2.3.1 Prova formal (mecânica) da correção do algoritmo de ordenação por inserção recursivo

Nesta seção faremos uma prova mecânica da correção do algoritmo de ordenação por inserção recursivo. Nosso objetivo é provar uma propriedade idêntica à invariante apresentada na prova da correção do algoritmo de ordenação por inserção iterativo. Ou seja, queremos provar que a função `ord_insercao` retorna uma lista ordenada e que a lista de saída é uma permutação da lista de entrada. Dividiremos esta tarefa em dois passos:

1. `ord_insercao` sempre retorna uma lista ordenada;
2. `ord_insercao` retorna uma permutação da lista de entrada.

A definição de ordenação foi apresentada por meio da regras de inferência abaixo:

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

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

A definição correspondente em Coq é dada por:

```
Inductive sorted: list nat -> Prop :=
| nil_sorted: sorted nil
| one_sorted: forall x, sorted (x :: nil)
| all_sorted: forall x y l, x <= y -> sorted (y :: l) ->
      sorted (x :: y :: l).
```

Queremos então, provar o seguinte lema:

Lema 2.3.3. *A função* `ord_insercao` *retorna uma lista ordenada como saída ao receber uma lista qualquer como entrada.*

Uma maneira mais precisa de apresentar este lema consiste em utilizar a linguagem da lógica de primeira ordem:

Lema 2.3.4. $\forall l, ordenada$ (`ord_insercao` l).

Este lema pode ser provado por indução na estrutura da lista l, e os detalhes são deixados como exercício.

O segundo passo da prova da correção consiste em mostrar que, ao receber a lista l como argumento, o resultado de aplicar a função `ord_insercao` à lista l é uma permutação de l. Em outras palavras, a lista de saída possui os mesmos elementos da lista de entrada. Formalmente, definimos indutivamente a noção de permutação em Coq da seguinte forma:

```
Inductive perm: list nat -> list nat -> Prop :=
| perm_refl: forall l, perm l l
| perm_hd: forall x l l', perm l l' -> perm (x::l) (x::l')
| perm_swap: forall x y l l', perm l l' -> perm (x::y::l) (y::x::l')
| perm_trans: forall l l' l'', perm l l' -> perm l' l'' -> perm l l''.
```

Agora `perm` é um predicado binário (recebe duas listas de números naturais como argumento) e é composto de quatro construtores. Cada um deles corresponde a uma das seguintes regras:

$$\frac{}{perm\ l\ l}\ (\texttt{perm\_refl}) \qquad \frac{perm\ l\ l'}{perm\ (x :: l)\ (x :: l')}\ (\texttt{perm\_hd})$$

$$\frac{perm\ l\ l'}{perm\ (x :: y :: l)\ (y :: x :: l')}\ (\texttt{perm\_swap})$$

$$\frac{perm\ l\ l' \qquad perm\ l'\ l''}{perm\ l\ l''}\ (\texttt{perm\_trans})$$

aqui também as variáveis x, y, l, l' e l'' são variáveis quantificadas universalmente.

A propriedade que queremos provar pode ser representada por meio do seguinte lema:

Lema 2.3.5. *A função* `ord_insercao` *retorna uma lista que é uma permutação da lista de entrada.*

Mais formalmente, podemos escrever a mesma ideia da seguinte forma:

Lema 2.3.6. $\forall l, perm\ l\ (\texttt{ord\_insercao}\ l)$.

Prova. A prova é feita por indução na estrutura da lista l. Temos então dois casos a considerar: quando l é a lista vazia, e quando l possui pelo menos um elemento.

- Quando $l = nil$, (`ord_insercao` l) corresponde a (`ord_insercao` nil) que retorna nil, de acordo com a definição de `ord_insercao`. A representação em árvore desta prova é dada a seguir, onde `def` corresponde ao passo de aplicação da definição da função `ord_insercao`.

$$\frac{\dfrac{}{perm\ nil\ nil}\ (\texttt{perm\_refl})}{perm\ nil\ (\texttt{ord\_insercao}\ nil)}\ (\texttt{def})$$

- Quando l não é a lista vazia, isto é, possui pelo menos um elemento, então $l = h :: tl$. Ou seja, l tem h como primeiro elemento e tl como cauda (o resto da lista). Queremos provar que

$$perm\ (h :: tl)\ (\texttt{ord\_insercao}\ (h :: tl))$$

Agora, podemos aplicar a definição da função `ord_insercao`:

$$\frac{perm\ (h :: tl)\ (\texttt{insere}\ h\ (\texttt{ord\_insercao}\ tl))}{perm\ (h :: tl)\ (\texttt{ord\_insercao}(h :: tl))}\ (\texttt{def})$$

Neste ponto ainda não podemos aplicar nenhuma das regras de permutação, mas observe que a lista `insere` h (`ord_insercao` tl) tem que ser uma permutação da lista $h ::$ (`ord_insercao` tl) para que a propriedade que estamos tentando provar seja verdadeira. Vamos então considerar este fato como um lema cuja prova será deixada como exercício:

Lema 2.3.7. *A lista* $(h :: tl)$ *é uma permutação da lista* `insere` h tl*.*

Em Coq este lema pode ser enunciado da seguinte forma:

```
Lemma insere_perm: forall l a, perm (a :: l) (insere a l).
```

Voltando para a nossa prova original, observe que podemos usar a transitividade da permutação para então, conseguirmos usar o lema acima:

$$\dfrac{\dfrac{perm\ (h :: tl)\ (h :: (\texttt{ord\_insercao}\ tl))\qquad (*)}{perm\ (h :: tl)\ (\texttt{insere}\ h\ (\texttt{ord\_insercao}\ tl))}\ (\texttt{perm\_trans})}{perm\ (h :: tl)\ (\texttt{ord\_insercao}(h :: tl))}\ (\texttt{def})$$

onde $(*)$ é a fórmula $perm\ (h :: (\texttt{ord\_insercao}\ tl))\ (\texttt{insere}\ h\ (\texttt{ord\_insercao}\ tl))$. Assim, a transitividade divide a prova em duas subprovas, e a subprova da esquerda pode ser feita usando a regra `perm_hd` e a hipótese de indução:

$$\dfrac{\dfrac{\dfrac{}{perm\ tl\ (\texttt{ord\_insercao}\ tl)}\ (\texttt{IH})}{perm\ (h :: tl)\ (h :: (\texttt{ord\_insercao}\ tl))}\ (\texttt{perm\_hd})\qquad (*)}{perm\ (h :: tl)\ (\texttt{insere}\ h\ (\texttt{ord\_insercao}\ tl))}\ (\texttt{perm\_trans})}{perm\ (h :: tl)\ (\texttt{ord\_insercao}(h :: tl))}\ (\texttt{def})$$

A subprova da direita, isto é, a prova de
$perm\ (h :: (\texttt{ord\_insercao}\ tl))\ (\texttt{insere}\ h\ (\texttt{ord\_insercao}\ tl))$
pode ser concluída com a ajuda do lema `insere_perm` comentado anteriormente.
□