Seção 2.3 Recursão e Relação de Recorrência

Definições Recursivas

Uma definição na qual o item que está sendo definido aparece como parte da definição é chamada **definição indutiva** ou **definição recursiva.** À primeira vista isto pode parecer sem sentido — como algo pode ser definido em termos dele próprio? Este procedimento funciona porque as definições recursivas são compostas de duas partes:

1. uma base, onde alguns casos simples do item que está sendo definido são dados explicitamente, e
2. um passo indutivo ou recursivo, onde outros casos do item que está sendo definido são dados em termos dos casos anteriores.

A parte 1 nos fornece um ponto de partida na medida em que trata alguns casos simples; enquanto a parte 2 nos permite construir novos casos a partir desses casos simples, para então construir outros casos a partir desses novos, e assim por diante. (A analogia com demonstrações por indução matemática justifica o nome "definição indutiva". Em uma demonstração por indução, existe a base da indução, isto é, a demonstração de $P(1)$ — ou a demonstração de P para algum outro valor inicial — e existe a hipótese indutiva, onde a validade de $P(k+1)$ é deduzida da validade de P para valores menores.)

A recursão é uma idéia importante que pode ser utilizada para definir seqüências de objetos, coleções mais gerais de objetos e operações sobre objetos. (O Predicado Prolog *na-cadeia-alimentar* da Seção 15 foi definido recursivamente.) Até mesmo algoritmos podem ser definidos recursivamente.

Seqüências Recursivas

Uma **seqüência** S é uma lista de objetos que são enumerados segundo alguma ordem; existe um primeiro objeto, um segundo, e assim por diante. $S(k)$ denota o k-esimo objeto da seqüência. Uma seqüência é definida recursivamente, explicitando-se seu primeiro valor (ou seus primeiros valores) e, a partir daí, definindo-se outros valores na seqüência em termos dos valores iniciais.

EXEMPLO 19 A seqüência S é definida recursivamente por:

1. $S(1) = 2$
2. $S(2) = 2S(n\text{-}1) \text{ para } n \geq 2$

Pela afirmação 1, 5(1), o primeiro objeto em *S, é 2*. Então pela sentença 2, o segundo objeto em *S* é 5(2) = 25(1) = 2(2) = 4. Novamente por 2, S(3) = 2S(2) = 2(4) = 8. Dessa forma podemos deduzir que *S* é a seqüência

2,4,8,16,32,... •

Uma regra semelhante à da sentença 2 no Exemplo 19, na qual se define um valor da seqüência a partir de um ou mais valores anteriores, é chamada **relação de recorrência.**

PRÁTICA 9 A seqüência *T é* definida recursivamente como se segue:

1. $T(1) = 1$
2. $T(n) = T(n-1) + 3$ para $n \geq 2$

Escreva os primeiros cinco valores da seqüência *T*. •

EXEMPLO 20 A famosa **seqüência de Fibonacci** é definida recursivamente por:

$F(1) = 1$
$F(2) = 2$
$F(n) = F(n-2) + F(n-1)$ para $n > 2$

Aqui os primeiros dois valores na seqüência são dados, e a relação de recursão define o *n*-ésimo termo a partir dos dois anteriores. •

PRÁTICA 10 Escreva os oito primeiros valores da seqüência de Fibonacci. •

A seqüência de Fibonacci tem sido estudada exaustivamente. Por ter sido definida de forma recursiva, podemos usar a indução para provar muitas de suas propriedades.

EXEMPLO 21 Prove que na seqüência de Fibonacci,

$F(n+4) = 3F(n + 2) - F(n)$ para todo $n \geq 1$.

Para a base da indução, devemos mostrar dois casos, $n = 1$ e $n = 2$. Para $n = 1$, temos:

$F(5) = 3F(3) - F(1)$

ou (usando valores obtidos na Prática 10)

$5 = 3(2) - 1$

que é verdadeiro. Para $n = 2$,

$F(6) = 3F(4) - F(2)$

ou

$8 = 3(3) - 1$

que também é verdadeiro. Suponhamos agora que para todo r, $1 \leq r \leq k$,

$F(r+4) = 3F(r+2) - F(r)$.

Agora, mostremos o caso $k+1$, onde $k + 1 \geq 3$. Então, desejamos mostrar que:

$F(k + 1 + 4) \stackrel{?}{=} 3F(k + 1 + 2) - F(k + 1)$

ou

$F(k + 5) \stackrel{?}{=} 3F(k + 3) - F(k+ 1)$

Da relação de recorrência na seqüência de Fibonacci temos:

$F(k+5) = F(k+3) + F(k+4)$

e, pela hipótese de indução, para $r = k-1$ e $r = k$, respectivamente, vem:

$F(k+3) = 3F(k+1) - F(k-1)$

e

$F(k+4) = 3F(k+2) - F\{k)$

Logo

$F(k+5) = 3\,F(k+1) - F(k-1) + 3\,F(k+2) - F(k)$
$= 3[F(k+1) - F(k+2)] - [F(k-1) - F(k)]$
$= 3F(k+3) - F(k+1)$ (usando novamente a relação de recorrência)

A indução completa foi necessária aqui, porque a relação de recorrência para a seqüência de Fibonacci usa mais termos do que o termo imediatamente anterior. •

PRÁTICA 11 Na demonstração do Exemplo 21, por que é necessário provar $n = 2$ como um caso especial? •

Conjuntos Recursivos

Os objetos de uma seqüência são ordenados — existe um primeiro objeto, um segundo, e assim por diante. Um *conjunto* de objetos é uma coleção de objetos na qual nenhuma regra de ordenação é imposta. Alguns conjuntos podem ser definidos recursivamente.

EXEMPLO 22 Na Seção 1.1 notamos que certas cadeias de proposições, conectivos lógicos e parênteses tais como $(A \wedge B)' \vee C$, são considerados válidos, enquanto outras cadeias como $\wedge \wedge A'' B$ não são válidas. A sintaxe para ordenar estes símbolos constitui a definição do conjunto de fórmulas proposicionais bem-formuladas, que é uma definição recursiva.

1. Qualquer proposição é uma wff.
2. Se P e Q são wffs, então também o serão $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P') e $(P \leftrightarrow Q)$.

Nós normalmente omitimos parênteses quando isso não nos causa confusão; então escrevemos $(P \vee Q)$ como $P \vee Q$, ou (P') como P'. Iniciando com uma proposição e usando a regra 2 repetidamente, podemos construir qualquer wff proposicional. Por exemplo, A,B, e C são wffs pela regra 1. Pela regra 2,

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

são também wffs. Novamente pela regra 2,

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

é uma wff. Aplicando a regra 2 mais uma vez, obtém-se a wff

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

Eliminando alguns parênteses, podemos escrever a wff como

$[(A \wedge B) \rightarrow C']'$ •

PRÁTICA 12 Mostre como construir a wff $[(A \vee (B')) \rightarrow C]$ a partir da definição do Exemplo 22.

PRÁTICA 13 Uma definição recursiva para o conjunto de pessoas que são ancestrais de João pode ser construída a partir da seguinte premissa:

Os pais de João são ancestrais de João

Forneça a base da indução. •

Cadeias de símbolos obtidas a partir de um "alfabeto" finito são objetos normalmente encontrados na ciência da computação. Os computadores armazenam dados em **cadeias binárias,** ou seja, cadeias do alfabeto consistindo em Os e 1 s; os compiladores entendem as instruções de um programa como cadeias de *tokens,* tais como palavras-chave e identificadores. A coleção de todas as cadeias de tamanho finito de um alfabeto, normalmente chamadas de cadeias *sobre* um alfabeto, podem ser definidas recursivamente (veja o próximo exemplo). Muitos conjuntos de cadeias de caracteres com propriedades especiais têm também definições recursivas.

EXEMPLO 23 O conjunto de todas as cadeias de símbolos (de tamanho finito) sobre um alfabeto A é denotado por A^*. A definição recursiva de A^* é:

1. A **cadeia** vazia À (a cadeia sem símbolos) pertence a A*.
2. Qualquer elemento de A pertence a A^*.
3. Se x e y são cadeias em A^*, então a **concatenação** xy também pertence a A^*.

As partes 1 e 2 constituem a base da recursão, e a parte 3 é o passo recursivo desta definição. Perceba que para qualquer cadeia x, $x\lambda = \lambda x = x$. •

PRÁTICA 14 Se $x = 1.011$ e $y = 001$, escreva as cadeias xy, yx, e $yx\lambda x$. •

PRÁTICA 15 Forneça uma definição recursiva para o conjunto de todas as cadeias binárias que são **palíndromos,** ou seja, cadeias iguais, se lidas do início para o fim ou ao contrário. •

EXEMPLO 24 Suponha que em uma certa linguagem de programação, os identificadores podem ser cadeias alfanuméricas de tamanho arbitrário, mas que devem começar com uma letra. Uma definição recursiva para um conjunto de cadeias desta natureza é:

1. Uma letra é um identificador.
2. Se A é um identificador, então a concatenação de A com qualquer letra ou dígito também é um identificador.

Uma notação mais simbólica para descrever conjuntos de cadeias que são definidas recursivamente é chamada **forma de Backus Naur,** ou **BNF,** originalmente desenvolvida para definir a linguagem de programação ALGOL. Na notação BNF, itens que são definidos em termos de outros itens são envolvidos por menor e maior (< >), enquanto itens específicos que não são definidos mais adiante não aparecem entre menor e maior. A linha vertical | indica uma escolha com o mesmo significado da palavra *ou*. A definição BNF de um identificador é:

⟨identificador⟩::= ⟨letra⟩ | ⟨identificador⟩⟨letra⟩ | ⟨identificador⟩⟨dígito⟩
⟨letra⟩ ::= $a|b|c| \cdots |z$
⟨dígito⟩ ::= $0|1| \cdots |9$

Então o identificador *mel* é obtido da definição por uma seqüência de escolhas tais como:

⟨identificador⟩ pode ser ⟨identificador⟩⟨dígito⟩
que pode ser ⟨identificador⟩2
que pode ser ⟨identificador⟩⟨letra⟩2
que pode ser ⟨identificador⟩*e*2
que pode ser ⟨letra⟩*e*2
que pode ser *me*2 •

Operações Recursivas

Podemos definir recursivamente certas operações sobre objetos, como nos exemplos abaixo.

EXEMPLO 25 Uma definição recursiva para a operação de exponenciação a^n, definida para um número real a diferente de zero e n inteiro não-negativo é:

1. $a^0 = 1$
2. $a^n = (a^{n-1})a$ para $n \geq 1$ •

EXEMPLO 26 Uma definição recursiva para a multiplicação de dois inteiros positivos m e n é:

1. $m(1) = m$
2. $m(n) = m(n-1) + m$ para $n \geq 2$ •

PRÁTICA 16 Seja x uma cadeia sobre algum alfabeto. Forneça uma definição recursiva para a operação x" (concatenação de x com ele próprio n vezes) para $n \geq 1$. •

Na Seção 1.1, definimos a operação de disjunção lógica de duas proposições. Isso pode servir como a base da recursão para a definição recursiva da disjunção de n proposições, para $n \geq 2$:

1. $A_1 \vee A_2$ definida como na Seção 1.1
2. $A_1 \vee \cdots \vee A_n = (A_1 \vee \cdots \vee A_{n-1}) \vee A_n$ para $n > 2$ **(D**

Usando esta definição, podemos generalizar a propriedade associativa da disjunção (equivalência tautológica 2a) para afirmar que em uma disjunção de n proposições, o grupamento por parênteses é desnecessário, porque todos esses grupamentos são equivalentes à expressão geral da disjunção de n proposições. Simbolicamente, para qualquer n, com $n \geq 3$ e qualquer p, com $1 \leq p \leq n-1$,

$$(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_n) \Leftrightarrow A_1 \vee \cdots \vee A_n$$

Esta equivalência pode ser provada por indução em n. Para $n = 3$,

$$A_1 \vee (A_2 \vee A_3) \Leftrightarrow (A_1 \vee A_2) \vee A_3 \qquad \text{(pela equivalência 2a)}$$
$$= A_1 \vee A_2 \vee A_3 \qquad \text{(pela equação (1))}$$

Suponha que para $n = k$ e $1 \leq p \leq k-1$

$$(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k) \Leftrightarrow A_1 \vee \cdots \vee A_k$$

Então para $1 \leq p \leq k$

$$\begin{aligned}
&(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_{k+1}) \\
&\quad = (A_1 \vee \cdots \vee A_p) \vee [(A_{p+1} \vee \cdots \vee A_k) \vee A_{k+1}] && \text{(pela equação (1))} \\
&\quad \Leftrightarrow [(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k)] \vee A_{k+1} && \text{(pela equivalência 2a)} \\
&\quad \Leftrightarrow (A_1 \vee \cdots \vee A_k) \vee A_{k+1} && \text{(pela hipótese de indução)} \\
&\quad = A_1 \vee \cdots \vee A_{k+1} && \text{(pela equação (1))}
\end{aligned}$$