11

Indução Estrutural

> Correctness is clearly the prime quality. If a system does not do what it is supposed to do, then everything else about it matters little.
>
> Berthrand Meyer, Cientista da Computação

11.1 Motivação

Como vimos nos dois capítulos anteriores (capítulos 9 e 10), a indução matemática é uma técnica de demonstração aplicável em diversas situações. Nestes capítulos, apresentamos um enfoque sobre a indução matemática que essencialmente abordou problemas matemáticos, porém, esta técnica é aplicável também a provas de propriedades sobre estruturas de dados recursivas e algoritmos sobre estas. A este tipo de demonstração de indução, damos o nome de indução estrutural.

O objetivo deste capítulo é o estudo da indução estrutural para demonstração de correção sobre alguns algoritmos sobre estruturas de dados simples como listas. Para evitar problemas relativos à utilização de atribuição de variáveis, e aspectos específicos de linguagens de programação, representaremos estruturas de dados, como definições sintáticas e algoritmos como funções, de maneira similar ao que fizemos no capítulo 1.

11.2 Indução Estrutural

Conforme apresentado no capítulo 1, definições sintáticas de conjuntos de termos devem possuir elementos iniciais (casos base) e, opcionalmente, formas de se construir termos mais complexos a partir de termos existentes (passo(s) indutivo(s)). De maneira simples, a técnica de indução estrutural pode ser resumida da seguinte maneira: Seja P a propriedade a ser demonstrada para todo termo

t pertencente a um conjunto $\mathcal{T}$. Para constatar que $P(t)$ é verdade basta mostrar que esta propriedade é verdadeira para cada um dos casos base e passos indutivos da definição do conjunto $\mathcal{T}$.

As próximas seções apresentarão a indução estrutural em exemplos concretos: números naturais na notação de Peano ($\mathcal{N}$) e listas.

11.2.1 Números Naturais na Notação de Peano

Conforme apresentado no capítulo 1, o conjunto $\mathcal{N}$, dos termos que representam números naturais na notação de Peano, pode ser definido pelas seguintes regras:

$$\begin{array}{l} zero \in \mathcal{N} \\ \text{se } n \in \mathcal{N} \text{ então } suc\, n \in \mathcal{N} \end{array}$$

Nesta notação, o número natural 3 é representado pelo termo $suc\,(suc\,(suc\,zero))$, isto é todo número natural ou é representado pelo termo $zero$ ou por uma sequência de n suc's que terminam com a constante $zero$.

Usando esta notação, podemos definir como funções recursivas operações sobre números naturais, como por exemplo, a adição:

$$\begin{array}{lcll} plus(zero, m) & = & m & (1) \\ plus(suc\, n, m) & = & suc(plus(n, m)) & (2) \end{array}$$

Numeramos as equações da definição de $plus$ para referenciar uma equação específica quando necessário. Como um exemplo da utilização da função $plus$, considere a soma: 2+3, que é representada como $plus(suc(suc\,zero), suc(suc(suc\,zero)))$:

$$\begin{array}{lcl} plus(suc(suc\,zero), suc(suc(suc\,zero))) & \equiv & \\ suc(plus(suc\,zero, suc(suc(suc\,zero)))) & \equiv & \{\text{pela equação 2 de } plus\} \\ suc(suc(plus(zero, suc(suc(suc\,zero))))) & \equiv & \{\text{pela equação 2 de } plus\} \\ suc(suc(suc(suc(suc\,zero)))) & & \{\text{pela equação 1 de } plus\} \end{array}$$

Note que o processo de execução da função $plus$ é completamente determinado por sua definição: se o primeiro parâmetro desta função é igual a $zero$, o seu resultado será o segundo parâmetro (m, na definição de $plus$). Porém, se o primeiro parâmetro não for igual a $zero$, necessariamente este deverá ser $suc\,n$, para algum $n \in \mathcal{N}$, e o resultado será o sucessor da chamada recursiva $plus(n, m)$. É útil que você faça mais algumas execuções da função $plus$ até que você tenha compreendido completamente seu funcionamento.

De acordo com a definição da função $plus$, note que $\forall m.plus(zero, m) \equiv m$ (pela equação 1 de $plus$), porém não é imediato que $\forall n.plus(n, zero)$. Isto se deve que o termo $plus(zero, m)$ pode ser reduzido imediatamente a m, de acordo com a equação 1 de $plus$, enquanto $plus(n, zero)$ não, uma vez que não é possível determinar se n é ou não igual a $zero$.

Em lógica, dizemos que a expressão $plus(zero, m)$ é *igual por definição*[1] a m, uma vez que esta igualdade pode ser deduzida diretamente pela definição de $plus$, executando-a. Note que apesar de evidentemente verdadeira, a igualdade $plus(n, zero)$ não pode ser considerada igual por definição a n, visto que não existe uma única possibilidade de execução para esta expressão pois, n pode ser ou não igual a $zero$. Neste caso, se desejamos demonstrar tal igualdade,

[1] Tradução livre do termo: "definitionally equal to".

devemos prová-la usando indução. Antes disso, vamos apresentar a definição do princípio de indução estrutural para o conjunto $\mathcal{N}$.

Definição 78 (Indução sobre $\mathcal{N}$)**.** Seja P uma propriedade qualquer sobre elementos de $\mathcal{N}$. Podemos demonstrar que $\forall n.n \in \mathcal{N} \rightarrow P(n)$ usando a seguinte fórmula:

$$P(zero) \land \forall n.n \in \mathcal{N} \land P(n) \rightarrow P(suc\, n)$$

■

Note que esta definição é exatamente igual ao princípio de indução matemática que vimos no capítulo 9, a menos do uso do conjunto $\mathcal{N}$ ao invés de $\mathbb{N}$ e das constantes *zero* e *suc*.

A seguir, apresentamos a prova da propriedade $\forall n.n \in \mathcal{N} \rightarrow plus(n, zero) \equiv n$, usando indução estrutural.

Teorema 54. *Para todo $n \in \mathcal{N}$, $plus(n, zero) \equiv n$.*

Demonstração. Esta demonstração será por indução sobre n.

1. Caso base ($n = zero$). Neste caso, temos que

$$\begin{array}{lll} plus(zero, zero) & \equiv & \\ zero & & \{\text{pela equação 1 de } plus\} \end{array}$$

   conforme requerido.

2. Passo indutivo ($n = suc\, n'$). Suponha $n' \in \mathcal{N}$ arbitrário e que $plus(n', zero) \equiv n'$. Temos que:

$$\begin{array}{lll} plus(suc\, n', zero) & \equiv & \\ suc(plus(n', zero)) & \equiv & \{\text{pela equação 2 de } plus\} \\ suc\, n' & & \{\text{pela hipótese de indução}\} \end{array}$$

   conforme requerido.

□

Observe que no passo indutivo desta demonstração, consideramos que o primeiro parâmetro n é tal que $n = suc\, n'$. A hipótese de indução é obviamente definida para n', o antecessor de n.

Usualmente, provas por indução estrutural sobre funções devem realizar a indução sobre o parâmetro recursivo da definição da função. Como a função *plus* é definida recursivamente sobre seu 1^o parâmetro, provas sobre esta devem ser feitas utilizando indução sobre este. Como um segundo exemplo de demonstração por indução estrutural, considere demonstrar que a adição é uma operação associativa, isto é:

$$plus(n, plus(m, p)) \equiv plus(plus(n, m), p)$$

Essa propriedade é demonstrada no teorema seguinte.

Teorema 55 (*plus* é uma operação associativa)**.** *Para todo $n, m, p \in \mathcal{N}$, temos que $plus(n, plus(m, p)) \equiv plus(plus(n, m), p)$.*

Demonstração. Esta prova será por indução sobre n. Suponha $m, p \in \mathcal{N}$ arbitrários.

1. Caso base ($n = zero$). Temos que:

$$\begin{array}{lll} plus(zero, plus(m,p)) & \equiv & \\ plus(m,p) & & \{\text{pela equação 1 de } plus\} \end{array}$$

conforme requerido.

2. Passo indutivo ($n = suc\, n'$): Suponha $n' \in \mathcal{N}$ arbitrário e que $plus(n', plus(m,p)) \equiv plus(plus(n',m),p)$. Temos que:

$$\begin{array}{lll} plus(suc\, n', plus(m,p)) & \equiv & \\ suc(plus(n', plus(m,p))) & \equiv & \{\text{pela equação 2 de } plus\} \\ suc(plus(plus(n',m),p)) & \equiv & \{\text{pela hipótese de indução}\} \\ plus(suc(plus(n',m)),p) & \equiv & \{\text{pela equação 2 de } plus\} \\ plus(plus(suc\, n',m),p) & & \{\text{pela equação 2 de } plus\} \end{array}$$

conforme requerido.

□

11.2.2 Exercícios

1. Prove que a soma de números na notação de Peano é uma operação comutativa, isto é, prove que:

$$\forall n. n \in \mathcal{N} \rightarrow \forall m. m \in \mathcal{N} \rightarrow plus(n,m) \equiv plus(m,n)$$

2. Considere a seguinte definição alternativa da soma na notação de Peano.

$$\begin{array}{lccc} plus_{alt}(n, zero) & = & n & (1) \\ plus_{alt}(n, suc\, m) & = & suc(plus_{alt}(n,m)) & (2) \end{array}$$

Prove que para quaisquer valores $n, m \in \mathcal{N}$, $plus_{alt}(n,m) \equiv plus(n,m)$.

3. Defina a função $mult(n,m)$ que realiza a multiplicação de números naturais na notação de Peano.

4. Prove que a função de multiplicação definida por você é uma operação comutativa.

11.2.3 Listas

Nesta seção, consideraremos algumas funções sobre listas e provas de propriedades sobre estas utilizando indução estrutural. No capítulo 1, apresentamos o conjunto de listas cujos elementos são de $\mathcal{T}$, $List\, \mathcal{T}$, como sendo os termos definidos recursivamente como:

$[\,] \in List\, \mathcal{T}$
se $t \in \mathcal{T}$ e $ts \in List\, \mathcal{T}$ então $t :: ts \in List\, \mathcal{T}$

Por questão de simplicidade, vamos considerar que os elementos de listas são valores booleanos, cuja definição apresentamos a seguir:

$$T \in \mathcal{B}$$
$$F \in \mathcal{B}$$

É importante notar que esta simplificação será feita apenas para fins de facilitar o entendimento e a escrita de exemplos. Todas as funções e suas respectivas propriedades são válidas para listas cujos elementos pertencem a um conjunto $\mathcal{T}$ qualquer. Desta forma, representaremos a lista que contém os elementos T e F, nesta ordem, como: $T :: F :: [\,]$. Note que o valor que representa uma lista vazia ($[\,]$) possui funcionalidade similar ao um ponteiro "nulo" em implementações de listas encadeadas em linguagens de programação como C/C++, a de indicar o final da lista em questão.

Como exemplos de funções sobre listas, considere, as funções para determinar o número de elementos ($length$) e concatenação de duas listas (++) apresentadas a seguir:

$$\begin{array}{lcll} length\ [\,] & = & 0 & (1) \\ length\ (t :: ts) & = & 1 + length\ ts & (2) \end{array}$$

$$\begin{array}{lcll} [\,] +\!+\ ys & = & ys & (1) \\ (x :: xs) +\!+\ ys & = & x :: (xs +\!+\ ys) & (2) \end{array}$$

Novamente, numeramos as equações para futura referência. Antes de apresentarmos um primeiro exemplo de propriedade a ser demonstrada para listas, vamos definir o princípio de indução para listas.

Definição 79 (Indução Estrutural para $List\ \mathcal{T}$)**.** Seja $\mathcal{T}$ um conjunto qualquer. Seja P uma propriedade sobre o conjunto de listas finitas de elementos do conjunto $\mathcal{T}$, $List\ \mathcal{T}$. Então, podemos provar que $\forall t.t \in List\ \mathcal{T} \to P(t)$ usando a seguinte fórmula:

$$P([\,]) \land \forall x.x \in \mathcal{T} \to \forall xs.xs \in List\ \mathcal{T} \land P(xs) \to P(x :: xs)$$

■

Intuitivamente, podemos provar que uma propriedade é verdadeira para todas as listas finitas se formos capazes de provar que esta vale para a lista vazia e, além disso, provarmos que a propriedade continua sendo verdadeira se inserirmos um novo elemento em uma lista qualquer para a qual a propriedade em questão era válida.

Como um exemplo de demonstração por indução sobre listas, considere a seguinte propriedade que pode ser usada para caracterizar a correção de um algoritmo de concatenação de duas listas: o tamanho da concatenação de duas listas xs e ys é igual a soma dos tamanhos de cada uma destas listas. Mais formalmente, a propriedade em questão é:

$$\forall xs.xs \in List\,\mathcal{T} \to \forall ys.ys \in List\,\mathcal{T} \to length(xs\ \texttt{++}\ ys) = length\,xs + length\,ys$$

Essa propriedade é facilmente demonstrada por indução sobre a primeira lista (xs). A indução será feita sobre a primeira lista devido ao fato de que a concatenação é definida recursivamente sobre a primeira lista fornecida como parâmetro.

Teorema 56. *Seja $\mathcal{T}$ um conjunto qualquer de termos. Então para todo $xs, ys \in List\,\mathcal{T}$, temos que $length(xs\ +\!+\ ys) = length\,xs + length\,ys$.*

Demonstração. A prova será por indução sobre xs. Suponha $ys \in List\,\mathcal{T}$ arbitrário.

1. Caso base ($xs = [\,]$). Temos que:

$$\begin{array}{lll} length([\,]\ +\!+\ ys) & \equiv & \\ length\,ys & \equiv & \{\text{pela equação 1 de } +\!+\} \\ 0 + length\,ys & \equiv & \\ length\,[\,] + length\,ys & & \{\text{pela equação 1 de } +\!+\} \end{array}$$

   conforme requerido.

2. Passo indutivo. Suponha $x \in \mathcal{T}$ arbitrário e que $length(xs\ +\!+\ ys) = length\,xs + length\,ys$. Temos que:

$$\begin{array}{lll} length((x :: xs)\ +\!+\ ys) & \equiv & \\ length(x :: (xs\ +\!+\ ys)) & \equiv & \{\text{pela equação 2 de } +\!+\} \\ 1 + length(xs\ +\!+\ ys) & \equiv & \{\text{pela equação 2 de } length\} \\ 1 + length\,xs + length\,ys & \equiv & \{\text{pela hipótese de indução}\} \\ length(x :: xs) + length\,ys & & \{\text{pela equação 2 de } length\} \end{array}$$

   conforme requerido.

□

Para o nosso próximo exemplo de uma propriedade sobre listas, considere a função *reverse*, que inverte uma lista fornecida como parâmetro.

$$\begin{array}{lcll} reverse\,[\,] & = & [\,] & (1) \\ reverse\,(x :: xs) & = & reverse\,xs\ +\!+\ (x :: [\,]) & (2) \end{array}$$

De maneira simples, *reverse* move o primeiro elemento da lista fornecida como parâmetro para o final do resultado de se inverter o restante desta lista. Note que só é possível inserir um elemento na primeira posição de uma lista. Se desejamos inserir um elemento ao final de uma lista, devemos concatená-lo ao final e não simplesmente inserí-lo. Por isso, definimos a função *reverse* em termos da operação de concatenação de duas listas.

Como exemplo do funcionamento da função *reverse*, considere a seguinte execução desta para a lista $T :: F :: T :: [\,]$, apresentada a seguir:

$$\begin{array}{lll} reverse(T :: F :: T :: [\,]) & \equiv & \\ reverse(F :: T :: [\,])\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 2 de } reverse\} \\ (reverse(T :: [\,])\ +\!+\ (F :: [\,]))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 2 de } reverse\} \\ ((reverse\,[\,]\ +\!+\ (T :: [\,]))\ +\!+\ (F :: [\,]))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 2 de } reverse\} \\ (([\,]\ +\!+\ (T :: [\,]))\ +\!+\ (F :: [\,]))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 1 de } reverse\} \\ (((T :: [\,]))\ +\!+\ (F :: [\,]))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 1 de } +\!+\} \\ (T :: ([\,]\ +\!+\ (F :: [\,])))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 2 de } +\!+\} \\ (T :: (F :: [\,]))\ +\!+\ (T :: [\,]) & \equiv & \{\text{pela equação 1 de } +\!+\} \\ T :: ((F :: [\,])\ +\!+\ (T :: [\,])) & \equiv & \{\text{pela equação 2 de } +\!+\} \\ T :: (F :: ([\,]\ +\!+\ (T :: [\,]))) & \equiv & \{\text{pela equação 2 de } +\!+\} \\ T :: (F :: (T :: [\,])) & & \{\text{pela equação 1 de } +\!+\} \end{array}$$

Como exemplo de propriedade sobre a função *reverse*, apresentaremos como esta se relaciona com a operação de concatenação de listas.

Teorema 57. *Seja $\mathcal{T}$ um conjunto qualquer de termos. Então para todo $xs, ys \in List\,\mathcal{T}$, temos que $reverse(xs \;{+}{+}\; ys) \equiv reverse\,ys \;{+}{+}\; reverse\,xs$.*

Demonstração. A prova será por indução sobre xs. Suponha $ys \in List\mathcal{T}$ arbitrário.

1. Caso base ($xs = [\,]$). Temos que:

$$\begin{array}{lll} reverse([\,]{+}{+}ys) & \equiv & \\ reverse\,ys & \equiv & \{\text{pela equação 1 de ++}\} \\ reverse\,ys + +[\,] & & \{\text{pela equação 1 de ++}\} \end{array}$$

   conforme requerido.

2. Passo indutivo. Suponha $x \in \mathcal{T}$ arbitrário e que $reverse(xs{+}{+}ys) \equiv reverse\,ys{+}{+}reverse\,xs$. Temos que:

$$\begin{array}{lll} reverse((x :: xs){+}{+}ys) & \equiv & \\ reverse(x :: (xs{+}{+}ys)) & \equiv & \{\text{pela equação 2 de ++}\} \\ reverse(xs{+}{+}ys){+}{+}(x :: [\,]) & \equiv & \{\text{pela equação 2 de } reverse\} \\ (reverse\,ys{+}{+}reverse\,xs){+}{+}(x :: [\,]) & \equiv & \{\text{pela hipótese de indução}\} \\ reverse\,ys{+}{+}(reverse\,xs{+}{+}(x :: [\,])) & \equiv & \{\text{++ é associativo}\} \\ reverse\,ys{+}{+}reverse(x :: xs) & & \{\text{pela equação 2 de } reverse\} \end{array}$$

□

Note que nesta demonstração usamos, sem demonstrar, o fato de que a operação de concatenação de listas é associativa, isto é:

$$\forall xs.\forall ys.\forall zs.xs \in List\,\mathcal{T} \land ys \in List\,\mathcal{T} \land zs \in List\,\mathcal{T} \to xs{+}{+}(ys{+}{+}zs) \equiv (xs{+}{+}ys){+}{+}zs$$

Esta demonstração simples é deixada como exercício para o leitor.

11.2.4 Exercícios

1. Prove que a concatenação de listas é uma operação associativa.

2. Prove o seguinte teorema envolvendo as funções *length* e *reverse*: Para toda lista $xs \in List\,\mathcal{T}$, temos que $length(reverse\,xs) \equiv length\,xs$.

3. Considere a seguinte definição alternativa de uma função que inverte uma dada lista:

$$\begin{array}{llll} reverse_{alt}\,xs & = & rev\,xs\,[\,] & (1) \\ & & & \\ rev\,[\,]\,ys & = & ys & (1) \\ rev\,(x :: xs)\,ys & = & rev\,xs\;\;(x :: ys) & (2) \end{array}$$

   (a) Mostre, passo a passo, a execução de $reverse_{alt}(T :: F :: F :: [\,])$.

   (b) Prove que para toda lista $xs \in List\,\mathcal{T}$, $reverse_{alt} xs \equiv reverse\,xs$, em que $reverse$ é a primeira definição apresentada de reverse neste texto.

4. Prove que para toda lista $xs \in List\,\mathcal{T}$, $reverse(reverse\,xs) \equiv xs$.