A Correção do Algoritmo de Ordenação por Inserção

Alan dos Santos Dias - 232007830
Bruno Henrique Duarte - 221022239
João Marcos Rodrigo Cardoso - 232027411

23 de novembro de 2025

1 Introdução

Este trabalho apresenta uma prova formal da correção do algoritmo de ordenação por inserção . A formalização foi desenvolvida no assistente de provas Coq. O assistente de provas Coq utiliza o sistema de Dedução Natural, o que o torna adequado para o desenvolvimento de atividades computacionais na disciplina de Lógica Computacional 1. Conforme discutido em aula, provas matemáticas realizadas apenas em papel estão suscetíveis a erros humanos, ambiguidades e saltos lógicos. O uso de uma ferramenta formal como o Coq é fundamental para mitigar esses riscos, garantindo o rigor e a correção mecânica de cada passo da demonstração.

Observação sobre o ambiente de prova: A formalização deste projeto foi realizada utilizando a plataforma online jsCoq (disponível em https://jscoq.github.io/scratchpad.html). As versões específicas utilizadas foram o jsCoq 0.12.3, executando sobre o núcleo do Coq versão 8.12.2 (build 81200).

2 Definição dos Algoritmos

O algoritmo de ordenação por inserção é composto por duas funções principais. A primeira é a função auxiliar *insert*, que insere um número natural em uma lista que já está ordenada, mantendo a ordenação.

```
Fixpoint insert (x:nat) l :=
  match l with
  | [] ⇒ [x]
  | h::tl ⇒ if (x <=? h)
              then x::l
              else h::(insert x tl)
  end.
```

A função principal *insertion_sort* percorre a lista de entrada recursivamente, inserindo cada elemento na cauda já ordenada.

```
Fixpoint insertion_sort (l: list nat) :=
  match l with
  | [] ⇒ []
  | h::tl ⇒ insert h (insertion_sort tl)
  end.
```

3 Propriedades Auxiliares

Para provar a correção total do algoritmo, precisamos estabelecer duas propriedades fundamentais sobre a função de inserção: 1. Ela preserva a ordenação dos elementos. 2. Ela preserva o conjunto de elementos (é uma permutação).

3.1 Preservação da Ordenação

O lema a seguir garante que, se inserirmos um elemento x em uma lista l que já está ordenada ($Sorted$), a lista resultante também estará ordenada. A prova é feita por indução estrutural na lista l e análise de casos sobre a comparação entre x e a cabeça da lista.

`Lemma` $insertPreservesSorted$:
$\forall\ x\ l,\ Sorted\ le\ l \rightarrow Sorted\ le\ (insert\ x\ l).$

A prova é realizada por indução na evidência de que a lista l já está ordenada ($H : Sorted\ le\ l$).

- Caso Base (Lista Vazia): A inserção de x em uma lista vazia resulta na lista unitária $[x]$, que é trivialmente ordenada.
- Passo Indutivo: Supondo uma lista da forma $a :: tl$ onde a cauda tl também é ordenada, temos dois cenários baseados na comparação entre x e a cabeça a:
  - Se $x \leq a$: O elemento x torna-se a nova cabeça da lista ($x :: a :: tl$). Como $x \leq a$ e o restante da lista já estava ordenado, a propriedade é preservada.
  - Se $x > a$: O algoritmo insere x recursivamente na cauda tl. Pela hipótese de indução, sabemos que essa inserção recursiva gera uma lista ordenada. Resta apenas provar que a cabeça original a preserva a ordem em relação à nova lista gerada, o que é garantido pois $a < x$ e a já era menor ou igual a todos os elementos de tl.

3.2 Preservação da Permutação

Além da ordenação, é necessário garantir que a operação de inserção não duplica nem remove elementos indevidamente. O lema abaixo estabelece que a lista resultante de $insert\ x\ l$ é uma permutação da lista $x :: l$. `Lemma` $insert\_perm : \forall\ x\ l,\ Permutation\ (x :: l)\ (insert\ x\ l).$

4 Teorema Principal

`Theorem` $insertion\_sort\_correct: \forall\ l,\ Sorted\ le\ (insertion\_sort\ l) \land Permutation\ (insertion\_sort\ l)\ l.$

5 Conclusão

Coloque aqui a conclusão do relatório.

Referências