rightarrow P) \rightarrow (P \rightarrow Q)]$ | (Axioma 2) |
| **3.** $(P \rightarrow P) \rightarrow (P \rightarrow Q)$ | (1,2, modus ponens) |
| **4.** $P \rightarrow P$ | (Exemplo 14) |
| **5.** $P \rightarrow Q$ | (3, 4, modus ponens) |

•

**PRÁTICA 14** Usando a lógica proposicional, demonstre o teorema

$$P' \rightarrow (P \rightarrow Q)$$

*(Dica:* Use os Axiomas 1 e 3.) •

O método de dedução pode ser estendido a fim de ser tornado ainda mais poderoso. Se a hipótese do teorema for uma série de conjunções $P_1 \wedge P_2 \wedge \ldots \wedge P_n$, simplesmente incluímos cada fator da hipótese como uma wff na seqüência da demonstração e deduzimos a tese dela. Por último, se a tese for, ela própria, uma implicação $R \rightarrow S$, podemos incluir $R$ na seqüência da demonstração, tornando-a parte da hipótese, e apenas deduzir $S$. (Isto é coerente com nosso entendimento da implicação, mas veja o Exercício 18 ao fim desta seção para uma justificativa formal.)

Para resumir, o método de dedução permite as seguintes abordagens:

1. Para provar o teorema $P \rightarrow Q$, deduzimos $Q$ a partir de $P$.
2. Para provar o teorema $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow Q$, deduzimos $Q$ a partir de $P_1, P_2, ..., P_n$.
3. Para provar o teorema $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow (R \rightarrow S)$, deduzimos $S$ de $P_1, P_2, ..., P_n, R$.

EXEMPLO 16 Uma prova para o teorema

$$(P' \rightarrow Q') \wedge (P \rightarrow S) \rightarrow (Q \rightarrow S)$$

usando a lógica proposicional é

| | |
|---|---|
| **1.** $P' \rightarrow Q'$ | (hipótese) |
| **2.** $P \rightarrow S$ | (hipótese) |
| **3.** $Q$ | (hipótese) |
| **4.** $(P' \rightarrow Q') \rightarrow (Q \rightarrow P)$ | (Axioma 3) |
| **5.** $Q \rightarrow P$ | (1,4, modus ponens) |
| **6.** $P$ | (3, 5, modus ponens) |
| **7.** $S$ | (2, 6, modus ponens) |

•

Um teorema da forma $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow Q$ indica que $Q$ pode ser deduzido de $P_1, P_2, ..., P_n$. Portanto, por exemplo, em uma seqüência de demonstração contendo as wffs $P' \rightarrow Q'$ e $P \rightarrow S$, podemos inserir a sentença $Q \rightarrow S$ e citar o teorema do Exemplo 16 como justificativa.

PRÁTICA 15 Encontre uma demonstração mais curta para o teorema do Exemplo 15. •

PRÁTICA 16 Use a lógica proposicional para provar o teorema

$$(P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

•

### Argumentos Válidos

Em português, um argumento (a acusação de um advogado, uma propaganda ou um discurso de um político) é normalmente apresentado como uma série de sentenças que podem ser simbolizadas por $P_1, P_2, ..., P_n$ seguidas de uma conclusão $Q$. O argumento é um **argumento válido** se a conclusão puder ser deduzida, do ponto de vista da lógica, da conjunção $P_1 \wedge P_2 \wedge ... \wedge P_n$ — em outras palavras, se $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow Q$ for um teorema.

EXEMPLO 17 O argumento a seguir é válido? "Meu cliente é canhoto, mas o diário não desapareceu, então meu cliente não é canhoto; portanto, o diário desapareceu." Existem apenas duas sentenças envolvidas aqui, que simbolizaremos como:

*C:* Meu cliente é canhoto.
*D:* O diário desapareceu.

O argumento é, portanto,

$$[C \wedge (D' \rightarrow C')] \rightarrow D$$

A validade do argumento é estabelecida pela seguinte prova:

| | |
|---|---|
| **1.** $C$ | (hipótese) |
| **2.** $D' \rightarrow C'$ | (hipótese) |
| **3.** $(D' \rightarrow C') \rightarrow (C \rightarrow D)$ | (Axioma 3) |
| **4.** $C \rightarrow D$ | (2, 3, modus ponens) |
| **5.** $D$ | (1,4, modus ponens) |

Perceba que a validade do argumento é uma função apenas de sua forma lógica, e não tem nada a ver com o fato de seus componentes serem ou não verdadeiros. Continuamos a não ter qualquer idéia sobre se o diário está realmente desaparecido ou não. Além disso, o argumento "Skoozes são rosa, mas se Gingoos não gosta-

rem de perskees, os skoozes não serão rosa; portanto, Gingoos não gostam de perkees", que têm a mesma forma, é também válido, apesar de não fazer qualquer sentido. •

Como qualquer tautologia também é um teorema na lógica proposicional, podemos inserir uma tautologia em qualquer passo de uma demonstração. Isto também constitui uma simplificação válida para resumir diversos passos de uma demonstração em um único passo, desde que esteja bem claro que não foram violadas quaisquer regras. Por exemplo, ao invés de

**1.** $P \wedge Q$ (hipótese)
**2.** $P \wedge Q \rightarrow Q$ (tautologia)
**3.** $Q$ (1,2, modus ponens)

poderíamos escrever

1. $P \wedge Q$ (hipótese)
2. $Q$ (1, tautologia $A \wedge B \rightarrow B$, modus ponens)

Tautologias da forma $P \wedge Q \rightarrow R$ nos dizem que $R$ pode ser deduzida das hipóteses $P$ e $Q$, de forma que a demonstração possa incluir passos como

1. $P$ (hipótese)
2. $Q$ (hipótese)
3. $R$ (tautologia $P \wedge Q \rightarrow R$)

**EXEMPLO 18** Considere o argumento "Se a taxa para importação diminuir, o comércio interno aumentará. Ou a taxa federal de desconto diminuirá ou o comércio interno não irá aumentar. A taxa para importação vai diminuir. Portanto a taxa federal de desconto vai diminuir". Usando

*I*: A taxa para importação vai diminuir.
*M:* O mercado interno vai aumentar.
*F: A* taxa federal de desconto vai diminuir.

o argumento é

$$[(I \rightarrow M) \wedge (F \vee M') \wedge I] \rightarrow F$$

Uma demonstração para estabelecer a validade é:

1. $I \rightarrow M$ (hipótese)
2. $F$ v $M'$ (hipótese)
3. I (hipótese)
4. $M$ (1, 3, modus ponens)
5. $F$ (tautologia $(F \vee M') \wedge M \rightarrow F$) •

A tautologia dada como justificativa do passo 5 mostra que $F$ pode ser deduzida de $F \vee M'$ e $M$. Uma tabela-verdade separada pode ser usada para provar que $(F \vee M') \wedge M \rightarrow F$ é, de fato, uma tautologia, mas sua veracidade é intuitivamente clara — ela diz que se $F$ ou não $M$ forem verdadeiros, e $M$ for verdadeiro (portanto não $M$ *não* é verdadeiro), então a outra afirmativa, $F$, precisa ser verdadeira. •

A tautologia $A \wedge B \rightarrow A \wedge B$ é particularmente útil em demonstrações, na medida em que justifica a dedução de $A \wedge B$ das hipóteses $A$ *e* $B$ que apareceram antes na demonstração. Isto é mostrado no passo 6 do próximo exemplo.

**EXEMPLO 19** Mostre que o seguinte argumento é válido "Se usamos a linguagem assembly, então o programa será executado mais rapidamente. Se usamos a linguagem assembly, o programa terá mais linhas de código. Portanto, se usamos a linguagem assembly, então o programa será executado mais rapidamente e terá mais linhas de código." Usando

*A:* Usamos a linguagem assembly.
*R:* O programa será executado mais rapidamente.
*L:* O programa terá mais linhas de código.

o argumento é $(A \rightarrow R) \wedge (A \rightarrow L) \rightarrow [A \rightarrow (R \wedge L)]$ e a prova é

| | | |
|---|---|---|
| **1.** | $A \rightarrow R$ | (hipótese) |
| **2.** | $A \rightarrow L$ | (hipótese) |
| **3.** | $A$ | (hipótese) |
| **4.** | $R$ | (1, 3, modus ponens) |
| **5.** | $L$ | (2, 3, modus ponens) |
| **6.** | $R \wedge L$ | (4, 5, $A \wedge B$ pode ser deduzido de $A$ e $B$) |

•

PRÁTICA 1 7 Mostre que o argumento a seguir é válido usando as letras *P*,*M* e C: "Se o produto for confiável, a parcela do mercado irá aumentar. Ou o produto é confiável ou os custos irão subir. A parcela de mercado não irá aumentar. Portanto os custos irão subir." •

Nossa versão da lógica proposicional inclui apenas uma regra de inferência, a modus ponens, que permite que $Q$ seja inserida como uma wff na demonstração a partir da verificação anterior de $P$ e $P \rightarrow Q$. Esta regra de inferência permite que se criem seqüências de demonstração da seguinte forma:

| | | |
|---|---|---|
| 1. | $P \rightarrow Q$ | (hipótese) |
| 2. | $Q'$ | (hipótese) |
| 3. | $(P \rightarrow Q) \wedge Q'$ | *(1,2A ∧ B*pode ser deduzida de *A* e *B)* |
| 4. | $P'$ | (3, tautologia $(A \rightarrow B) \wedge B' \rightarrow A'$, modus ponens) |

Algumas versões da lógica proposicional incluem uma outra regra de inferência que reproduz este processo. A **modus tollens** ("método de negação") diz: das wffs $P \rightarrow Q$ *e* $Q'$, podemos inferir a wff $P'$. O sistema formal pode ser estendido para incluir outras regras de inferência. O quadro a seguir ilustra algumas regras de inferências e os nomes usualmente atribuídos a elas. Nós delineamos a lógica proposicional com o uso apenas do modus ponens porque qualquer coisa deduzida com o uso das demais regras de inferência listadas no quadro pode ser deduzida em nosso sistema usando as tautologias apropriadas e o modus ponens, da mesma forma que a dedução acima poderia ser substituída por uma aplicação do modus tollens.

**Regras de Interferência para Lógica Proposicional**

| De: | Podemos inferir | Nome |
|---|---|---|
| $P$ e $P \rightarrow Q$ | $Q$ | **Modus ponens** (nossa única regra de inferência) |
| $P \rightarrow Q$ e $Q'$ | $P'$ | **Modus tollens** |
| $P \vee Q$ e $Q'$ | $p$ | Silogismo disjuntivo |
| $P \rightarrow Q$ e $Q \rightarrow R$ | $P \rightarrow R$ | Silogismo hipotético |
| $P$ e $Q$ | $P$ | Simplificação conjuntiva |
| $P$ | $P \vee Q$ | Amplicação disjuntiva |

## Uma Preliminar

Podemos ter tido o seguinte pensamento: Se teoremas são como tautologias, então para mostrar que uma wff é um teorema, por que, para mostrar que uma wff é um teorema, não mostramos apenas que é uma tautologia? A determinação de se uma wff é uma tautologia pode ser feita através da construção da tabela-verdade ou do algoritmo *TestaTautologia* da Seção 1.1. Isto seria uma tarefa mais mecânica do que construir uma demonstração. Então por que falamos de provas?

Se fôssemos nos restringir à lógica proposicional, não teríamos a necessidade do conceito de provas formais. No entanto, sabemos que nem todas as sentenças podem ser simbolizadas dentro da lógica proposicional, e que não há procedimento mecânico (como a construção da tabela-verdade) para determinar a validade de wffs predicativas. Uma técnica de prova formal precisa ser usada; e o que fizemos aqui para lógica proposicional pode servir como base para o que será feito na lógica de predicados.

## Revisão da Seção 1.3

**Técnicas**

- Prova de teoremas na lógica proposicional
- Uso da lógica proposicional para provar a validade de um argumento na língua portuguesa

**Idéias Principais**

A prova de sistemas formais consiste em seqüências de hipóteses, axiomas ou wffs inferidas das wffs anteriores da seqüência.
Todo teorema na lógica proposicional é uma tautologia, e toda tautologia é um teorema.
Em argumentos válidos, a tese pode ser deduzida da conjunção das demais sentenças.

## Exercícios 1.3

★1. Justifique cada um dos passos na demonstração de $(P \rightarrow Q) \wedge [P \rightarrow (Q \rightarrow R)] \rightarrow (P \rightarrow R)$ a seguir:

1. $P \rightarrow Q$
2. $P \rightarrow (Q \rightarrow R)$
3. $[P \rightarrow (Q \rightarrow R)] \rightarrow [(P \rightarrow Q) \rightarrow (P \rightarrow R)]$
4. $(P \rightarrow Q) \rightarrow (P \rightarrow R)$
5. $P \rightarrow R$

2. Justifique cada passo na demonstração de $[(P \rightarrow Q) \rightarrow P] \rightarrow [(P \rightarrow Q) \rightarrow Q]$ a seguir:

**1.** $(P \rightarrow Q) \rightarrow (P \rightarrow Q)$
**2.** $[(P \rightarrow Q) \rightarrow (P \rightarrow Q)]$
$([(P \rightarrow Q) \rightarrow P] \rightarrow [(P \rightarrow Q) \rightarrow Q])$
**3.** $[(P \rightarrow Q) \rightarrow P] \rightarrow [(P \rightarrow Q) \rightarrow Q]$

Nos Exercícios de 3 a 12, demonstre que cada wff é um teorema da lógica proposicional. Você pode usar qualquer teorema demonstrado anteriormente, incluindo os exercícios anteriores.

**3.** $[(Q' \rightarrow P') \wedge P \wedge (Q \rightarrow R)] \rightarrow R$
★**4.** $P' \wedge (P \vee Q) \rightarrow Q$
**5.** $[(P \rightarrow (Q \rightarrow R)) \wedge (P \vee S') \wedge Q] \rightarrow (S \rightarrow R)$
**6.** $P \rightarrow (P \vee Q)$
★**7.** $(P')' \rightarrow P$
**8.** $P \rightarrow (P')'$
**9.** $(P \rightarrow Q) \rightarrow (Q' \rightarrow P')$
★**10.** $P \vee Q \rightarrow Q \vee P$
**11.** $P' \wedge (Q \rightarrow P) \rightarrow Q'$
**12.** $[P \rightarrow (Q \rightarrow R)] \rightarrow (Q \rightarrow (P \rightarrow R))$

Usando a lógica proposicional, prove que os argumentos dos Exercícios 13 a 17 são válidos. Use os símbolos proposicionais indicados.

13. Se o programa é eficiente, ele executará rapidamente: Ou o programa é eficiente ou ele tem um erro. No entanto, o programa não executa rapidamente. Portanto o programa tem um erro. *(E, R, B)*

14. A colheita é boa, mas não há água suficiente. Se tivesse bastante água ou não tivesse bastante sol, então haveria água suficiente. Portanto, a colheita é boa e há bastante sol. (C, *A, H, S)*

★15. Rússia tinha um poder superior, e ou a França não era forte ou Napoleão cometeu um erro. Napoleão não cometeu um erro, mas se o exército não tivesse falhado, a França seria forte. Portanto, o exército falhou e a Rússia tinha um poder superior. *(R, F, N, E)*

16. Não é verdade que se as taxas de eletricidade subirem, o consumo diminuirá, nem é verdade que novas usinas de energia serão construídas ou as contas não serão atrasadas. Portanto o consumo não diminuirá e as contas serão atrasadas. *(T, C, U, Co)*

17. Se José pegou as jóias ou a sra. Krasov mentiu, então ocorreu um crime. O sr. Krasov não estava na cidade. Se ocorreu um crime, então o sr. Krasov estava na cidade. Portanto José não pegou as jóias, (*J, M, C, E)*

★18. a. Use uma tabela-verdade para verificar que $A \rightarrow (B \rightarrow C) \Leftrightarrow (A \wedge B) \rightarrow C$ é uma tautologia
b. Prove que $A \rightarrow (B \rightarrow C) \Leftrightarrow (A \wedge B) \rightarrow C$ usando uma série de equivalências.
c. Explique como esta equivalência justifica a extensão do método de dedução que diz para provar $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow (R \rightarrow S)$, basta deduzir $S$ de $P_1, P_2, ..., P_n$ e $R$.

# Seção 14 Lógica de Predicados

Na lógica de predicados, as wffs são formadas de predicados, quantificadores, conectivos lógicos e símbolos de grupamento. Neste sistema, uma wff "verdadeira" significa uma wff válida — uma wff que seja válida em qualquer interpretação possível. Desejamos obter os axiomas e as regras de inferência que nos permitam provar todas as wffs válidas, e apenas as válidas, como teoremas.

## Axiomas e Regras de Inferência

Wffs predicativas que têm a mesma forma lógica que tautologias são válidas. Por exemplo, no Exemplo 11 mostramos que a wff

$$P(x) \rightarrow [Q(x) \rightarrow P(x)]$$

é válida. Ela tem a forma do axioma $P \rightarrow (Q \rightarrow P)$ no cálculo proposicional, apesar de não ser uma wff proposicional. Se permitirmos as formas dos axiomas e a regra de inferência da lógica proposicional, onde as wffs são wffs predicativas, continuaremos a obter como teoremas todas as wffs predicativas que tenham a forma de tautologias. No entanto, existem algumas wffs predicativas que não têm formas tautológicas, mas ainda assim são válidas devido à sua estrutura e ao significado dos quantificadores universal e existencial (veja o Exemplo 11). Para os termos como teoremas, nosso sistema formal usará novos axiomas e uma nova regra de inferência que lança mão do significado dos quantificadores. Os axiomas para a lógica de predicados são mostrados a seguir, onde *P,Q e R* representam wffs predicativas:

### Axiomas para a Lógica de Predicados

1. $P \rightarrow (Q \rightarrow P)$
2. $[P \rightarrow (Q \rightarrow R)] \rightarrow [(P \rightarrow Q) \rightarrow (P \rightarrow R)]$
3. $(Q' \rightarrow P') \rightarrow (P \rightarrow Q)$
4. $(\forall x)[P(x) \rightarrow Q(x)] \rightarrow [(\forall x)P(x) \rightarrow (\forall x)[Q(x)]$
5. $(\forall x)\ P(x) \rightarrow P(x)$ ou $(\forall x)P(x) \rightarrow P(a)$ onde $a$ é uma constante
6. $(\exists x)P(x) \rightarrow P(t)$ onde $t$ é uma constante ou nome de uma variável ainda não usada no decorrer da demonstração.
7. $P(x) \rightarrow (\exists x)P(x)$ ou $P(a) \rightarrow (\exists x)P(x)$ onde $a$ é uma constante e $x$ não ocorre em $P(a)$
8. $[(\exists x)P(x)]' \leftrightarrow (\forall x)[P(x)]'$

Aqui a notação *P(x)* indica que $x$ ocorre na wff, mas outras variáveis também podem ocorrer. Portanto,

$$(\forall x)(\exists y)S(x, y) \rightarrow (\exists y)S(a, y)$$

onde $a$ é uma constante, é uma instância do Axioma 5 (tome *P(x)* como sendo $(\exists y)S(x, y)$).

Como na lógica proposicional, é óbvio que estes axiomas particulares poderiam ser usados, mas podemos ver que eles são intuitivamente válidos. O Axioma 4 diz que se todos os elementos do domínio que tiverem a propriedade *P* também tiverem a propriedade *Q*, e se todos os elementos do domínio, de fato, tiverem a propriedade *P*, então todos os elementos do domínio têm a propriedade *Q*. O Axioma 5 diz que se uma propriedade for verdadeira para todos os elementos do domínio, ela será verdadeira para um $x$ arbitrário ou uma constante $a$. O Axioma 6 diz que se existe um objeto para o qual a propriedade *P* é verdadeira, podemos dar um nome a este objeto; no entanto, este nome deve ser arbitrário mas diferente de qualquer outro que já tenhamos usado na seqüência da demonstração. (Esta necessidade faz com que queiramos usar o Axioma 6 o quanto antes possível na demonstração a fim de que os outros axiomas não tenham essas restrições.) O Axioma 7 diz que se *P* for verdadeira para um valor particular, então há algum membro do domínio para o qual ela é verdadeira. O Axioma 8 confirma nossa compreensão intuitiva do significado dos quantificadores universal e

existencial; se for falso que algum elemento do domínio tem a propriedade *P*, então todo elemento do domínio não terá a propriedade *P* e vice-versa.

Os Axiomas 5, 6 e 7 podem ser usados em demonstrações, junto com o modus ponens, para remover quantificadores universais, quantificadores existenciais e incluir quantificadores existenciais, respectivamente. Uma nova regra de inferência nos permite inserir quantificadores universais, mas apenas sob devidas circunstâncias. As regras de inferência para a lógica de predicados são:

**Regras de Inferência para a Lógica de Predicados**

1. **Modus ponens:** $Q$ pode ser inferida de $P$ e $P \rightarrow Q$.
2. **Generalização:** $(\forall x)Q$ pode ser inferida de $Q$ desde que (a) $Q$ não tenha sido deduzida de qualquer hipótese na qual $x$ seja uma variável livre e (b) $Q$ não tenha sido deduzida pelo uso do Axioma 6 de uma wff da forma $(\exists y)Q(y)$ na qual $x$ seja uma variável livre.

A necessidade das restrições (a) e (b) serão discutidas rapidamente. Antes, porém, vejamos corno usar esses axiomas e as regras de inferências em demonstrações.

EXEMPLO 20 Com o uso da lógica de predicados, prove o teorema

$$(\forall x)[P(x) \wedge Q(x)] \rightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$$

No Exemplo 1 l(c) vimos que esta wff é válida, portanto, como todas as wffs válidas são teoremas, somos capazes de achar uma demonstração. Como de costume, a hipótese é nosso ponto de partida.

**1.** $(\forall x)[P(x) \wedge Q(x)]$ (hipótese)

A tese $(\forall x)P(x) \wedge (\forall x)Q(x)$ será alcançada se $(\forall x)P(x)$ e $(\forall x)Q(x)$ forem passos anteriores da seqüência. A estratégia de ataque geral deveria ser tirar o quantificador universal que aparece no passo 1, o qual fornece o acesso a $P(x)$ e $Q(x)$ e então inserir o quantificador universal separadamente para cada uma das duas wffs, usando a generalização. Uma demonstração é:

| | | |
|---|---|---|
| **1.** | $(\forall x)[P(x) \wedge Q(x)]$ | (hipótese) |
| **2.** | $P(x) \wedge Q(x)$ | (1, Axioma 5, modus ponens) |
| **3.** | $P(x)$ | (2, tautologia $A \wedge B \rightarrow A$, modus ponens) |
| **4.** | $Q(x)$ | (2, tautologia $A \wedge B \rightarrow B$, modus ponens) |
| **5.** | $(\forall x)P(x)$ | (3, generalização) |
| **6.** | $(\forall x)Q(x)$ | (4, generalização) |
| **7.** | $(\forall x)P(x) \wedge (\forall x)Q(x)$ | (5, 6, $A \wedge B$ pode ser deduzida de $A$ e $B$) |

Como mostrado no passo 5, a generalização foi aplicada a $P(x)$, que foi deduzida de $(\forall x)[P(x) \wedge Q(x)]$. Como x não é livre em $(\forall x)[P(x) \wedge Q(x)]$, a condição (a) da generalização é satisfeita. A condição (b) não se aplica. O passo 6 também é uma aplicação válida da generalização. •

PRÁTICA 18 Use a lógica de predicados para provar o teorema

$$(\forall x)P(x) \rightarrow (\exists x)P(x)$$ •

A lógica de predicados, assim como a lógica proposicional, pode-se mostrar, é completa é correta — toda wff válida é um teorema e todo teorema é uma wff válida. Este sistema de axiomas e regras de inferências permite que exatamente as wffs corretas sejam provadas, mas outro conjunto de axiomas e de regras de inferência também poderia ser usado. Na verdade, mesmo neste sistema, os Axiomas 6 e 7 não são realmente necessários; poderíamos simplesmente ter definido o quantificador existencial em função do quantificador universal (a essência do Axioma 8) e então ter provado os Axiomas 6 e 7. No entanto, a inclusão desses axiomas simplifica nosso trabalho.

As restrições usadas na generalização, no entanto, são necessárias. Sem a restrição (a), a seqüência

| | | |
|---|---|---|
| **1.** | $P(x)$ | (hipótese) |
| **2.** | $(\forall x)P(x)$ | (1, generalização incorreta) |

poderia ser uma prova da wff $P(x) \rightarrow (\forall x)P(x)$, que não é uma wff válida. O elemento $x$ do domínio pode ter a propriedade $P$, mas isto não significa que cada elemento do domínio tenha a propriedade $P$. Sem a restrição (b) a seqüência

| | | |
|---|---|---|
| **1.** | $(\forall x)(\exists y)Q(x, y)$ | (hipótese) |
| **2.** | $(\exists y)Q(x, y)$ | (1, Axioma 5, modus ponens) |
| **3.** | $Q(x, y)$ | (2, Axioma 6, modus ponens) |
| **4.** | $(\forall x)Q(x, t)$ | (3, generalização incorreta) |

seria uma prova da wff $(\forall x)(\exists y)Q(x, y) \rightarrow (\forall x)Q(x, t)$, que também não é uma wff válida. Por exemplo, a interpretação onde o domínio consiste em inteiros e $Q(x, y)$ significa que $x + y = 0$, então é verdade que para cada inteiro $x$ existe um inteiro $y$ (o simétrico de $x$) tal que $x + y = 0$. No entanto, se $t$ é um elemento fixo particular do domínio, então não é verdade que a soma do mesmo inteiro $t$ a qualquer $x$ resultará em zero.

Pode ser mostrado que se $P \leftrightarrow Q$ for válida, então $Q$ pode ser substituída por $P$ dentro de uma expressão em uma dedução ou seqüência de demonstração. O uso desta expressão doravante simplifica as provas. (Mencionamos uma idéia desta prova antes como uma forma de eliminar os conectivos de disjunção e conjunção.)

**EXEMPLO 21** A wff

$$(\forall x)[P(x) \vee Q(x)] \rightarrow (\exists x)P(x) \vee (\forall x)Q(x)$$

é um teorema da lógica de predicados?

Aqui precisamos, antes de mais nada, refletir se a wff parece ou não válida. Se nos parecer válida, tentaremos achar uma prova para ela; se não, tentaremos achar uma interpretação na qual ela seja falsa. Esta wff diz que se todo elemento do domínio tiver a propriedade $P$ ou a propriedade $Q$, então pelo menos um elemento do domínio tem a propriedade $P$ ou todos elementos do domínio têm a propriedade $Q$. Isto parece bem razoável, portanto procuraremos uma prova. Os dois primeiros passos na seqüência da demonstração incluem a hipótese e a reescreve em uma forma mais útil para se trabalhar.

| | | |
|---|---|---|
| **1.** | $(\forall x)[P(x) \vee Q(x)]$ | (hipótese) |
| **2.** | $(\forall x)([P(x)]' \rightarrow Q(x))$ | (substituição de $A \vee B \leftrightarrow A' \rightarrow B$ em 1) |

A tese consiste em wffs separadas para $P(x)$ e $Q(x)$, cada qual com seus próprios quantificadores; o Axioma 4 nos permite quebrar a linha 2 em duas wffs diferentes.

| | | |
|---|---|---|
| **3.** | $(\forall x)[P(x)]' \rightarrow (\forall x)Q(x)$ | (2, Axioma 4, modus ponens) |

O lado esquerdo do passo 3 sugere que ele pode ser útil para usar o Axioma 8 em seguida.

| | | |
|---|---|---|
| **4.** | $(\forall x)[P(x)]' \leftrightarrow [(\exists x)P(x)]'$ | (Axioma 8) |
| **5.** | $[(\exists x)P(x)]' \rightarrow (\forall x)Q(x)$ | (substituição de 4 em 3) |

O passo 5 está intimamente relacionado ao que desejamos. A seqüência da demonstração é:

| | | |
|---|---|---|
| **1.** | $(\forall x)[P(x) \vee Q(x)]$ | (hipótese) |
| **2.** | $(\forall x)([P(x)]' \rightarrow Q(x))$ | (substituição de $A \vee B \leftrightarrow A' \rightarrow B$ em 1) |
| **3.** | $(\forall x)[P(x)]' \rightarrow (\forall x)Q(x)$ | (2, Axioma 4, modus ponens) |
| **4.** | $(\forall x)[P(x)]' \leftrightarrow [(\exists x)P(x)]'$ | (Axioma 8) |
| **5.** | $[(\exists x)P(x)]' \rightarrow (\forall x)Q(x)$ | (Substituição de 4 em 3) |
| **6.** | $(\exists x)P(x) \vee (\forall x)Q(x)$ | (5, tautologia $(A' \rightarrow B) \rightarrow A \vee B$, modus ponens) • |

O método de dedução nos permite incluir hipóteses "temporárias" ao longo da demonstração, como podemos ver no próximo exemplo.

**EXEMPLO 22** A wff

$$[P(x) \rightarrow (\forall y)Q(x, y)] \rightarrow (\forall y)[P(x) \rightarrow Q(x, y)]$$

é um teorema. Na demonstração a seguir, $P(x)$ é introduzida no passo 2 como uma hipótese temporária que nos permite deduzir $Q(x, y)$ no passo 4. O passo 5 apenas atesta a dependência de $Q(x,$ y) da hipótese temporária; e, naturalmente, toda a wff do passo 5, $P(x) \rightarrow Q(x, y)$ é dependente da hipótese do passo 1. Como $y$ não

é uma variável livre no passo 1, a condição (a) da regra de generalização não é violada no passo 6, e a condição (b) não se aplica.

| | | |
|---|---|---|
| **1.** | $P(x) \rightarrow (\forall y)Q(x, y)$ | (hipótese) |
| **2.** | $P(x)$ | (hipótese temporária) |
| **3.** | $(\forall y)Q(x, y)$ | (1,2, modus ponens) |
| **4.** | $Q(x, y)$ | (3, Axioma 5, modus ponens) |
| **5.** | $P(x) \rightarrow Q(x, y)$ | (4 deduzido do 2) |
| **6.** | $(\forall y)[P(x) \rightarrow Q(x, y)]$ | (5, generalização) |

•

PRÁTICA 19 Usando a lógica de predicados, prove o teorema

$$(\forall y)[P(x) \rightarrow Q(x, y)] \rightarrow [P(x) \rightarrow (\forall y)Q(x, y)]$$

•

O Exemplo 22 e a Prática 19 mostraram que a wff

$$(\forall y)[P(x) \rightarrow Q(x, y)] \leftrightarrow [P(x) \rightarrow (\forall y)Q(x, y)]$$

é válida. Isto significa que o quantificador universal pode ser "negligenciado" para as subwffs que não contenham a variável quantificada. Vale ainda um resultado semelhante para o quantificador existencial. Como um resultado particular, existem duas ou mais formas de se representar sentenças da língua portuguesa como wffs predicativas, como nos Exercícios 7 a 9 da Seção 1.2.

## Argumentos Válidos

Para provar a validade de um argumento que contenha sentenças quantificadas, procedemos quase como antes. Convertemos o argumento em forma simbólica, e mostramos que a tese pode ser obtida da hipótese. No entanto, desta vez, trabalharemos com a lógica de predicados, ao invés da lógica proposicional. Um argumento $P_1 \wedge P_2 \wedge ... \wedge P_n \rightarrow Q$ é válido quando for um teorema, que também o torna uma wff válida.

EXEMPLO 23 Mostre que o argumento a seguir é válido: "Todo microcomputador tem uma porta serial. Alguns microcomputadores têm porta paralela. Portanto alguns computadores têm ambas as portas serial e paralela". Usando

*M(x): x* é um microcomputador.
*S(x): x* tem porta serial.
*P(x): x* tem porta paralela.

o argumento é

$$(\forall x)[M(x) \rightarrow S(x)] \wedge (\exists x)[M(x) \wedge P(x)] \rightarrow (\exists x)[M(x) \wedge S(x) \wedge P(x)]$$

Perceba que se tentarmos simbolizar este argumento na lógica proposicional, obteremos $A \wedge B \rightarrow C$, que não é um argumento válido. A lógica proposicional é, simplesmente, não expressiva o suficiente para conter as relações entre as partes deste argumento que o tornam válido.

Uma demonstração é

| | | |
|---|---|---|
| **1.** | $(\forall x)[M(x) \rightarrow S(x)]$ | (hipótese) |
| **2.** | $(\exists x)[M(x) \wedge P(x)]$ | (hipótese) |
| **3.** | $M(a) \wedge P(a)$ | (2, Axioma 6, modus ponens) |
| **4.** | $M(a) \rightarrow S(a)$ | (1, Axioma 5, modus ponens) |
| **5.** | $M(a)$ | (3, tautologia $A \wedge B \rightarrow B$, modus ponens) |
| **6.** | $S(a)$ | (4, 5, modus ponens) |
| **7.** | $M(a) \wedge P(a) \wedge S(a)$ | (3, 6, $A \wedge B$ pode ser deduzida de $A$ e $B$) |
| **8.** | $M(a) \wedge S(a) \wedge P(a)$ | (substituição de $A \wedge B \leftrightarrow B \wedge A$ em 7) |
| **9.** | $(\exists x)[M(x) \wedge S(x) \wedge P(x)]$ | (8, Axioma 7, modus ponens) |

Mais uma vez, é *a. forma* do argumento que vale e não seu conteúdo. Perceba que na demonstração usamos o Axioma 6 no passo 3 para inserir a nova constante *a,* e então usamos o Axioma 5 no passo 4 com a mesma constante. Estes dois passos não podem ser trocados pois o Axioma 6 pode introduzir novos nomes apenas se eles ainda não tiverem sido usados na demonstração. Eis por que o Axioma 6 deve ser usado o mais cedo possível na demonstração. •

PRÁTICA 20 Mostre que o argumento a seguir é válido: "Todas as músicas de rock são barulhentas. Existem algumas músicas de rock, logo existem algumas músicas barulhentas." Use os predicados *R(x)* e *B(x).* •

## Revisão da Seção 1.4

### Técnicas

- Demonstração de teoremas na lógica de predicados
- Uso da lógica de predicados para provar a validade de um argumento na língua portuguesa

**Idéia Principal**

Todo teorema da lógica de predicados é uma wff válida, e toda wff válida é um teorema.

## Exercícios 1.4

1. Justifique cada um dos passos na demonstração a seguir

$$(\exists x)[P(x) \rightarrow Q(x)] \rightarrow [(\forall x)P(x) \rightarrow (\exists x)Q(x)]$$

1. $(\exists x)[P(x) \rightarrow Q(x)]$
2. $P(a) \rightarrow Q(a)]$
3. $(\forall x)P(x)$
4. $P(a)$
5. $Q(a)$
6. $(\exists x)Q(x)$

★2. Considere a wff

$$(\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)[P(x) \wedge Q(x)]$$

a. Encontre uma interpretação que demonstre que esta wff não é válida.
b. Encontre falha na seguinte "demonstração" desta wff.

| | | |
|---|---|---|
| 1. | $(\exists x)P(x)$ | (hipótese) |
| 2. | $P(a)$ | (1, Axioma 6, modus ponens) |
| 3. | $(\exists x)Q(x)$ | (hipótese) |
| 4. | $Q(a)$ | (3, Axioma 6, modus ponens) |
| 5. | $P(a) \wedge Q(a)$ | ($A \wedge B$ pode ser deduzida de $A$ e $B$) |
| 6. | $(\exists x)[P(x) \wedge Q(x)]$ | (5, Axioma 7, modus ponens) |

3. Considere a wff $(\forall y)(\exists x)Q(x, y) \rightarrow (\exists x)(\forall y)Q(x, y)$.
a. Encontre uma interpretação que demonstre que esta wff não é válida.
b. Encontre a falha na seguinte "demonstração" desta wff.

| | | |
|---|---|---|
| 1. | $(\forall y)(\exists x)Q(x, y)$ | (hipótese) |
| 2. | $(\exists x)Q(x, y)$ | (1, Axioma 5, modus ponens) |
| 3. | $Q(a, y)$ | (2, Axioma 6, modus ponens) |
| 4. | $(\forall y)Q(a, y)$ | (3, generalização) |
| 5. | $(\exists x)(\forall y)Q(x, y)$ | (4, Axioma 7, modus ponens) |

Nos Exercícios 4 a 7 demonstre que cada wff é um teorema da lógica de predicados.

★**4.** $(\forall x)P(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$

**5.** $(\forall x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)[P(x) \wedge Q(x)]$

**6.** $(\exists x)(\exists y)P(x, y) \rightarrow (\exists y)(\exists x)P(x, y)$

**7.** $(\forall x)(\forall y)Q(x, y) \rightarrow (\forall y)(\forall x)Q(x, y)$

Nos Exercícios 8 a 15, prove que as wffs são teoremas da lógica de predicados ou apresente uma interpretação para provar que não são válidas.

★**8.** $(\exists x)[A(x) \wedge B(x)] \rightarrow (\exists x)A(x) \wedge (\exists x)B(x)$

**9.** $(\exists x)[R(x) \vee S(x)] \rightarrow (\exists x)R(x) \vee (\exists x)S(x)$

**10.** $[(\forall x)P(x) \rightarrow (\forall x)Q(x)] \rightarrow (\forall x)[P(x) \rightarrow Q(x)]$

★**11.** $(\exists x)(\forall y)Q(x, y) \rightarrow (\forall y)(\exists x)Q(x, y)$

**12.** $(\forall x)P(x) \vee (\exists x)Q(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$

**13.** $(\forall x)[A(x) \rightarrow B(x)] \rightarrow [(\exists x)A(x) \rightarrow (\exists x)B(x)]$

**14.** $(\forall y)[Q(x, y) \rightarrow P(x)] \rightarrow [(\exists y)Q(x, y) \rightarrow P(x)]$

★**15.** $[P(x) \rightarrow (\exists y)Q(x, y)] \rightarrow (\exists y)[P(x) \rightarrow Q(x, y)]$

Usando a lógica de predicados, prove que os argumentos dos Exercícios 16 a 20 são válidos. Use os símbolos predicados mostrados.

16. Há um astrônomo que não é míope. Todo mundo que usa óculos é míope. Portanto, todo mundo ou usa óculos ou usa lentes de contato. Portanto, algum astrônomo usa lentes de contato. *(A(x), M(x), O(x), L(x))*

★17. Todo membro da mesa vem da indústria ou do governo. Todos do governo que são advogados são a favor da moção. John não é da indústria, mas ele não é advogado. Portanto, se John for um membro da mesa, ele é favor da moção. *(M(x), I(x), G(x), A(x), F(x),j)*

18. Existem algumas estrelas de cinema que são mais ricas que as outras. Todo mundo que é mais rico que os outros também paga mais impostos que os outros. Portanto, existe uma estrela de cinema que paga mais impostos que os outros. *(E(x), R(x, y), I(x, y))*

19. Todo estudante da Ciência da Computação trabalha mais que alguém e todo mundo que trabalha mais que alguém também dorme menos que esta pessoa. Maria é uma estudante da Ciência da Computação. Portanto Maria dorme menos que outra pessoa. *(E(x), T(x, y), D(x, y), m)*

20. Todo embaixador fala apenas com diplomatas e algum embaixador fala com alguém, portanto existe um diplomata. *(E(x), F(x*, y), *D(x))*

21. Prove que

$$[(\forall x)P(x)]' \leftrightarrow (\exists x)[P(x)]'$$

é válida. *{Dica:* ao invés de uma seqüência de demonstração, use o Axioma 8 e substitua as expressões equivalentes.)

22. A equivalência do Exercício 21 diz que se for falso que todo elemento do domínio tem a propriedade *P*, então algum elemento do domínio também deixa de ter a propriedade *P* e vice-versa. O elemento que não tem a propriedade *P* é chamado de **contra-exemplo** da sentença que todo elemento tem a propriedade *P*. Portanto, um contra-exemplo da sentença

$$(\forall x)(x \text{ é ímpar})$$

no domínio dos inteiros é o número 10, um inteiro par. (Naturalmente, existem diversos outros contra-exemplos para esta sentença). Encontre contra-exemplos no domínio dos inteiros para as seguintes sentenças. (Um inteiro $x > 1$ é **primo** se seus únicos divisores forem 1 e $x$ .)

**a.** $(\forall x)(x \text{ é negativo})$
**b.** $(\forall x)(x \text{ é a soma dos inteiros pares})$
**c.** $(\forall x)(x \text{ é primo} \rightarrow x \text{ é ímpar})$
**d.** $(\forall x)(x \text{ é primo} \rightarrow (-1)^x = -1)$
**e.** $(\forall x)(x \text{ é primo} \rightarrow 2^x - 1 \text{ é primo})$

# Seção 1.5 Programação Lógica e Prova de Correção

A lógica de predicados, outrora objeto de interesse apenas de matemáticos e filósofos, tem diversas aplicações de importância na Ciência da Computação. Duas delas serão discutidas nesta seção.

## Programação Lógica

Na lógica de predicados, usamos regras de inferência para chegarmos a teses a partir das hipóteses. Se uma tese tiver sido demonstrada como conseqüência de determinada hipótese, então, em uma interpretação na qual a hipótese seja verdadeira, a tese também será verdadeira. A linguagem de programação Prolog, que significa *progamming* in *logic*, também ajuda a chegar a teses a partir das hipóteses. A linguagem inclui predicados, conectivos lógicos e regras de inferência. Ela permite a descrição de uma interpretação, ou melhor, de hipóteses verdadeiras em uma interpretação.

As linguagens de programação com as quais você provavelmente já tem familiaridade, tal como Pascal, são conhecidas como **linguagens procedurais.** A maior parte dos programas escritos em linguagens procedurais destinam-se a resolver o problema à mão. O programador, portanto, diz ao computador como resolver o problema. Prolog, no entanto, é uma **linguagem declarativa** (também chamada de **linguagem descritiva).** Um programa Prolog consiste em declarações ou descrições sobre uma interpretação, isto é, quais as hipóteses que são verdadeiras em uma interpretação. O conjunto de declarações é também chamado de base de dados do Prolog. Para determinar se uma dada tese, posta na forma de uma pergunta pelo usuário, é ou não verdadeira para a interpretação, Prolog usa sua base de dados e aplica suas regras de inferências (sem a necessidade de qualquer instrução por parte do programador).

Itens em uma base de dados do Prolog podem ter duas formas, conhecidas em Prolog como *fatos* e *regras.* (Porém as regras do Prolog são apenas outro tipo de fatos, e não devem ser confundidas com as regras de inferência.)

Os **fatos do Prolog** permitem definir predicados. Por exemplo, suponhamos que desejemos criar um programa Prolog que descreva as cadeias alimentares em uma determinada região ecológica. Devemos começar com um predicado binário *come.* Então descreveremos este predicado fornecendo os pares de elementos no domínio que tornam *come* verdadeiro. Portanto, teríamos os fatos

*come*(urso, peixe)
*come*(urso, raposa)
*come*(veado, mato)

em nossa base de dados. (Os detalhes exatos dos comandos Prolog variam de implementação para implementação, portanto daremos aqui apenas o espírito da linguagem através do uso de um pseudocódigo semelhante ao Prolog.) Neste exemplo, "urso", "peixe", "raposa", "veado" e "mato" são constantes porque representam elementos específicos do domínio. Como o domínio propriamente dito não é especificado, exceto na declaração dos predicados, neste ponto podemos fazer inferir que o domínio consiste em "urso", "peixe", "raposa", "veado" e "mato". E saudável que o usuário mantenha um entendimento e faça um uso consistente dos predicados em um programa Prolog. Portanto,

*come(urso,* peixe)

pode ser usado tanto para representar o fato de que ursos comem peixes ou de que peixes comem ursos! Arbitramos a convenção de que *come(x,* y) significa *"x* come *y'.*

Podemos incluir descrições de dois predicados unários, *animal* e *planta* para a base de dados incluindo os fatos

*animal(urso)*
*animal*(peixe)
*animal*(raposa)
*animal(veado)*
*planta(mato)*

De posse deste programa Prolog (base de dados), podemos fazer algumas perguntas simples.

EXEMPLO 24 A pergunta

is*(animal(*urso))

simplesmente pergunta se o fato *animal(urso)* está na base de dados. Como este fato está na base de dados, o Prolog responderá à pergunta com yes. Outros diálogos com o Prolog poderiam incluir

*is(come(veado,mato))*

yes

is*(come*(urso, coelho))

no •

Perguntas podem incluir variáveis, como mostrado no próximo exemplo.

EXEMPLO 25 A pergunta

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

produz

peixe
raposa

como resposta. O Prolog respondeu à pergunta procurando em sua base de dados por todos os fatos que se ajustassem ao padrão *come*(urso, x), onde *x* é uma variável. A resposta "peixe" é dada antes porque as regras são percorridas da primeira para a última. •

As perguntas podem conter os conectivos lógicos **and, or** e **not.**

PRÁTICA 21 Dada a base de dados

*come*(urso, peixe)
*come*(urso, raposa)
*come(*veado, mato)
*animal(urso)*
*animal (peixe)*
*animal(raposa)*
*animal(veado)*
*planta(mato)*

qual será a resposta do Prolog para a pergunta

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

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

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

Isto indica que *x* é uma presa se for animal que é comido. Se incluirmos esta regra a nossa base de dados, então a resposta à pergunta

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

teremos a resposta

peixe
raposa

### Classes de Horn e Resolução

Como os fatos e as regras do Prolog se relacionam com o formalismo da lógica de predicados? Podemos descrever os fatos em nossa base de dados pelas wffs

*C(u, p)*
*C(u, r)*
C(v, *m)*
*A(u)*
*A(p)*
*A(r)*
*A(v)*
*P(m)*

e a regra pela wff

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

Quantificadores universais são partes explícitas da regra que aparecem em um programa Prolog, mas o Prolog trata a regra como sendo quantificada universalmente,

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

e usa repetidamente nosso Axioma 5 da lógica de predicados para tirar o quantificador universal e permitir que as variáveis assumam, a seu tempo, cada valor no domínio.

Tanto os fatos quanto as regras são exemplos de **Cláusulas de Horn.** Uma cláusula de Horn é uma wff composta por predicados ou as negações dos predicados (com variáveis ou constantes como argumentos) unidos por disjunções, onde no máximo um predicado não é negado. Portanto o fato

*C(d, g)*

é um exemplo de cláusula de Horn porque contém um único predicado não-negado. A wff

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

é um exemplo de uma cláusula de Horn porque consiste em três predicados unidos pela disjunção onde *Pr(x)* não está negado. Pela lei de De Morgan, ele é equivalente a

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

que, por sua vez, e equivalente a

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

e, portanto, representa uma regra em nosso programa Prolog.

A regra de inferência usada pelo Prolog é chamada **resolução.** Duas cláusulas de Horn em uma base 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 cláusula. A nova cláusula elimina o termo de correspondência e fica, então, disponível para uso em respostas às perguntas. Por exemplo,

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

é resolvida para *B(b).* Isto mostra que o Prolog considera

$$(A(a) \wedge [A(a) \rightarrow B(b)]) \rightarrow B(b)$$

como um teorema, o que é uma simples aplicação do modus ponens. Portanto a regra de inferência do Prolog inclui o modus ponens como um caso especial.

Na aplicação da regra de resolução, as variáveis são consideradas como "correspondentes" a qualquer símbolo constante. (Isto é uma aplicação repetida do Axioma 5.) Em qualquer nova cláusula resultante, as variáveis são substituídas por suas constantes associadas de uma maneira consistente. Portanto, em reposta à pergunta "qual *x* é uma presa", o Prolog procura, na base de dados, por uma regra com o predicado desejado *Pr(x)* como o conseqüente. Ele encontra

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

Ele então prossegue pela base de dados na busca de outras cláusulas que possam ser resolvidas com esta cláusula. A primeira dessas cláusulas é o fato *E(b,fí).* Estas duas cláusulas se resolvem em

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

(Perceba que a constante *p* substituiu *x* em todos os lugares.) Usando esta nova cláusula, ela poderá ser resolvida com o fato *A(p)* para concluir *Pr(p).* Tendo alcançado todas as conclusões possíveis da resolução do fato *C(u, p),* Prolog refaz o processo para procurar por outra cláusula a resolver com a aplicação da cláusula da regra; desta vez ele encontrará *C(u, r).*

Como um exemplo mais complexo de resolução, suponhamos que incluímos a regra

*caçado(x)* **if** *presa(x)*

à base de dados. Esta regra, na forma simbólica, é

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

Que é resolvida com a regra de definição de *presa,*

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

para chegar à nova regra

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

A pergunta

***which****(x:caçado(x))*

usará esta nova regra para concluir

peixe
raposa

**EXEMPLO 26** Suponha que um programa Prolog contenha as seguintes entradas:

*come(urso,* peixe)
*come*(peixe, peixinho)
*come(*peixinho, alga)
*come(*quati, peixe)
*come(urso,* quati)
*come(urso,* raposa)
*come*(raposa, coelho)
*come*(coelho, mato)
*come*(urso, veado)
*come(*veado, mato)
*come*(gato-selvagem, veado)

*animal(urso)*
*animal(peixe)*
*animal*(peixinho)
*animal(quati)*
*animal(raposa)*
*animal(coelho)*
*animal(veado)*
*animal*(gato-selvagem)

*planta*(mato)
*planta*(alga)

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

Então realiza-se o seguinte diálogo com o Prolog:

is*(animal(*coelho))

yes

is(*come*(gato-selvagem, mato))

no

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

urso
quati

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

peixinho alga
coelho mato
veado mato

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

peixe
peixinho
peixe
quati
raposa
coelho
veado
veado

Perceba que peixe é listado duas vezes como satisfazendo a última pergunta por que o peixe é comido pelo urso (fato 1) e pelo quati (fato 3). Analogamente, o veado é comido pelo urso e pelo gato-selvagem. •

PRÁTICA 22

a. Formule uma regra Prolog que defina o predicado *predador*.
b. Incluindo esta regra à base de dados do Exemplo 26, qual seria a resposta à pergunta

**which***(x:predador(x))* •

## Recursão

As regras do Prolog são implicações. Seus antecedentes podem depender de fatos, como em

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

ou em outras regras como em

*caçado(x)* if *presa(x)*

O antecedente de uma regra pode também depender da mesma regra, caso no qual a regra é definida em função dela mesma. Uma definição em que o item sendo definido é, ele próprio, parte da definição é chamada de definição recursiva.

Como um exemplo, vamos supor que desejamos usar a base de dados ecológica do Exemplo 26 para estudar a cadeia alimentar. Podemos definir uma relação binária *na-cadeia-alimentar(x, y)* que significa "y está na cadeia alimentar de x". Isto, por sua vez, pode significar duas coisas:

1. *x* come *y* diretamente

ou

2. *x* come algum animal que come algum animal que come algum animal ... que come *y*.

O caso 2 pode ser reescrito como:

*2'. x* come *z* e *y* está na cadeia alimentar de *z.*

O caso 1 simplesmente verifica fatos existentes, mas sem (2'), *na-cadeia-alimentar* não significa nada além de *come.* Por outro lado, *(2')* sem (1) nos coloca na busca infinita de algum animal que coma algum animal que coma algum animal... sem nos dizer quando parar. Definições recursivas sempre precisam de um ponto de parada que consista em informações específicas.

A regra Prolog para *na-cadeia-alimentar* incorpora (1) e (*2')* é:

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

Esta é uma **regra recursiva,** pois define o predicado de *na-cadeia-alimentar* em termos de *na-cadeia-alimentar.*

**EXEMPLO 27** Após a inclusão da regra *na-cadeia-alimentar* à base de dados do Exemplo 26, a seguinte pergunta é feita:

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

A resposta é a seguinte (os números foram usados para fins de referência):

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

O Prolog simplesmente aplica o caso de

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

primeiro, obtendo as respostas 1 a 4 diretamente dos fatos *come(urso,* peixe), *come*(urso, quati) e assim por diante. Passemos ao caso recursivo

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

uma correspondência de *come*(urso, *z)* ocorre com *z* igual a "peixe". O Prolog então procura todas as soluções para relação *na-cadeia-alimentar(peixe, y).* Usando primeiro o caso simples de *na-cadeia-alimentar,* uma correspondência ocorre com o fato *come*(peixe, peixinho). Este gera a resposta 5, peixinho. Não há outros fatos da forma *come(*peixe, *y),* portanto a próxima tentativa é o caso recursivo:

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

Uma correspondência para *come*(peixe, z) ocorre com *z* igual a "peixinho". O Prolog então procura por todas as soluções para a relação *na-cadeia-alimentar(peixinho, y).* Usando o caso simples de *na-cadeia-alimentar,* uma correspondência pode ocorrer com o fato *come*(peixinho, alga). Isto resulta na resposta 6, alga. Como não há outros fatos na forma *come*(peixinho, *y),* a próxima coisa a fazer é o caso recursivo

*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". O Prolog então procura por todas as soluções para a relação *na-cadeia-alimentar(alga, y).* Uma busca em toda base de dados não revela fatos da forma *come(alga, y)* (ou *come*(alga, *z)),* portanto nem o caso simples nem o caso recursivo de *na-cadeia-alimentar(alga, y)* pode ser seguido deste ponto em diante.

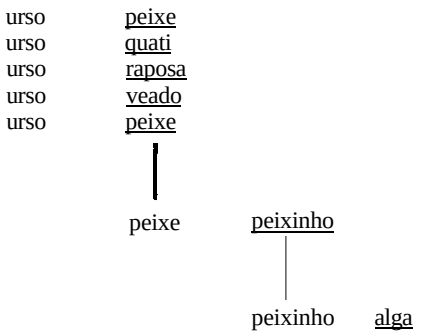



**Figura 1.2**

A Fig. 1.2 mostra a situação neste ponto. O Prolog encontrou um "beco sem saída" com *na-cadeia-alimentar(alga,* y) e fará um retrocesso no caminho de busca. Como não há outros fatos da forma *come*(peixinho, *z),* a busca por soluções de *na-cadeia-alimentar(*peixinho, *y)* termina. Então, por não haver mais outros fatos na forma *come*(peixe, z), a busca por soluções de *na-cadeia-alimentar(peixe,y* ) também termina. Voltando no caminho de busca, existe outra correspondência para *come*(urso, z) com *z* igual a "quati" que gera outro caminho de busca. •

No Exemplo 27, uma vez que o Prolog tenha começado a investigar *na-cadeia-alimentar(peixe,* y), todas as respostas a perguntas que possam ser obtidas pela exploração deste caminho (respostas 5 e 6) serão geradas antes das demais (respostas 7 a 12). A busca que visa priorizar a exploração até pontos distantes dos caminhos e depois voltar por eles, para então percorrer outros caminhos, é chamada de estratégia de **busca em profundidade.**

PRÁTICA 23 Faça o acompanhamento da execução do programa Prolog do Exemplo 27 e explique por que ocorrem as respostas 7 a 12. •

Uma regra recursiva é necessária quando o predicado sendo descrito é passado de um objeto para o outro. O predicado *na-cadeia-alimentar* tem esta propriedade:

*na-cadeia-alimentar(x, y)* A *na-cadeia-alimentar(y, z)* $\rightarrow$ *na-cadeia-alimentar(x, z)*

### Sistemas Especialistas

Muitas aplicações interessantes vêm sendo desenvolvidas, em Prolog e linguagens semelhantes para programação lógica, que reúnem uma base de dados de fatos e regras, e então a usam para gerar conclusões. Tais programas são conhecidos como **sistemas especialistas, sistemas baseados no conhecimento** ou **sistemas baseados em regras.** A base de dados em um sistema especialista tenta retratar o conhecimento ("acumulado com a experiência") de um especialista humano em alguma área particular do conhecimento, incluindo os fatos sabidos pelo sujeito e seus métodos de obtenção de conclusões destes fatos. O sistema especialista não apenas simula a ação de um humano, mas pode ser questionado a fim de indicar por que tomou certas decisões ao invés de outras.

Alguns sistemas especialistas vêm sendo construídos a fim de simular diagnósticos de médicos especialistas a partir de sintomas de pacientes, decisões de um gerente de fábrica sobre o controle de válvulas em uma fábrica química baseado nas leituras dos sensores, decisões de um comprador de roupas para uma butique baseado em pesquisas de mercado, a escolha feita por um consultor ao fazer a especificação da configuração de um sistema de computadores baseado nas necessidades do consumidor, e diversas outras. O desafio de fazer os sistemas especialistas reside em extrair todos os fatos pertinentes e regras do especialista humano.

## Prova de Correção

As páginas anteriores mostraram como a lógica de predicados pode ser usada como base para toda uma linguagem de programação. A lógica de predicados é também útil na verificação formal da correção de programas escritos em linguagens de programação mais tradicionais (procedurais).

A **verificação do programa** tenta garantir que um programa de computador está correto. "Correção" tem uma definição mais limitada do que tem no uso do dia-a-dia. Um programa está **correto** se o seu compor-

tamento está de acordo com as especificações. No entanto, isto não necessariamente indica que o programa resolve o problema para o qual foi projetado para resolver; as especificações do programa podem não estar de acordo com ou não prever todos os aspectos das necessidades do cliente. **Validação do programa,** que não discutiremos, tenta garantir que o programa atinja as necessidades originais do cliente. Em um projeto de desenvolvimento de um grande programa, "program V & V" foi considerado tão importante que um grupo de pessoas separadas dos programadores pode ser designado para realizar esta tarefa de validação.

A verificação de programas pode ser abordada informalmente, através de testes do programa ou formalmente, através da *prova de correção.* Os **testes de programa** buscam mostrar que, para valores de entrada particulares produzem valores de saída aceitáveis. A tarefa de testar programas é checar a performance do programa em um largo e representativo conjunto de valores de entrada; em geral, testar para todos os valores de entrada não é possível. A **prova de correção** usa as técnicas da lógica formal para provar que, dada quaisquer variáveis de entradas que satisfaçam determinados predicados ou propriedades, as variáveis de saída produzidas pela execução do programa satisfazem outras propriedades especificadas.

Para distinguir entre a prova de correção e o teste de um programa, considere o seguinte programa para calcular o tamanho *c* da hipotenusa de um triângulo retângulo, dados os valores positivos *a* e *b* de seus catetos. Para provar que o programa está correto devemos estabelecer que sempre que *a* e *b* satisfazem os predicados $a > 0$ e $b > 0$, então após a execução do programa, o predicado $a^2 + b^2 = c^2$ é satisfeito. Testar este programa exigiria que tomássemos diversos valores específicos para *a e b,* computássemos o valor de *c* e verificássemos se $a^2 + b^2$ é igual a $c^2$ para cada caso.

Descrevendo a prova de correção mais formalmente, denotemos por *X* um conjunto arbitrário de valores de entrada de algum programa ou trecho de programa *P.* O conjunto correspondente de valores-saída *Y* é produzido a partir de *X* por quaisquer transformações que *P* realize nos dados. Denotaremos essas transformações por $Y = P(X)$. Um predicado *Q* descreve as condições que os valores de entrada devem satisfazer. Por exemplo, se um programa se destina a achar a raiz quadrada de um número positivo, temos um valor de entrada, *x,* e $Q(x)$ deve ser $"x > 0"$. Um predicado *R* que descreve as condições que os valores de saída devem satisfazer. Essas condições sempre envolverão também os valores de entrada; portanto, em nosso caso da raiz quadrada, se *y* for o único valor de saída, então desejamos que $R(x, y)$ seja $"y^2 = x"$. O programa *P* será correto se

$$(\forall x)(Q(X) \rightarrow R[X, P(X)]) \qquad \textbf{(1)}$$

Vamos abreviar (1) por

$$\{Q\}P\{R\} \qquad (2)$$

Esta notação sugere que *Q* e *R* são comentários de programa não-executáveis, mas tenhamos em mente que (2) *representa uma implicação* que precisa ser válida para todo *X.* A condição *Q* é chamada de **pré-condição** para o programa *P,* a condição *R é a* **pós-condição.**

Ao invés de simplesmente termos um predicado inicial e um predicado final, um programa ou trecho de programa é dividido em comandos individuais $s_i$ com os predicados inseridos antes e depois. Esses predicados são chamados de **asserção** por fazerem exigências sobre o que deve ser verdadeiro a respeito das variáveis de programas nos diversos pontos do mesmo. Portanto, há uma série de asserções, $Q, R_1, R_2, \ldots, R_n$. *P* está provavelmente correto se as seguintes implicações forem válidas:

$$\begin{aligned}
&\{Q\}s_0\{R_1\}\\
&\{R_1\}s_1\{R_2\}\\
&\{R_2\}s_2\{R_3\}\\
&\quad\vdots\\
&\{R_{n-1}\}s_{n-1}\{R\}
\end{aligned}$$

As asserções intermediárias são normalmente obtidas trabalhando-se da asserção de saída *R* para trás. Quando as asserções intermediárias tiverem sido decididas, as implicações podem ser provadas usando-se um sistema de lógica formal de axiomas e regras de inferência.

## O Axioma da Atribuição

Suponha que o comando $s_i$ seja um comando de atribuição da forma $x := e$, isto é, a variável *x* recebe o valor de *e,* onde *e* é alguma expressão. É aceito como axioma que a implicação

$$\{R_i\}s_i\{R_{i+1}\}$$

é válida se $R_i$ for o predicado $R_{i+1}$ com *e* substituído em todos os pontos de *x.* Em forma abreviada, o **axioma da atribuição** é

**Axioma da Atribuição**

$$\{R_e^x\}x := e\{R\}$$

onde $\{R_e^x\}$ significa "substitui $e$ por $x$ em $R$".

EXEMPLO 28 Se o comando do programa é $x := x - 1$ e a asserção após este comando é $x > 0$, então a asserção antes deste comando deve ser $x - 1 > 0$. Escrevendo a pré-condição e a pós-condição como comentários de programa, o trecho de programa

```
{x - 1 > 0}
x := x - 1
{x > 0}
```

está correto por causa do axioma da atribuição.

Isto realmente funciona? Para todo $x$, se $x - 1 > 0$ antes do comando ser executado (perceba que isto significa que $x > 1$), então após o comando o valor de $x$ é reduzido a 1, neste caso $x > 0$.

PRÁTICA 24 De acordo com o axioma da atribuição, qual a pré-condição do seguinte trecho de programa? Explique por que funciona. •

```
{pré-condição}
x := x - 2;
{x = y}
```
•

EXEMPLO 29 Verifique a correção do seguinte trecho de programa para trocar os valores de $x$ e $y$:

```
temp := x;
x := y;
y := temp;
```

No início deste trecho de programa, $x$ e $y$ têm certos valores. Portanto, devemos expressar a pré-condição como $x = a$ e $y = b$. A pós-condição desejada é, então, $x = b$ e $y = a$. Usando o axioma da atribuição, podemos trabalhar de trás para frente, da pós-condição para a pré-condição para encontrar as asserções anteriores:

```
{y = b, x = a}
temp := x;
{y = b, temp = a}
x := y;
{x = b, temp = a}
y := temp;
{x = b, y = a}
```

A primeira asserção está de acordo com a pré-condição; o axioma da atribuição, aplicado repetidamente, assegura, então, que o trecho de programa está correto. •

PRÁTICA 25 Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas:

```
{x = 3}
y := 4;
z := x + y;
{z = 7}
```
•

Algumas vezes a pré-condição de um trecho de programa é trivialmente verdadeira, como mostrado no próximo exemplo.

EXEMPLO 30 Verifique a correção do seguinte trecho de programa para computar $y = x - 4$.

```
y := x;
y := y - 4;
```

Neste caso a pós-condição desejada é $y = x - 4$. Usando o axioma do fim para o início, a partir da pós-condição, obtemos

```
{x − 4 = x − 4}
 y := x;
{y − 4 = x − 4}
 y := y − 4;
{y = x − 4}
```

A pré-condição é sempre verdadeira; por isso, pelo axioma da atribuição, cada asserção seguinte, incluindo a pós-condição, é verdadeira. •

### A Regra Condicional

**Um comando condicional** é um comando de programa da forma

```
if condição B then
  P1
else
  P2
```

Quando este comando é executado, uma condição *B,* que é ou verdadeira ou falsa, é avaliada. Se *B* for verdadeira, o trecho $P_1$ do programa é executado, mas se *B* for falsa, o trecho $P_2$ do programa é que é executado.

Lembremos que determinar a correção de qualquer comando $s_i$, do programa envolve provar que a implicação

$$\{Q\}\, s_i\, \{R\}$$

*é* verdadeira, onde *Q e R* são a pré-condição e a pós-condição, respectivamente, para o comando. Se $s_i$ é um comando condicional, então uma *regra condicional de inferêneia* pode ser usada para provar esta implicação.

**Regra Condicional de Inferência**
A implicação

$$\{Q\}s_i\{R\}$$

onde $s_i$ *é o* comando condicional

**if** condição *B* **then**
  $P_1$
**else**
  $P_2$

pode ser inferida de

$$\{Q \wedge B\}P_1\{R\}$$

e $\{Q \wedge B'\}P_2\{R\}$

**EXEMPLO 31** Verifique a correção do trecho de programa abaixo com a pré-condição e pós-condição mostradas:

```
{n = 5}
   if n >=10 then
     y: = 100
   else
     y := n + 1;
{y = 6}
```

Aqui a pré-condição é $n = 5$, e a condição *B* a ser avaliada é n > = 10. A fim de aplicar a regra condicional, devemos provar que

$$\{n = 5 \text{ e } n \geq 10\}\ y := 100\ \{y = 6\}$$

é válida. Esta implicação é verdadeira porque seu antecedente, $n = 5$ e $n \geq 10$, é falso. Precisamos ainda mostrar que

$\{n = 5 \text{ e } n < 10\}\ y := n + 1\ \{y = 6\}$

é válida. Trabalhando de trás para frente, temos

$\{n + 1 = 6 \text{ ou } n = 5\}$
$y := n + 1;$

Portanto

$\{n = 5\} y := n + 1 \{y = 6\}$

é verdadeira pelo axioma da atribuição e portanto

$\{n = 5 \text{ e } n < 10\}\ y := n + 1\ \{y = 6\}$

é verdadeira. A regra condicional nos permite concluir que o trecho do programa está correto. •

PRÁTICA 26 Verifique a correção do trecho de programa abaixo com a pré-condição e a pós-condição mostradas:

```
{x = 4}
   if x < 5 then
     y := x - 1
   else
     y:=7;
{y = 3}
```
•

EXEMPLO 32 Verifique a correção do seguinte trecho de programa para computar *max(x, y)*, o maior dentre dois valores x e *y*.

```
if x >=y then
  max := x
else
  max := y;
```

A pós-condição desejada reflete a definição de um máximo $(x \geq y$ e $max = x)$ ou $(x < y$ e max = y). As duas implicações a serem provadas são

$\{x \geq y\}\ max := x\ \{(x \geq y \text{ e } max = x) \text{ ou } (x < y \text{ e } max = y)\}$

e

$\{x < y\}\ max := y\ \{(x \geq y \text{ e } max = x) \text{ ou } (x < y \text{ e } max = y)\}$

Cada qual é verdadeira pelo axioma da atribuição. (No primeiro caso, por exemplo, o axioma da atribuição nos diz que algo da forma $E \vee F \rightarrow C$ é verdadeiro, portanto vale que $E \rightarrow C$ e $E$ é equivalente a $x \geq y$.) A regra condicional assegura, então, que o trecho do programa está correto. •

No próximo capítulo veremos como verificar a correção de um comando de laço, onde a execução de uma seção de código pode ser repetida diversas vezes.

Como vimos, a prova de correção envolve uma porção de trabalho detalhado. É uma ferramenta difícil de ser aplicada a programas grandes que já existam. É muito mais simples provar a correção enquanto o programa estiver sendo desenvolvido. Além disso, a lista de asserções do início ao fim especifica a conduta desejada do programa, e pode ser usada desde o início em sua concepção. Além disso, as asserções servem como uma documentação de valor após o término do programa.

## Revisão da Seção 1.5

### Técnicas

- Formulação de fatos e regras na forma do Prolog
- Formulação de perguntas na forma do Prolog
- Determinação das respostas a uma pergunta usando uma base de dados Prolog
- Verificação da correção de um trecho de programa que inclua comandos de atribuição
- Verificação da correção de um trecho de programa que inclua comandos condicionais

### Idéias Principais

Uma linguagem declarativa incorpora predicados, conectivos lógicos e regras de inferência para chegar a teses a partir das hipóteses sobre uma determinada interpretação.
Os elementos deste tipo de linguagem são baseados na lógica de predicados ao invés das instruções que realizam um algoritmo.

Um sistema de lógica formado de axiomas e regras de inferência pode ser usado para provar a correção de trechos de programas.

## Exercícios 1.5

*(Nota:* Nos Exercícios 14, 15, 17 e 20, o * denota multiplicação.)
Os Exercícios de 1 a 6 referem-se à base de dados do Exemplo 26; encontre os resultados das perguntas em cada caso.

1. **is**(*come*(urso, peixinho))

2. **is**(*come*(raposa,coelho))

★3. **which***(x:come(*quati, *x))*

4. ***which****(x:come(x,* mato))

5. **which**(*x:come*(urso,x) **and** *come(x,* coelho))

6. **which***(x:presa(x)* **and** not(*come*(raposa, *x)))*

★7. Formule uma regra Prolog que defina "herbívoro" a fim de incluí-la à base de dados do Exemplo 26.

8. Se a regra do Exercício 7 for incluída na base de dados do Exemplo 26, qual será a resposta à pergunta

   **which***(x:herbívoro(x))*

9. Declare uma base de dados Prolog que forneça informações sobre estados e suas capitais. Algumas cidades são grandes, outras pequenas. Alguns estados estão ao sul, outros ao norte.
   a. Escreva uma pergunta para achar todas as cidades capitais.
   b. Escreva uma pergunta para achar todos os estados cujas capitais sejam cidades pequenas.
   c. Escreva uma pergunta para achar todos os estados ao norte cujas capitais sejam grandes cidades.
   d. Formule uma regra para definir as cidades cosmopolitas como as cidades grandes que sejam capitais dos estados do sul.
   e. Escreva uma pergunta para achar todas as cidades cosmopolitas.

★10. Suponha que exista uma base de dados Prolog que forneça informações sobre autores e os livros que escreveram. Os livros serão classificados como ficção, biografia ou referência.
   a. Escreva uma pergunta para verificar se Chico Buarque escreveu *Estorvo.*
   b. Escreva uma pergunta para achar todos os livros escritos por Paulo Coelho.
   c. Formule uma regra para definir autores de livros não-ficção.
   d. Escreva uma pergunta para achar todos os autores de livros não-ficção.

11. Suponha que exista uma base de dados Prolog que dê informações sobre uma família. Os predicados *homem, mulher* e *pais-de* (que indica se se trata do pai ou da mãe de um elemento) foram incluídos.
    a. Formule uma regra para definir *pai-de.*
    b. Formule uma regra para *definir filha-de.*
    c. Formule uma regra recursiva para definir *ancestral-de.*

12. Suponha que exista uma base de dados Prolog que forneça informações sobre as partes que compõem um motor de automóvel. Os predicados *grande, pequena, parte-de* são fornecidos.
    a. Escreva uma pergunta que encontre todos os itens pequenos que são partes de outros itens.
    b. Escreva uma pergunta que encontre todos os itens grandes que têm subitens pequenos.
    c. Formule uma regra recursiva para definir *componente-de*

★13. De acordo com o axioma da atribuição, qual a pré-condição do seguinte trecho de programa? Explique por que ele funciona.

```
{pré-condição}
   x: = x + 1;
{x = y- 1}
```

14. De acordo com o axioma da atribuição, qual a pré-condição do trecho de programa a seguir? Explique por que ele funciona.

```
{pré-condição}
   x:= 2*x;
{x>y}
```

**15.** Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas.

```
{ x=1}
   y:= x+ 3;
   y : = 2 * y ;
{y = 8}
```

16. Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas.

```
{x>0}
   y:= x + 2;
   z:=y+ 1;
{z>3}
```

★17. Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas:

```
v = x(x - 1);

y:= x- 1;
y:=x*y;
```

**18.** Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas.
v:= 2x+ 1:

```
y : = x;
y := y + y;
y:=y+ 1;
```

★19. Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas.

```
{y = 0}
if y < 5 then
  y:= y + 1
else
  v:=5;
{y = 1}
```

20. Verifique a correção do seguinte trecho de programa com a pré-condição e pós-condição mostradas.

```
{x = 7}
   if x <= 0 then
      v:= x
   else
      y:=2*x;
 {y=  14}
```

**21.** Verifique a correção do trecho de programa a seguir para computar *min(x, y),* o menor dentre dois valores *x* e v:

```
if x <= y then
  min : = x
else
  min := v;
```

**22.** Verifique a correção do seguinte trecho de programa para computar |*x* , o valor absoluto de *x:*

```
if x>= 0 then
  abs := x
else
  abs := —x;
```

## Revisão do Capítulo

### Terminologia

algoritmo
antecedente
argumento válido
asserção
axioma
axioma da atribuição
busca em profundidade
cláusula de Horn
comando
comando condicional
conectivo binário
conectivo unário
conjunção
conseqüente
contradição
cálculo predicado
cálculo proposicional
dedução
definição recursiva
disjunção
domínio
dual de uma equivalência
equivalência
escopo
esquema de axioma
fato Prolog
fator
fórmula bem-formulada (wff)
generalização
hipótese
implicação
interpretação
lei de De Morgan
linguagem declarativa
linguagem descritiva
linguagem procedural
lógica de sentenças
lógica de predicados
lógica proposicional
modus ponens
modus tollens
negação
ocorrência ligada (de uma variável)
ocorrência livre (de uma variável)
parcela
predicado
predicado binário
predicado n-ário
predicado ternário
predicado unário
programa correto
proposição
prova
prova de correção
prova de seqüência
pré-condição
pseudocódigo
pós-condição
quantificador existencial
quantificador universal
regra Prolog
regra condicional de inferência
regra de inferência
regra recursiva
resolução
sistema especialista
sistema formal completo
sistema formal correto
sistemas baseados em regra
sistemas baseados no conhecimento
tabela-verdade
tautologia
teorema
tese
teste do programa
validação do programa
variável ligada
variável livre
verificação do programa
wff predicativa
wff proposicional
wff válida
wffs equivalentes

## Auto-Testes

Responda às seguintes respostas com verdadeiro ou falso sem consultar o capítulo.

### Seção 1.1

1. Uma contradição é qualquer wff proposicional que não é uma tautologia.
2. A disjunção de qualquer wff proposicional com uma tautologia tem o valor-verdade verdadeiro.
3. O Algoritmo *TestaTautologia* determina se qualquer wff proposicional é ou não uma tautologia.
4. Wffs proposicionais equivalentes têm a mesma tabela-verdade para todas as atribuições de valores-verdade a suas componentes.
5. Uma das leis de De Morgan diz que a negação da disjunção é a disjunção das negações (das parcelas).

### Seção 1.2

6. Uma wff predicativa que comece com um quantificador universal é universalmente verdadeira, isto é, é verdadeira em todas as interpretações.
7. Na wff predicativa $(\forall x)P(x, y)$, $y$ é uma variável livre.
8. Algumas wffs predicativas podem não ter valor-verdade em algumas interpretações.
9. O domínio de uma interpretação consiste em valores para os quais a wff predicativa nessa interpretação é verdadeira.
10. Uma wff predicativa não tem interpretação na qual seja falsa.

s

### Seção 1.3

11. O modus ponens permite que qualquer wff proposicional seja derivada dos axiomas.
12. A lógica proposicional é completa porque toda tautologia é um teorema.
13. Uma argumento válido é um argumento no qual a conclusão é sempre verdadeira.
14. O método de dedução permite o uso de hipóteses como passos adicionais na seqüência de demonstração da tese.
15. Todo axioma da lógica proposicional é uma tautologia, mas nem toda tautologia é um axioma.

### Seção 1.4

16. Os axiomas da lógica de predicados permitem que quantificadores existenciais e universais sejam incluídos ou removidos durante a seqüência de demonstração.
17. Um dos axiomas da lógica de predicados diz que "para todo, não" é o mesmo que "não é verdade que exista um".
18. Todo teorema da lógica proposicional é também um teorema da lógica de predicados.
19. Uma wff predicativa que não é válida não pode ser um teorema na lógica de predicados.
20. A generalização deve ser usada o mais cedo possível em uma seqüência de demonstração.

### Seção 1.5

21. Uma regra Prolog descreve um predicado.
22. O modus ponens é um caso especial da resolução do Prolog.
23. Um programa provavelmente correto sempre dá as respostas certas a um dado problema.
24. Se uma asserção após uma atribuição é $y > 4$, então a pré-condição precisa ser $y \geq 4$.
25. A prova de correção envolve o desenvolvimento cuidadoso de conjuntos de dados para testes.

## No Computador

Para os Exercícios 1 a 5, escreva um programa que produza a saída desejada para uma dada entrada.

1. *Entrada:* Valores-verdade para os dois símbolos proposicionais $A$ e $B$
   *Saída:* Valores-verdade correspondentes (devidamente rotulados, naturalmente) para

   $$A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B, A'$$

2. *Entrada:* Valores-verdade para os dois símbolos proposicionais $A$ *e* $B$
   *Saída:* Valores-verdade correspondentes para as wffs

   $$A \rightarrow B' \quad \text{e} \quad B' \wedge [A \vee (A \wedge B)]$$

3. *Entrada:* Valores-verdade para os dois símbolos proposicionais *A, B* e *C.*
   *Saída:* Valores-verdade correspondentes para as wffs

   $A \vee (B \wedge C') \rightarrow B'$ e $A \vee C' \leftrightarrow (A \vee C)'$

4. *Entrada:* Valores-verdade para os três símbolos proposicionais *A, B* e C e uma representação de uma wff proposicional simples. Símbolos especiais podem ser usados como conectivos lógicos, e pode também ser usada a **notação pós-fixa;** por exemplo,

   $A\ B \wedge C \vee$ para $(A \wedge B) \vee C$

   ou

   $A'\ B \wedge$ para $(A' \wedge B)$

   *Saída:* Valores-verdade correspondentes para as wffs

5. *Entrada:* Uma representação de uma wff simples, como no exercício anterior
   *Saída:* A decisão de se a wff é ou não uma tautologia

6. Se você tiver uma versão disponível do Prolog, entre a base de dados do Exemplo 26 e realize as perguntas indicadas lá. Além disso, inclua a regra recursiva *na-cadeia-alimentar* e realize a pergunta

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