Lógica Computacional 1
Equivalência Entre Diferentes Noções de Permutação

Oseias Romeiro Magalhães / 211036123

4 de setembro de 2022

1 Introdução

Neste projeto temos como objetivo provar a equivalência entre duas noções distintas de Permutação. A permutação é uma técnica de contagem utilizada para determinar quantas maneiras existem para ordenar os elementos de um conjunto finito. Neste projeto, será utilizado listas e provaremos se elas são permutações uma da outra, segundo duas definições:

- Contagem de ocorrências de cada elemento e verificar que todo elemento ocorre o mesmo número de vezes nas duas listas, para que assim, uma seja uma permutação da outra;
- Outra forma é utilizando uma estrutura indutiva, raciocínio da qual partimos de propriedades conhecidas para chegarmos a uma conclusão.

2 Materiais

Foi utilizado para o desenvolvimento do projeto o jsCoq [2] como assistente de provas online, utilizando a linguagem Coq [1] para escrever as provas. Além da utilização do *Git* e do *GitHub* para hospedagem e versionamento o projeto.

Além disso, também foi utilizado para fontes de conhecimento o *Software Foundation* [4] para trabalhar com o jsCoq e a linguagem Coq, além da utilização do conteúdo da disciplina de Lógica Computacional 1 [3].

3 Descrição

3.1 Definições

Construímos as Permutações como mencionadas na introdução [1], da seguinte forma:

```
Fixpoint num_oc (x: ℕ) (l: list ℕ): ℕ :=
  match l with
  | nil ⇒ 0
  | h::tl ⇒ if (x =? h) then S (num_oc x tl) else num_oc x tl
end.

Definition equiv l l' := ∀ x, num_oc x l = num_oc x l'.
```

```
Inductive perm: list ℕ → list ℕ → ℙ :=
| perm_eq: ∀ l, perm l l
| perm_swap: ∀ x y l, perm (x :: y :: l) (y :: x :: l)
| perm_hd: ∀ x l l', perm l l' → perm (x :: l) (x :: l')
| perm_trans: ∀ l l' l'', perm l l' → perm l' l'' → perm l' l'' → perm l l''.

Lemma perm_equiv_Permutation: ∀ l₁ l₂, perm l₁ l₂ ↔ Permutation l₁ l₂.
```

3.2 Provas

3.2.1 Parte 1: perm_equiv_Permutation

Nessa primeira parte provamos a equivalência da definição *perm* com a *Permutation* da biblioteca do Coq (que também segue uma estrutura indutiva), com as listas l1 e l2.

```
Lemma perm_equiv_Permutation: ∀ l₁ l₂, perm l₁ l₂ ↔ Permutation l₁ l₂.
```

Estrutura indutiva de *Permutation*:

```
Permutation (A : Type) : list A -> list A -> Prop :=
  perm_nil : Permutation nil nil
| perm_skip : forall (x : A) (l l' : list A),
Permutation l l' -> Permutation (x :: l) (x :: l')
| perm_swap : forall (x y : A) (l : list A),
Permutation (y :: x :: l) (x :: y :: l)
| perm_trans : forall l l' l'' : list A,
Permutation l l' -> Permutation l' l'' -> Permutation l l''
```

Para mostrar a equivalência utiliza-se uma bi-implicação e demonstramos ambos lados da implicação. Em cada lado, assume o antecedente e aplica a indução na hipótese.

Assim temos quatro sub-provas, que para prova-las, basta utilizar as regras de cada estrutura indutiva. Logo:

- *perm l1 l2* $\rightarrow$ *Permutation l1 l2*

  Se duas listas l1 e l2 são permutações uma da outra pela definição *perm*, então o mesmo também é válido pela definição de *Permutation*.

- *Permutation l1 l2* $\rightarrow$ *perm l1 l2*

  Se duas listas l1 e l2 são permutações uma da outra pela definição *Permutation*, então o mesmo também é válido pela definição de *perm*.

3.2.2 Parte 2: perm_equiv

Aqui provamos o Teorema em que *perm* e *equiv* são definições equivalentes. Quaisquer listas que estejam relacionadas por *equiv* estão, necessariamente, relacionadas por *perm*. Ou seja, temos que provar uma dupla implicação. Segue-se o teorema:

```
Theorem perm_equiv: ∀ l l', equiv l l' ↔ perm l l'.
```

- *equiv* l l' $\rightarrow$ *perm* l l'

  não concluído

- *equiv* l l' $\rightarrow$ *perm* l l'

  Já nesse lado, assumimos o antecedente e é feito a indução na hipótese. Gerando assim, quatro ramos para serem provados, sendo eles equivalentes as regras da estrutura de indução de *perm* mostrado em 3.1, porém agora utilizando a definição *equiv*.

  Para isso, foram construídos alguns lemas: *equiv_nil*, *equiv_hd* e *equiv_trans*. Enquanto outros passos foram provados diretamente dentro da prova principal.

4 Conclusão

Com isso, mostramos que as duas noções de permutação, uma com estrutura indutiva e outra por número de ocorrências de elementos, são equivalentes. Podendo ambas ser utilizadas para verificar se duas listas são permutações uma da outra. Além de provarmos a equivalência com uma estrutura presente na própria biblioteca Coq. Assim, foi desenvolvido e aprimorado os conhecimentos com software de assistência de provas e construções de provas em lógica de primeira ordem.

Referências

[1] Coq. *Coq Proof Assistant*. `https://coq.inria.fr/`.

[2] jsCoq. *Online Coq Proof Assistant*. `https://coq.vercel.app/`.

[3] Falvio Moura. *Disciplina de Lógica Computacional 1*. `http://flaviomoura.info/lc1-2022-1.html`.

[4] Benjamin C. Pierce et al. *Software Foundations*. `https://softwarefoundations.cis.upenn.edu/`.