A Lógica na solução de problemas computacionais

Uma prova formal da correção do algoritmo de ordenação por inserção

Nesta seção definiremos o algoritmo de ordenação por inserção recursivamente. A estrutura de dados utilizada é a de listas ligadas que possui dois construtores: `nil` para representar a lista vazia, e `::` que nos permite construir uma nova lista a partir de um número natural e de uma lista dados. Assim, a lista unitária contendo apenas o natural 5 é representada por `5 :: nil`, enquanto que a lista `1 :: (5 :: nil)`, ou simplesmente `1 :: 5 :: nil`, representa a lista que tem 1 como primeiro elemento, e a lista `5 :: nil` como cauda. A definição da função `ord_insercao` abaixo foi feita no assistente de provas Coq, ferramenta que utilizaremos para o desenvolvimento do projeto do curso. A definição é iniciada com a palavra reservada `Fixpoint` utilizada para definir funções recursivas:

```
Fixpoint ord_insercao l :=
 match l with
   | nil => nil
   | h :: tl => insere h (ord_insercao tl)
 end.
```

A função recursiva `ord_insercao` recebe como argumento uma `l`. No corpo da função, analisamos as duas situações possíveis para a lista `l`:

1. No primeiro caso, `l` é a lista vazia, e portanto a função `ord_insercao` retorna a lista vazia já que não há nada a ser ordenado;

2. No segundo caso, a lista `l` é da forma `h :: tl`, isto é, `l` tem `h` como primeiro elemento e `tl` como cauda (o restante dos elementos). Neste caso, a ordenação é feita inserindo `h` na versão ordenada de `tl`.

A função `insere` que aparece na definição de `ord_insercao` é definida recursivamente como a seguir:

```
Fixpoint insere (n:nat) (l: list nat) :=
  match l with
  | nil => n :: nil
  | h :: tl => if n <=? h then (n :: l)
     else (h :: (insere n tl))
end.
```

A função `insere` recebe como argumento um número natural `n` e uma lista `l`, e retorna a lista obtida após inserir `n` na lista `l`. Novamente, a definição é feita baseada na estrutura da lista `l`:

1. Se `l` for a lista vazia, então a lista unitária `n :: nil`;

2. Se `l` for da forma `h :: tl`, então `n` é comparado `h`. Se `n` for menor ou igual a `h` então a lista `(n :: l)` é retornada. Quando `n` é maior do que `h` então `n` é inserido via uma chamada recursiva da função `insere` na cauda `tl`.

O exercício da aula passada foi provar o teorema abaixo usando indução na estrutura da lista `l`:

Teorema 1. *Prove que, se `l` é uma lista ordenada então `insere n l` é também uma lista ordenada.*

Agora, gostaria de propor uma outra reflexão sobre uma possível prova alternativa para o teorema acima. Observe a noção de ordenação com que estamos trabalhando:

```
Inductive ordenada: list nat -> Prop :=
| ord_nil: ordenada nil
| ord_one: forall x, ordenada (x :: nil)
| ord_all: forall x y l, x <= y -> ordenada (y :: l)  -> ordenada (x :: y :: l).
```

Já sabemos que cada construtor desta definição indutiva corresponde a uma das regras a seguir:

$$\frac{}{\texttt{ordenada nil}}\ (\texttt{ord\_nil}) \qquad \frac{}{\texttt{ordenada}(\texttt{x} :: \texttt{nil})}\ (\texttt{ord\_one})$$

$$\frac{\texttt{x} \leq \texttt{y} \qquad \texttt{ordenada}(\texttt{y} :: \texttt{l})}{\texttt{ordenada}(\texttt{x} :: \texttt{y} :: \texttt{l})}\ (\texttt{ord\_all})$$

Então a hipótese do Teorema 1 é feita sobre uma definição indutiva, e portanto podemos fazer uma outra prova por indução na hipótese de que "a lista `l` está ordenada". Como preparação para a aula de hoje faça um esboço desta prova.