O Princípio da Indução e suas Aplicações

Gabriel Alves 17/0033813,
Henrique Mariano 17/0012280,
Matheus Breder 17/0018997,
Yuri Serka 17/0024385.

19 de novembro de 2018

Resumo

Neste trabalho foi trabalhado os conceitos de Indução, que pode ser Fraca ou Forte e como elas se completam. Também foi proposto a análise da correção do algoritmo de ordenação "*InsertionSort*".

1 Introdução

A indução matemática é um poderoso artifício matemático utilizado para demonstrar a verdade de um número infinito de proposições. Há, no entanto, dois tipos de indução, a fraca e a forte.

A indução fraca toma como passo base mostrar que o enunciado vale para $n = b$ e então é feito o passo indutivo que consiste em mostrar que, se o enunciado vale para $n = k \geq b$, então o mesmo enunciado vale para $n = k+1$.

Matemáticamente podemos escrever o Princípio da Indução Fraca, também conhecidada como Princípio da Indução Matemática, como:

$$PIM = \begin{cases} h_1 = P(0), & \text{onde } P \text{ é uma propriedade qualquer} \\ \forall n(P(n)), & \text{se} \quad \forall m(P(m) \implies P(m+1)) \end{cases}$$

A indução forte ou princípio da boa ordem, toma como caso base mostrar que o enunciado vale para todos os números anteriores ao que se quer provar, se isso acontecer então a propriedade vale para este número.

Matemáticamente podemos escrever o Princípio da Indução Forte como:

$$PIF = \forall n(P(n)), \quad \text{se} \quad \forall k(\forall m(m < k, P(m)) \implies P(k))$$

Com isto podemos provar grande parte dos problemas de indução que forem propostos.

2 Indução forte é equivalente a indução fraca

Queremos provar que a indução fraca complementa a indução forte e vice-versa, portanto temos algo como:

$$\forall m(P(m) \longrightarrow P(m+1)) \iff \forall k(\forall m(m < k, P(m)) \longrightarrow P(k))$$

Portanto temos duas provas para analisar.

$$\forall m(P(m) \longrightarrow P(m+1)) \implies \forall k(\forall m(m<k, P(m)) \longrightarrow P(k)) \tag{1}$$

Primeiro foi tentado resolver usando dedução natural como será mostrado a seguir.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall m(m<k,\varphi[\frac{x}{m}])]^x}{m_0<k,\varphi[\frac{x}{m_0}]}\,(\forall e) \qquad \dfrac{\forall m(\varphi[\frac{x}{m}] \longrightarrow \varphi[\frac{x}{m+1}])}{\varphi[\frac{x}{m_0}] \longrightarrow \varphi[\frac{x}{m_0+1}]}\,(\forall e)}{\varphi[\frac{x}{m_0+1}]}\,(\rightarrow_e)}{\forall x\varphi}\,(\forall i)}{\varphi[\frac{x}{k_0}]}\,(\forall e)}{\forall m(m<k,\varphi[\frac{x}{m}]) \longrightarrow \varphi[\frac{x}{k_0}]}\,(\rightarrow_i),x}{\forall k(\forall m(m<k,\varphi[\frac{x}{m}]) \longrightarrow \varphi[\frac{x}{k}])}\,(\forall i)$$

No entanto ainda há dúvidas se esta prova está correta, pois já nos primeiros ramos da árvore temos: $m_0 < k, \varphi[\frac{x}{m_0}] \quad e \quad \varphi[\frac{x}{m_0}] \longrightarrow \varphi[\frac{x}{m_0+1}]$ e não temos total certeza se os dois $\varphi[\frac{x}{m_0}]$ são iguais.

Todavia, apenas analisando a equação (1) e entendendo-a chega-se a conclusão de que, pela hipótese temos que se $\varphi[\frac{x}{k}] = T \implies \varphi[\frac{x}{k+1}] = T$, e como na indução forte temos de provar que a propriedade vale para todos os menores que um t qualquer, quando pararmos em t-1, então por hipótese teremos que $\varphi[\frac{x}{(t-1)+1}] = \varphi[\frac{x}{t}] = T$ que é exatamente o que queremos provar.

$$\forall k(\forall m(m<k, P(m)) \longrightarrow P(k)) \implies \forall m(P(m) \longrightarrow P(m+1)) \tag{2}$$

Foi tentado montar a árvore para esta prova também. Abaixo está mostrado o que conseguimos fazer:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi[\frac{x}{m_0}]]^x \qquad \dfrac{\dfrac{\dfrac{\forall k(\forall m(m<k,\varphi[\frac{x}{m}]) \longrightarrow \varphi[\frac{x}{k}])}{\forall m(m<k_0,\varphi[\frac{x}{k_0}] \rightarrow \varphi[\frac{x}{k_0}])}\,(\forall e)}{m_0<k_0,\varphi[\frac{x}{m_0}] \rightarrow \varphi[\frac{x}{k_0}]}\,(\forall e)}{}}{\varphi[\frac{x}{k_0}]}\,(\rightarrow_e)}{\forall x\varphi}\,(\forall i)}{\varphi[\frac{x}{m_0+1}]}\,(\forall e)}{\varphi[\frac{x}{m_0}]) \longrightarrow \varphi[(\frac{x}{m_0+1}])}\,(\rightarrow_i),x}{\forall m(\varphi[\frac{x}{m}]) \longrightarrow \varphi[\frac{x}{m+1}])}\,(\forall i)$$

Analisando a equação (2) temos que se a propriedade vale para todos os anteriores a k, ou seja, $\forall m : m < k, \varphi[\frac{x}{m}] = T$, e definindo o conjunto com todos os elementos menores a k como sendo $\mathbb{M}$, então se pegarmos qualquer i $\in \mathbb{M}$, teremos $i < k$. Mas e se irmos além e selecionar um elemento j, tal que j = i - 1, logo $j < i$, como tanto i como j são menores que k então são verdadeiros pela hipótese. Agora temos que i pode ser provado por j, pois ao aplicarmos a indução fraca teremos: $\varphi[\frac{x}{j}] = T$ e $\varphi[\frac{x}{i}] = T$, mas $\varphi[\frac{x}{j}] = \varphi[\frac{x}{i-1}]$ então $\varphi[\frac{x}{i}]$ deve ser verdade, que é o que a indução forte garante.

3 insertion sort é correto

4 Tentativas anteriores

Podemos dizer que a prova que encontra-se abaixo, decorre dos esforços da equipe em outra tarefa: Comprovar a corretude do algoritmo merge_sort. Desse modo, conseguimos trazer as experiências adquiridas com esta tarefa, para cá, e criar a prova que se segue. Já com o auxílio de uma definição de sorted? e outras, em um documento de teoria do pvs, presente em nosso repositório.

4.1 Algoritmo correto

Um algoritmo é definido como correto se cumpre as responsabilidades a ele denominadas. Ou seja, por exemplo, um algortimo geral de ordenação é correto se ao ser aplicado sob qualquer lista de objetos reconhecidamente ordenáveis, retornar uma lista de mesma cardinalidade, e com os mesmo termos, ordenados.

4.2 Um teorema que afira a corretude

Considerando os pseudo-códigos abaixo, que definem o algoritmo insertionSort, recursivamente, nós temos:

$$InsertionSort(l) = \begin{cases} l, & \text{se } l \text{ é nula} \\ insert(h, insertionSort(l')), & \text{se } l = h :: l' \end{cases}$$

Sendo o insert:

$$Insert(x,l) = \begin{cases} x :: null, & \text{se } l \text{ é nula} \\ x :: l, & \text{se } l = h :: l' \text{ e } x \leq h \\ h :: (Insert(x,l')), & \text{se } l = h :: l' \text{ e } x > h \end{cases}$$

Desse modo, para realizar uma prova correta acerca da corretude do algoritmo, podemos definir uma função que nos indique se há ou não uma lista ordenada (em linguagem do pvs):

```
sorted?(l:list[nat]) : RECURSIVE boolean =
  CASES l OF
    null: TRUE,
    cons(h, tl): CASES tl OF
           null: TRUE,
           cons(hh, ttl): (h <= hh) AND sorted?(tl)
           ENDCASES
  ENDCASES
  MEASURE length(l)
```

Basicamente esta função checa recursivamente, para cada termo da lista, a partir de sua cabeça, se este termo é menor ou igual ao termo posterior. Se isso ocorre em todos os casos, então temos uma lista ordenada. Também há o tratamento especial para o caso em que recebe-se uma lista nula: A mesma é considerada ordenada. Poderíamos então, definir em pseudo-código:

$$sorted?(l) = \begin{cases} TRUE, & \text{se } l \text{ é nula} \\ h :: l' = & \begin{cases} TRUE, & \text{se } l' \text{ é nula} \\ TRUE, & \text{se para } u :: l'' : (h \leq u) \quad AND \quad sorted?(l'') \end{cases} \\ FALSE, & \text{caso contrário} \end{cases}$$

Certamente existem definições melhores para este algoritmo. No entanto, o mais importante aqui é considerar que sorted? nos indica se há uma lista ordenada, segundo os conceitos já explicitados mais acima.

Desse modo, podemos construir o seguinte teorema, o qual iremos provar, e que ao ser provado nos levará ao resultado que esperamos: Provaremos que insertionSort é um algoritmo correto:

$$\forall l : list[reais] \quad sorted?(l) \implies sorted?(insertionSort(l))$$

Este teorema nos diz que: Se houver uma lista não ordenada, então teremos uma lista ordenada após aplicar o InsertionSort, ou se houver uma lista ordenada, então ainda teremos uma lista ordenada ao aplicar o insertionSort.

Mais adiante, para nos auxiliar, podemos considerar a seguinte função:

$$antecedente(x,l) = \begin{cases} b, & \text{se } b :: l' \quad e \quad l' = x :: l'' \\ antecedente(x,l'), & \text{se } l = h :: l' \quad e \quad l' = u :: l'' \quad e \quad u \neq x \\ null, & \text{se } l' \text{ é nula} \end{cases}$$

4.3 Indução forte no teorema que afere a corretude

Desse modo, podemos provar o nosso teorema por meio de uma indução forte, no conjunto dos reais. No entanto, essa prova também valeria para qualquer lista de objetos reconhecidamente ordenáveis:

Comecemos criando dois passos base:

$$i) \quad l = null, \text{ onde l é uma lista}$$

Assim aplicando insertionSort em l temos uma lista ordenada por definição.

$$ii) \quad l = a, \text{ onde l é uma lista e } a \in \mathbb{R}$$

Assim aplicando insertionSort em l termos uma lista ordenada por definição.

Podemos então criar o nosso passo indutivo no tamanho da lista (que é a característica que é reduzida ao passo da recursão):

$$iii) \forall K, M : list[reais] \quad e \quad length(K) < length(M) :$$

É verdadeiro:

$$sorted?(K) \implies sorted?(insertionSort(K))$$

Ou seja, insertionSort(K) é uma lista ordenada. Sendo que:

$$M = a :: insertionSort(K), \text{ uma lista de reais}$$

então temos:

$$insertionSort(K) = h :: insertionSort(K)', \text{ h é a cabeça da lista.}$$

temos pela definição de insertionSort, quando aplicarmos insertionSort(M):

$$Insert(a, insertionSort(K))$$

, mas pela hípotese de indução iii), insertionSort(K) é uma lista ordenada. Então temos os casos:

$$1) \quad a \leq h$$

então:

$$a :: insertionSort(K)$$

é o retorno, que por definição é uma lista ordenada (a cabeça é menor ou igual que a cabeça de insertionSort(K), que também é uma lista ordenada). Ou seja: sorted?(a::insertionSort(K)) retorna TRUE.

$$2) \quad a > h$$

então:

$$h :: insert(a, insertionSort(K)')$$

é o retorno.

Neste caso podemos dizer que:

$$*) \quad \exists j, \text{ tal que } a \leq j,$$

ou

$$**) \quad \text{a encontra o termo nulo}$$

Logo, se temos *) então sabemos que:

$$a > antecedente(insertionSort(K)', j) \quad a \leq j$$

, portanto ao inserir a após o antecedente(insertionSort(K), j), ainda temos uma lista ordenada, pois insertionSort(K) é ordenado pela hipótese de indução iii). Ou seja sorted?(h::insert(a, insertionSort(K)')) retorna TRUE.

e se temos **) então sabemos que por definição a será inserido após o último termo não nulo de insertionSort(K), e:

$$a > antecedente(insertionSort(K)', null)$$

, ou seja, a é maior que o termo antecedente ao nulo. Logo, ainda temos uma lista ordenada. Ou seja sorted?(h::insert(a, insertionSort(K)')) retorna TRUE.

Assim, insertionSort sempre ordena uma lista sobre a qual é aplicado.

$$C.Q.D$$

5 Conclusão

Vemos com este exercício a grande utilidade da prova por indução. Como também as grandes dificuldades apresentadas ao tentar utilizar a lógica matematica corretamente, por exemplo, ao tentar aplicar a dedução natural na seção 2. De todo modo, como observado na seção 3 a indução é muito poderosa em casos reais, em que queremos descobrir se um algoritmo é correto. Vemos também, que a necessidade de programas que auxiliam na construção de provas, como por exemplo, o pvs, é grande devido a maior velocidade para executar os pasos da prova, e em verificar a recursão dos algoritmos definidos aqui em pseudo-código. De qualquer forma, e com este trabalho, podemos ter um pouco do gosto do grande mundo que são as abstrações na computação: Estivemos passeando entre diferentes abstrações computacionais e matemáticas, na tentativa de comprovar que, uma máquina, ao executar um algoritmo, irá retornar sempre o resultado esperado aos olhos humanos. Essa é uma das utilidades das abstrações, mas sabemos que a própria ciência, ou mesmo a computação, e os avanços científicos, são construídos por meio de abstrações que modelam adequadamente a realidade. Deste modo, podemos dizer que foi um exercício de extrema valia para a equipe, os conhecimentos aqui praticados e esperamos que tenha sido uma ótima leitura, para o leitor.