2.3 O algoritmo de ordenação por inserção - versão recursiva

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`.

É importante notar a forma como a função `insere` é definida. A inserção do natural `n` na lista `l` não é feita de qualquer forma, mas de forma a preservar a ordenação de `l`. Ou seja, se a lista `l` estiver ordenada então `insere n l` é também uma lista ordenada. Você concorda com esta afirmação? Para que possamos responder a esta questão de uma forma precisa, vamos iniciar apresentando uma definição de ordenação. O que significa dizer que a lista `l` está ordenada? Claramente, a lista vazia está ordenada, assim como qualquer lista unitária. Podemos representar estes dois casos por meio das seguintes regras de inferência:

$$\frac{}{sorted\ nil}\ (nil\_sorted) \qquad \frac{}{\forall n, sorted(n :: nil)}\ (one\_sorted)$$

Agora, uma lista com pelo menos dois elementos, digamos $x :: y :: l$, onde x é o primeiro elemento, y é o segundo elemento e l o restante da lista, estará ordenada se $x \leq y$ e a sublista $y :: l$ estiver ordenada:

$$\frac{sorted(y :: l) \qquad x \leq y}{sorted(x :: y :: l)}\ (all\_sorted)$$

Ou seja, para provarmos que a lista $x :: y :: l$ está ordenada, precisamos provar que $x \leq y$ e que a lista $y :: l$ também está ordenada.

Agora podemos responder de forma precisa a questão acima que enunciaremos como um teorema:

Teorema 2.3.1. *Prove que, se `l` é uma lista ordenada então `insere n l` é também uma lista ordenada.*

Prova: A prova é feita por indução na estrutura da lista `l`. Se `l` for a lista vazia então `insere n l` é uma lista unitária, que por definição está ordenada (veja a regra (*one_sorted*)). Se `l` não é a lista vazia, então `l` é da forma `h::tl` onde `h` é o primeiro elemento de `l`, e `tl` é a cauda da lista `l`. Portanto precisamos provar que `insere n (h::tl)` é uma lista ordenada. De acordo com a definição da função `insere`, temos dois casos a considerar:

1. `n` $\leq$ `h`: Neste caso, `insere n (h::tl)` = `n::h::tl`, e portanto precisamos mostrar que a lista `n::h::tl` está ordenada. Para isto utilizaremos a regra (*all_sorted*), que reduz esta prova a duas subprovas mais simples: a primeira consiste em mostrar que `n` $\leq$ `h`, o que é direto do caso que estamos considerando. A segunda subprova, consiste em provar que a lista `h::tl` está ordenada, mas esta é a hipótese do problema, uma vez que `l` = `h::tl`.

A prova feita até este momento pode ser representada por meio da seguinte árvore de dedução:

$$\dfrac{\dfrac{(one\_sorted)\ \dfrac{}{sorted(n :: nil)}}{(\text{DEF})\ \dfrac{}{sorted(insere\ n\ nil)}} \quad (n \leq h)\ \dfrac{(\text{HIP})\ \dfrac{}{n \leq h} \qquad \dfrac{}{sorted\ l}\ (\text{HIP})}{sorted(n :: h :: tl)} \qquad \dfrac{??}{sorted(h :: (insere\ n\ tl))}\ (n > h)}{sorted(insere\ n\ (h :: tl))}\ (\text{DEF})}{sorted(insere\ n\ l)}\ \text{INDUÇÃO EM } l$$

Como provar que a lista $h :: (insere\ n\ tl)$ está ordenada? O exercício de hoje consiste em responder esta pergunta. Ou seja, complete a prova acima detalhando todos os passos.