Projeto de Lógica Computacional 1

A Correção da Ordenação por Borbulhamento no Rocq

Sofia Dy La Fuente Monteiro, 211055530

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0182 – Lógica Computacional 1

Resumo: Este projeto consiste na formalização e verificação do algoritmo de ordenação bubble sort utilizando o assistente de provas Coq. O objetivo é demonstrar matematicamente a correção do algoritmo, garantindo que, para qualquer lista de números naturais, o algoritmo produz uma lista que é uma permutação da original (ou seja, contém os mesmos elementos) e que está ordenada em ordem não decrescente.

Palavras-chave: Coq, bubble sort, verificação formal, algoritmos de ordenação, propriedades de ordenação, permutação, listas, lemas, provas.

1. Introdução

O Bubble Sort (ou ordenação por borbulhamento) é um algoritmo de ordenação simples que organiza uma lista comparando pares de elementos vizinhos e trocando suas posições sempre que estão fora da ordem desejada. Esse processo é repetido várias vezes sobre a lista até que nenhuma troca seja necessária, indicando que os elementos estão totalmente ordenados. A cada passagem, os maiores valores "borbulham" para o final da lista, o que dá nome ao algoritmo.

Este projeto tem como objetivo empregar conceitos da Lógica de Primeira Ordem (LPO) na resolução de problemas computacionais, por meio da formalização completa do algoritmo Bubble Sort utilizando o assistente de provas Rocq (Coq). A proposta consiste não apenas em implementar o algoritmo, mas também em demonstrar matematicamente suas principais propriedades de corretude.

O Bubble Sort é baseado na função recursiva bubble l, responsável por comparar elementos consecutivos de uma lista e efetuar trocas sempre que o primeiro elemento não é menor ou igual ao segundo. A função bs l aplica repetidamente essa etapa de "borbulhamento", construindo a versão final da lista. O objetivo central do trabalho é demonstrar que bs l produz uma lista ordenada que é também uma permutação da lista original l, garantindo que o algoritmo ordene corretamente os elementos sem alterar o conjunto de valores presentes.

2. Estrutura do projeto

O projeto foi desenvolvido utilizando o Rocq (Coq), um assistente de provas que possibilita a escrita de programas formais, a especificação de propriedades matemáticas e a construção de provas verificadas por computador. É uma ferramenta voltada para a formalização de raciocínios matemáticos e computacionais, garantindo que cada passo das demonstrações seja validado de forma rigorosa pelo próprio sistema.

Todos os arquivos do projeto além das instruções de compilação estão armazenados no repositório do Github.

3. Implementação do Projeto

Para a implementação do projeto foi escolhido a proposta 2, que consiste na formalização e verificação do algoritmo Bubble Sort. Essa proposta foi escolhida porque o bubble sort é um dos métodos de ordenação mais simples, embora não seja o algoritmo mais eficiente.

O objetivo principal do projeto é demonstrar, por meio do assistente de provas Rocq, que o algoritmo bs (Bubble Sort) satisfaz as seguintes propriedades fundamentais:

- Ordenação (Sorted): o resultado produzido por bs l encontra-se em ordem crescente;
- Permutação (Permutation): a lista resultante contém exatamente os mesmos elementos da lista original l, apenas reorganizados, sem perda ou duplicação de valores.

Em outras palavras, o projeto busca provar o seguinte teorema:

```
Theorem bubble_sort_correct: forall l,
                                Sorted le (bs l) /\
                                Permutation (bs l) l.
```

3.1 Function bubble

Esta função realiza uma única passada de "borbulhamento", onde é comparado os dois primeiros elementos (x e y). Se x <= y, mantém o x na frente e continua processando o resto (y::l). Se x > y, ela inverte (y vem para frente) e continua tentando empurrar o x para baixo. O objetivo dessa primeira etapa de recursão é empurrar o maior elemento para a cauda da lista.

3.2 Fixpoint bs

Este é o algoritmo de ordenação principal. Ele funciona de forma recursiva. Para ordenar uma lista h::l, primeiro ele ordena a cauda l (chamando bs l), e depois usa a função bubble para inserir o elemento h na lista já ordenada.

3.3 Lemma bubble_perm:

Este lema estabelece que, para qualquer lista l, a lista resultante da aplicação da função bubble (borbulhamento) é uma permutação de *l*, ou seja, o algoritmo de borbulhamento apenas reorganiza a posição dos elementos, sem inserir ou remover valores da lista original. A prova foi construída por indução, onde possuímos 4 casos principais:

Casos base:

- Quando a lista é vazia;
- Quando a lista possui apenas um elemento;

Em ambos os casos, a lista ordenada é igual a lista original.

Casos recursivos:

Quando a lista possui pelo menos dois elementos, podemos ter os seguintes casos:

1. Quando o primeiro elemento é menor ou igual ao segundo ($x \leq y$): Quando isso ocorre, não há permutação entre os elementos. Dessa forma podemos aplicar diretamente a hipótese de indução sobre a chamada recursiva do borbulhamento na cauda da lista.

2. Quando o primeiro elemento é maior que o segundo ($x > y$): Quando isso ocorre é necessário permutar os dois elementos. Nesse caso, é utilizado o lema *perm_swap*, que garante que a troca de dois elementos consecutivos configura uma permutação válida. Depois aplicamos a hipótese de indução, concluindo a prova.

Esse lema foi construído com base na prova de permutação apresentada em sala para o algoritmo de MergeSort [1]. O resultado obtido garante que as operações de permutação feitas pelo borbulhamento não alteram os elementos da lista, apenas sua ordenação.

3.4 Lemma bs_perm:

Esse lema demonstra que o resultado final do algoritmo de ordenação por borbulhamento preserva todos os elementos da lista original. Ele afirma que, para qualquer lista l, a lista obtida pela aplicação da função bs l é uma permutação de l. Ou seja, esse lema garante que todos os elementos da lista original estarão na lista ordenada, sem que nenhum elemento seja acrescentado ou descartado. Esse lema garante que a repetição dessas aplicações ao longo do processo de ordenação também preserva totalmente a lista original. A prova foi construída por indução:

Caso base:

Quando l é a lista vazia, o resultado de *bs* também é a lista vazia.

Passo indutivo:

Considerando l = (h :: l'), a definição de *bs* aplica a função bubble à lista composta por (h :: bs l').

Pela hipótese de indução, sabemos que *bs* l' é uma permutação de l'. Em seguida, utilizamos o lema *bubble_perm*, que garante que a função preserva os elementos da lista. Combinando essas duas propriedades por meio da transitividade da relação de permutação concluímos que *bs* (h :: l') é uma permutação de (h :: l'), ou seja, da lista original.

3.5 Definition le_all (x:nat) (l:list nat)

Essa definição introduz o primeiro conceito que caracteriza a ordenação de uma lista: elemento x é menor ou igual a todos os elementos contidos na lista l. Essa definição é importante porque a partir dela é possível provar outras propriedades de ordenação.

3.6 Lemma le_all_perm

O primeiro lema da ordenação garante que se x é menor que todos os elementos em l, e l é uma permutação de l', então x também é menor que todos em l'.

3.7 Lemma sorted_imp_le_all

Este lema afirma que, se uma lista iniciada por um elemento x está ordenada le (x :: l), então x é menor ou igual a todos os elementos da cauda da lista l. A prova é feita por indução.

Caso Base:

Ocorre quando a lista é vazia;

Passo indutivo:

Usamos a definição de ordenação *Sorted le*, que garante que $x \leq a$ para o primeiro elemento a da cauda, e que a cauda restante também está ordenada. Pela transitividade, podemos provar que x é menor ou igual a todos os elementos de l.

3.8 Lemma le_all_sorted

Este lema é o inverso do lema anterior. Ele afirma que, se uma lista l já está ordenada e um elemento x é menor ou igual a todos os elementos de l, então é possível inserir x no início da lista sem comprometer a ordenação.

A prova utiliza diretamente a definição indutiva de Sorted. Primeiramente, infere-se que a cauda da lista está ordenada. Depois é provado que x é menor ou igual ao primeiro elemento de l, ou seja, x corresponde a cabeça da lista o que nos permite concluir que a lista resultante também é ordenada.

3.9 bs_sorted

Esse lema tem como objetivo demonstrar que o algoritmo bubble sort produz efetivamente uma lista ordenada. Ele afirma que, para qualquer lista l, vale a propriedade Sorted le (bs l).

A prova deste lema se mostrou um pouco complexa principalemente a prova de que o bubble sort preserva a ordenação em cada passo recursivo. A demonstração do bs_sorted não pode ser concluída, o que deixou a prova do algoritmo incompleta.

3.10 Lemma sorted_tl

Esse lema é importante porque garante que se a lista inteira está ordenada, retirar o primeiro elemento continua mantendo a lista ordenada. Este lema garante que a ordenação de uma lista permanece mesmo se considerarmos apenas a sua cauda. Ele afirma que, se uma lista do tipo (h :: l) está ordenada segundo a relação então a sub lista formada pela remoção do primeiro elemento, também está ordenada.

3.11 O Teorema Final

O teorema final prova a corretude do algoritmo Bubble Sort. Ele afirma que, para qualquer lista l, o resultado da aplicação da função bs l satisfaz simultaneamente duas propriedades fundamentais:

- Ordenação (Sorted): a lista resultante encontra-se ordenada em ordem crescente segundo a relação *le*;
- Permutação (Permutation): a lista ordenada contém exatamente os mesmos elementos da lista original, possivelmente em outra ordem, garantindo que nenhum elemento foi perdido ou adicionado durante o processo de ordenação.

Infelizmente, esse teorema não pode ser totalmente provado. A propriedade de permutação foi demonstrada por meio do lema *bs_perm*, mas propriedade de ordenação depende do lema *bs_sorted*, que não pode ser provado.

Dessa forma, a corretude completa do Bubble Sort permaneceu incompleta, já que somente a preservação dos elementos pode ser provada, mas não há garantia formal da ordenação do resultado.

4. Conclusão

Este projeto teve como objetivo principal a verificação formal da correção do algoritmo bubble sort utilizando o assistente de provas Coq. Ao longo do trabalho foi atingirdo parcialmente os objetivos: Foi demonstrado que o algoritmo preserva todos os elementos da lista original através de propriedades de permutação, provando que bubble e bs são efetivamente permutações da entrada.

Embora a prova completa da ordenação não tenha sido finalizada devido a dificuldades na demonstração de um dos lemas principais, o desenvolvimento do trabalho proporcionou um aprendizado significativo sobre lógica formal, estruturação de provas e uso do Rocq como ferramenta de verificação.

Referências:

[1] MOURA, Flávio. *Implementação e prova do Merge Sort em Rocq*. Arquivo lc1_2025_2_mergesort.v. Material didático da disciplina Lógica Computacional I, 2025. Disponível em: https://flaviomoura.info/files/lc1_2025_2_mergesort.v. Acesso em: dez. 2025.

[2] MOURA, Flávio. Lógica e Computação – Apostila de Introdução à Lógica Clássica e Algoritmos (LCA). Disponível em: https://flaviomoura.info/files/lca.pdf Acesso em: dez. 2025.

[3] M. Ayala-Rincón and F. L. C. de Moura. Applied Logic for Computer Scientists – Computational Deduction and Formal Proofs. UTCS. Springer, 2017.

[4] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004.

[5] F. S. C. da Silva, A. C. V. de Melo, and M. Finger. *Lógica Para Computação*. THOMSON PIONEIRA, 2006.