List Concatenation

Question 1 - List Concatenation

The function $\texttt{LConcat}(\texttt{x}, \texttt{y})$, which joins two lists together, can be implemented as follows:

$\vdash \{\mathsf{list}(\texttt{x}, \alpha) \star \mathsf{list}(\texttt{y}, \beta)\}$
$\texttt{list\_concat}(\texttt{x}, \texttt{y})\ \{$
  $\{P_1\}$
  $\texttt{if}\ (\texttt{x} = \texttt{null})\ \{$
    $\texttt{x} := \texttt{y}$
  $\}\ \texttt{else}\ \{$
    $\texttt{t} := \texttt{x};$
    $\texttt{n} := [\texttt{x} + 1];$
    $\{P_2\}$
    $\texttt{ex} + \texttt{fr} + \texttt{co}$ | $\{P_3\}$
    | $\{\exists \alpha_1, \alpha_2, b\,.\, \mathsf{lseg}(\texttt{x}, \texttt{t}, \alpha_1) \star \texttt{t} \mapsto b, \texttt{n} \star \mathsf{list}(\texttt{n}, \alpha_2) \star \alpha \doteq \alpha_1 \cdot [b] \cdot \alpha_2\}$
    | $\texttt{while}\ (\texttt{n} \neq \texttt{null})\ \{$
    | $\texttt{t} := \texttt{n};$
    | $\texttt{n} := [\texttt{n} + 1];$
    | $\}$
    $[\texttt{t} + 1] := \texttt{y}$
  $\};$
  $\{P_4\}$
  $\texttt{return}\ \texttt{x}$
$\}$
$\{\mathsf{list}(\mathsf{ret}, \alpha \cdot \beta)\}$

(a) Give the assertions P_1 and P_4. Explain how they are constructed in general, given the function pre- and post-condition.

(b) Partially complete the proof sketch, reaching the assertion P_2. Give P_2.

(c) Give the assertion P_3 and explain the step from P_2 to P_3.

(d) Explain the while loop invariant in words (and heap diagrams, if you like).

(e) Explain in words why P_3 entails the loop invariant.

(f) Complete the proof sketch.

(g) List (no pun intended) the entailments related to list segments and lists mentioned in the lectures that you had to use in parts (a)-(f).

Answer:

The full proof sketch is as follows:

```
list_concat(x, y) {
  {P_1 = list(x, α) ⋆ list(y, β) ⋆ t, n ≐ null}
  if (x = null) {
    {list(x, α) ⋆ list(y, β) ⋆ t, n ≐ null ⋆ x ≐ null}
    {x ≐ null ⋆ α ≐ ϵ ⋆ list(y, β) ⋆ t, n ≐ null}
    x := y
    {x ≐ y ⋆ α ≐ ϵ ⋆ list(y, β) ⋆ t, n ≐ null}
    {list(x, α · β) ⋆ t, n ≐ null}
    {list(x, α · β)}
  } else {
    {list(x, α) ⋆ list(y, β) ⋆ t, n ≐ null ⋆ x ≠̇ null}
    {∃a, z, α'. x ↦ a, z ⋆ list(z, α') ⋆ α ≐ a:α' ⋆ list(y, β) ⋆ t, n ≐ null}
    t := x;
    {∃a, z, α'. x ↦ a, z ⋆ list(z, α') ⋆ α ≐ a:α' ⋆ list(y, β) ⋆ t ≐ x ⋆ n ≐ null}
    n := [x + 1];
    {P_2 = ∃a, α'. x ↦ a, n ⋆ list(n, α') ⋆ α ≐ a:α' ⋆ list(y, β) ⋆ t ≐ x}
    ex + fr + co
      // Isolate the resource of the loop
      {P_3 = t ↦ a, n ⋆ list(n, α') ⋆ α ≐ a:α' ⋆ t ≐ x}
      // Establish loop invariant
      {∃α_1, α_2, b . lseg(x, t, α_1) ⋆ t ↦ b, n ⋆ list(n, α_2) ⋆ α ≐ α_1 · [b] · α_2}
      while (n ≠ null) {
        {∃α_1, α_2, b . lseg(x, t, α_1) ⋆ t ↦ b, n ⋆ list(n, α_2) ⋆ α ≐ α_1 · [b] · α_2 ⋆ n ≠̇ null}
        t := n;
        {∃α_1, α_2, b, t. lseg(x, t, α_1) ⋆ t ↦ b, t ⋆ list(n, α_2) ⋆ α ≐ α_1 · [b] · α_2 ⋆ t ≐ n ⋆ n ≠̇ null}
        // append node to list segment
        {∃α_1, α_2, b. lseg(x, t, α_1 · [b]) ⋆ list(n, α_2) ⋆ α ≐ (α_1 · [b]) · α_2 ⋆ t ≐ n ⋆ n ≠̇ null}
        // Rename α_1 to fit invariant
        {∃α_1, α_2. lseg(x, t, α_1) ⋆ list(n, α_2) ⋆ α ≐ α_1 · α_2 ⋆ t ≐ n ⋆ n ≠̇ null}
        // Unfold list at n
        {∃α_1, α_2, b, z, α_3. lseg(x, n, α_1) ⋆ n ↦ b, z ⋆ list(z, α_3) ⋆ α_2 ≐ [b] · α_3 ⋆ α ≐ α_1 · α_2 ⋆ t ≐ n}
        {∃α_1, b, z, α_3. lseg(x, n, α_1) ⋆ n ↦ b, z ⋆ list(z, α_3) ⋆ α ≐ α_1 · [b] · α_3 ⋆ t ≐ n}
        n := [n + 1];
        {∃α_1, b, α_3. lseg(x, t, α_1) ⋆ t ↦ b, n ⋆ list(n, α_3) ⋆ α ≐ α_1 · [b] · α_3}
        // Re-establish loop invariant
        {∃α_1, α_2, b. lseg(x, t, α_1) ⋆ t ↦ b, n ⋆ list(n, α_2) ⋆ α ≐ α_1 · [b] · α_2}
      };
      {∃α_1, α_2, b. lseg(x, t, α_1) ⋆ t ↦ b, n ⋆ list(n, α_2) ⋆ α ≐ α_1 · [b] · α_2 ⋆ n ≐ null}
      // α_2 = null
      {∃α_1, b. lseg(x, t, α_1) ⋆ t ↦ b, null ⋆ α ≐ α_1 · [b]}
    {∃α_1, b. lseg(x, t, α_1) ⋆ t ↦ b, null ⋆ α ≐ α_1 · [b] ⋆ list(y, β)}
    [t + 1] := y
    {∃α_1, b. lseg(x, t, α_1) ⋆ t ↦ b, y ⋆ α ≐ α_1 · [b] ⋆ list(y, β)}
    // append node to list segment
    {lseg(x, y, α) ⋆ list(y, β)}
    // append list to list segment
    {list(x, α · β)}
  };
  {P_4 = list(ret, α · β)[x/ret] = list(x, α · β)}
  return x
}
{list(ret, α · β)}
```

(a) When entering the function body, we initialise the local variables to `null`:

$$P_1 = \mathsf{list}(\mathtt{x}, \alpha) \star \mathsf{list}(\mathtt{y}, \beta) \star \mathtt{t}, \mathtt{n} \doteq \mathtt{null}$$

Before exiting the function body, if the function has post-condition Q and returns E, we need to reach the assertion $Q[E/\texttt{ret}]$. Therefore, in this case:

$$P_4 = \mathsf{list}(\mathsf{ret}, \alpha \cdot \beta)[\texttt{x}/\texttt{ret}] = \mathsf{list}(\texttt{x}, \alpha \cdot \beta)$$

(b) From the above proof sketch, we have that

$$P_2 = \exists a, \alpha'.\, \texttt{x} \mapsto a, \texttt{n} \star \mathsf{list}(\texttt{n}, \alpha') \star \alpha \doteq a{:}\alpha' \star \mathsf{list}(\texttt{y}, \beta) \star \texttt{t} \doteq \texttt{x}$$

(c) We have that

$$P_3 = \texttt{t} \mapsto a, \texttt{n} \star \mathsf{list}(\texttt{n}, \alpha') \star \alpha \doteq a{:}\alpha' \star \texttt{t} \doteq \texttt{x}$$

The existentials a and α' are removed using existential elimination, and $\mathsf{list}(\texttt{y}, \beta)$ is not used by the loop, and is therefore framed off.

(d) The loop is designed to reach the last node of the list starting at `x`. Before each iteration of the loop, we have: **(1)** the traversed part of the list, captured by a list segment from `x` to `t` holding some values α_1; **(2)** the current node at `t`, holding some value b and pointing to `n`; and **(3)** the part still to be traversed, captured by a list at `n`, holding some values α_2. All of the values also have to be appropriately connected, via **(4)** $\alpha = \alpha_1 \cdot [b] \cdot \alpha_2$.

(e) For **(1)**, as $\texttt{x} = \texttt{t}$, we trivially have an empty list segment starting in `x` and ending in `t`, holding an empty list, which is the initial value of α_1. For **(2)**, we have $\texttt{t} \mapsto a, \texttt{n}$, meaning that the initial value of b is a. For **(3)**, we have $\mathsf{list}(\texttt{n}, \alpha')$, meaning that the initial value of α_2 is α'. Finally, for **(4)**, given (1-3), $\alpha = a{:}\alpha'$ does entail $\alpha = \alpha_1 \cdot [b] \cdot \alpha_2$.

(f) The proof sketch is given above in full.

(g) Lemmas from lecture 1.