Projeto de Lógica Computacional 1 (2025/2)

Flávio L. C. de Moura

28 de outubro de 2025

O presente projeto visa utilizar a Lógica de Primeira Ordem (LPO) para resolver problemas computacionais não triviais. Ao longo das últimas aulas estamos construindo a formalização completa do algoritmo *mergesort* no assistente de provas Rocq. A seguir apresentaremos diversas propostas para o projeto. Cada grupo deve selecionar uma proposta.

1 A correção da ordenação por inserção no Rocq

O algoritmo *Insertion Sort* (ordenação por inserção) é baseado na função recursiva `insert x l` que insere o elemento `x` na lista ordenada `l` como a seguir:

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

Observe que uma das propriedades a serem provadas nesta proposta consiste em mostrar que `insert x l` retorna uma lista ordenada, se `l` estiver ordenada. A função principal, chamada `insertion_sort l`, é definida como a seguir:

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

O objetivo desta proposta é mostrar que o algoritmo `insertion_sort l` retorna uma permutação ordenada da lista `l`:

```
Theorem insertion_sort_correct: forall l,
                                Sorted le (insertion_sort l) /\
                                Permutation (insertion_sort l) l.
```

Pense em como a propriedade a ser provada para a função `insert` (ver parágrafo anterior) poderia ajudar na prova do teorema `insertion_sort_correct`. Uma parte importante deste projeto consiste em enunciar e provar resultados auxiliares para simplificar as provas dos resultados principais (estratégia de divisão e conquista).

Link para o repositório:
`https://github.com/flaviodemoura/insertion_sort`

2 A correção da ordenação por borbulhamento no Rocq

O algoritmo *Bubble Sort* (ordenação por borbulhamento) é baseado na função recursiva `bubble l` que compara elementos consecutivos de uma lista e, recursivamente, troca suas posições quando o primeiro destes elementos não é menor ou igual a segundo:

```
Function bubble (l: list nat ) {measure length l} :=
 match l with
 | nil => nil
 | x::nil => x::nil
 | x::(y::l) =>
     if x <=? y
     then x::(bubble (y::l))
           else y::(bubble (x::l))
           end.
```

A função principal, chamada `bs l`, é definida como a seguir:

```
Fixpoint bs (l: list nat) :=
 match l with
 | nil => nil
 | h::l => bubble (h::(bs l))
 end.
```

O objetivo desta proposta é mostrar que o algoritmo `bs l` retorna uma permutação ordenada da lista `l`:

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

Uma parte importante deste projeto consiste em enunciar e provar resultados auxiliares para simplificar as provas dos resultados principais (estratégia de divisão e conquista).

Link para o repositório:
`https://github.com/flaviodemoura/bubble_sort`

3 A correção do algoritmo de ordenação por seleção

O algoritmo de ordenação por seleção (*Selection Sort*) é baseado na função recursiva `select\_min l` que retorna o menor elemento da lista `l`, caso ele exista:

```
Function select_min (l : list nat) {measure length l} : option nat :=
 match l with
 | nil => None
 | h::nil => Some h
 | h1::h2::tl => if h1 <=? h2
                 then select_min (h1::tl)
                 else select_min (h2::tl)
 end.
```

A função principal, chamada `ssort l`, é definida como a seguir:

```
Fixpoint ssort (l: list nat) :=
 match l with
 | nil => nil
 | h::tl => match select_min l with
            | None => nil
            | Some m => m::(ssort tl)
            end
 end.
```

O objetivo desta proposta é mostrar que o algoritmo `ssort l` retorna uma permutação ordenada da lista `l`:

```
Theorem selection_sort_correct: forall l,
                                Sorted le (ssort l) /\
                                Permutation (ssort l) l.
```

Uma parte importante deste projeto consiste em enunciar e provar resultados auxiliares para simplificar as provas dos resultados principais (estratégia de divisão e conquista).

Link para o repositório:
`https://github.com/flaviodemoura/selection_sort`

4 Equivalência entre diferentes noções de indução

Indução matemática é uma técnica de prova muito poderosa que desempenha um papel fundamental tanto em Matemática quanto em Computação. Se $P(n)$

denota uma propriedade dos números naturais $\mathbb{N} = \{0, 1, 2, \ldots\}$ então o princípio da indução matemática (PIM) é dado por:

$$\frac{P\ 0 \qquad \forall k, P\ k \Longrightarrow P\ (k+1)}{\forall n, P\ n}\ (PIM)$$

Na descrição acima, chamamos $P\ 0$ de *base da indução* e $\forall k, P\ k \Longrightarrow P\ (k{+}1)$ de *passo indutivo*. No passo indutivo, $P\ k$ é a *hipótese de indução*.

Observe que o passo indutivo é a parte interessante de qualquer prova por indução. A base da indução consiste apenas na verificação de que a propriedade vale para uma situação particular.

Em Rocq, podemos visualizar o PIM via o comando `Print nat_ind`:

```
forall P : nat -> Prop, P 0 ->
                        (forall n : nat, P n -> P (S n)) ->
                        forall n : nat, P n
```

Uma variação do PIM bastante útil é conhecida como *Princípio da Indução Forte (PIF)*:

$$\frac{\forall k, (\forall m, m < k \Longrightarrow P\ m) \Longrightarrow P\ k}{\forall n, P\ n}\ (PIF)$$

O objetivo desta proposta é mostrar que o PIM e o PIF são equivalentes.

```
 Definition PIM :=
  forall P: nat -> Prop, (P 0) ->
                         (forall k, P k -> P (S k)) ->
                         forall n, P n.

Definition PIF :=
  forall Q: nat -> Prop, (forall k, (forall m, m<k -> Q m) -> Q k) ->
                         forall n, Q n.

Theorem PIM_equiv_PIF: PIM <-> PIF.
```

Link para o repositório:
`https://github.com/flaviodemoura/ind-equiv`

5 Tema livre

Alternativamente, os grupos podem trabalhar com outros algoritmos e/ou teorias desde que não sejam excessivamente simples. Consulte o professor para que a escolha seja aprovada, e uma estratégia de prova seja construída.

6 Etapas do Projeto

- O prazo para finalização do projeto é o dia *[2025-12-07 dom]*.
- O repositório deve conter:
    1. O arquivo `relatorio.pdf` contendo o nome e matrícula dos participantes, assim como todas as informações pertinentes ao projeto, como a estruturação dos arquivos de prova, a estruturação das provas (em linguagem natural, evite colar código no relatório) e as explicações detalhadas de como foi o desenvolvimento do projeto.
    2. Os arquivos com as provas como descrito no relatório.
- Os grupos que tiverem interesse podem agendar uma apresentação para o dia *[2025-12-08 seg]*.
- O agendamento deve ser feito até o final do dia *[2025-12-07 dom]* via email (flaviomoura@unb.br) ou mensagem do Teams.