Software Verification
Exercise Solution: Separation Logic

By application of small axioms and the frame rule, we obtain the following proof outline:

copytree(i; j) =
    {tree τ i}
    if i = nil then
        {tree τ i ∧ i = nil}
        j := i
        {tree τ i ∧ i = nil ∧ j = i}  // ... by e.g. Hoare's forward assignment axiom.
        {empty ∧ τ = ε ∧ i = nil ∧ j = nil * empty}
    else
        newvar i₁, i₂, v, j₁, j₂ in
            {tree τ i ∧ i ≠ nil}
            (i → j, a, k) * (tree τ₁ i₁) * (tree τ₂ k) ∧ τ = (τ₁, a, τ₂)
            i₁ := [i];
            (i → i₁, a, k) * (tree τ₁ i₁) * (tree τ₂ k) ∧ τ = (τ₁, a, τ₂)
            v := [i + 1];
            (i → i₁, v, k) * (tree τ₁ i₁) * (tree τ₂ k) ∧ τ = (τ₁, v, τ₂)
            i₂ := [i + 2];
            (i → i₁, v, i₂) * (tree τ₁ i₁) * (tree τ₂ i₂) ∧ τ = (τ₁, v, τ₂)
            copytree(i₁, j₁);
            (i → i₂, v, i₂) * (tree τ₁ i₁) * (tree τ₂ i₂) ∧ τ = (τ₁, v, τ₂)
            copytree(i₂, j₂);
            (i → i₁, v, i₂) * (tree τ₁ i₁) * (tree τ₂ i₁) * (tree τ₂ i₂) * (tree τ₂ j₂) ∧ τ =
            (τ₁, v, τ₂)
            j := cons(j₁, v, j₂);
            (i → i₁, v, i₂) * (tree τ₁ i₁) * (tree τ₂ i₁) * (tree τ₂ i₂) * (tree τ₂ j₂) ∧ τ =
            (τ₁, v, τ₂)
            (j → j₁, v, j₂) ∧ τ = (τ₁, v, τ₂)
            (i → i₁, v, i₂) * (tree τ₁ i₁) * (tree τ₂ i₂) * (j → j₁, v, j₂) * (tree τ₁ j₁) *
            (tree τ₂ j₂) ∧ τ = (τ₁, v, τ₂)
            {tree τ i * tree τ j}
        end
    end
    {tree τ i * tree τ j}
Remarks:

There are, as usual, several proofs for the correctness of a single code snippet. For example, we can prove the first branch of the if-statement as follows:

{tree \( \tau \) \( i \) \& \( i = \text{nil} \)}
{\( \tau = \varepsilon \) \& \( \text{empty} \) \& \( i = \text{nil} \)}
{\( \tau = \varepsilon \) \& (empty \& i = \text{nil}) \& (empty \& j = \text{nil})} // By e.g. Hoare's backward axiom:
\( j := i \)
{\( \tau = \varepsilon \) \& (tree \( \varepsilon \) \( i \)) \& (tree \( \varepsilon \) \( j \))}
{tree \( \tau \) \( i \) \& tree \( \tau \) \( j \)}

The proof of this code snippet uses only familiar rules of Hoare logic: assignment and consequence. The implications used by the rule of consequence are of course now expressed in separation logic.

The next part of the proof employs the small axioms and the frame rule. It is convenient to use the following derived axiom for heap lookup:

{\( e \mapsto e' \) } \( x := [e] \) {\( e \mapsto e' \) \& \( x = e' \)}
provided \( x \) does not appear free in \( e \) or \( e' \).

Here is a detailed proof of the first heap lookup:

{tree \( \tau \) \( i \) \& \( i \neq \text{nil} \)}
{\( \exists j,a,k, \tau_1,\tau_2. (i \mapsto j,a,k) \& (tree \tau_1 j) \& (tree \tau_2 k) \& \tau = (\tau_1,a,\tau_2) \)}
{\( i \mapsto j \)}
{\( i_1 := [i] \)}
{\( i \mapsto j \& i_1 = j \)}
{((i \mapsto j \& i_1 = j)) \& (\exists a,k,\tau_1,\tau_2. (i+1 \mapsto a,k) \& (tree \tau_1 j) \& (tree \tau_2 k) \& \tau = (\tau_1,a,\tau_2))}
{\( \exists a,k,\tau_1,\tau_2. (i \mapsto j,a,k \& i_1 = j) \& (tree \tau_1 j) \& (tree \tau_2 k) \& \tau = (\tau_1,a,\tau_2) \)}
{\( \exists j,a,k,\tau_1,\tau_2. (i \mapsto j,a,k \& i_1 = j) \)}
{\( \exists a,k,\tau_1,\tau_2. (i \mapsto i_1,a,k) \& (tree \tau_1 i_1) \& (tree \tau_2 k) \& \tau = (\tau_1,a,\tau_2) \)}
{\( \exists a,k,\tau_1,\tau_2. (i \mapsto i_1,a,k) \)}
{\( \exists j,a,k,\tau_1,\tau_2. (i \mapsto i_1,a,k) \& (tree \tau_1 i_1) \& (tree \tau_2 k) \& \tau = (\tau_1,a,\tau_2) \)}

Note that we applied Auxiliary Variable Elimination to quantify only \( j \) - the other variables were quantified in the frame. In contrast to this, the following detailed proof of the first recursive call to copytree uses AuxVarElim to quantify both \( \tau_1 \) and \( \tau_2 \). It includes no quantifiers.
Here is a detailed proof of the final cons command. It is similar to the previous proof because it excludes existential quantifies from the frame:

\[
\exists \tau_1, \tau_2. (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_2 \ast i_2) \wedge \tau = (\tau_1, v, \tau_2)
\]

\[
{\exists \tau_1, \tau_2. (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_2 \ast i_2) \wedge \tau = (\tau_1, v, \tau_2)}
\]

\[
\{\text{tree } \tau_1 \ast i_1 \ast (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_2 \ast i_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{\text{copytree}(i_1, j_1)\}
\]

\[
\{\text{tree } \tau_1 \ast i_1 \ast \text{tree } \tau_1 \ast j_1\}
\]

\[
\{\text{tree } \tau_1 \ast i_1 \ast \text{tree } \tau_1 \ast j_1 \ast (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_2 \ast i_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{\exists \tau_1, \tau_2. (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_1 \ast j_1) \ast (\text{tree } \tau_2 \ast i_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{\emptyset \ast (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_1 \ast j_1) \ast (\text{tree } \tau_2 \ast i_2) \ast (\text{tree } \tau_2 \ast j_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{\text{empty} \}
\]

\[
\{j := \text{cons}(j_1, v, j_2)\}
\]

\[
\{j \mapsto j_1, v, j_2\}
\]

\[
\{j \mapsto j_1, v, j_2 \ast (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_1 \ast j_1) \ast (\text{tree } \tau_2 \ast i_2) \ast (\text{tree } \tau_2 \ast j_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{i \mapsto i_1, v, i_2 \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_1 \ast j_1) \ast (\text{tree } \tau_2 \ast i_2) \ast (\text{tree } \tau_2 \ast j_2) \ast (j \mapsto j_1, v, j_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]

\[
\{\exists \tau_1, \tau_2. (i \mapsto i_1, v, i_2) \ast (\text{tree } \tau_1 \ast i_1) \ast (\text{tree } \tau_1 \ast j_1) \ast (\text{tree } \tau_2 \ast i_2) \ast (\text{tree } \tau_2 \ast j_2) \ast (j \mapsto j_1, v, j_2) \wedge \tau = (\tau_1, v, \tau_2)\}
\]