Verification of Functional Programs

Having defined the semantics of functional programs, we proceed to present several methods for proving the correctness of such programs.

As in the case of imperative programs, the proof methods are partitioned into proofs for partial correctness and proofs for termination which, when combined, provide us with a proof of total correctness.
Partial Correctness: Formulation of the Problem

The first step is to decide about the style of specification. For functional programs, it is natural to consider an approach in which both the specification and implementation are given as functional programs, in which case, partial correctness amounts to proving the inclusion

\[ f_{\text{imp}} \subseteq f_{\text{spec}} \]

This inclusion ensures that for each argument \( \bar{a} \in (D^\perp)^n \) such that \( f_{\text{imp}}(\bar{a}) \) is defined (different from \( \perp \)), then so is \( f_{\text{spec}} \) and their values coincide. Note that \( f_{\text{spec}} \) may be defined over a bigger range than \( f_{\text{imp}} \).

For example, we can take \( F \Leftarrow \mathcal{T}_{91}[F] \), where

\[ \mathcal{T}_{91}[F](x) = \text{if } x > 100 \text{ then } x - 10 \text{ else } F(F(x + 11)) \]

as the implementation of the \( 91 \) function, and

\[ f_{91}(x) = \text{if } x > 100 \text{ then } x - 10 \text{ else } 91 \]

as its specification.

Verifying that \( f_{\mathcal{T}_{91}} \), the fix-point of \( \mathcal{T}_{91} \), is partially correct w.r.t the specification \( f_{91} \) amounts to proving the inclusion

\[ f_{\mathcal{T}_{91}} \subseteq f_{91} \]
Proofs Based on Computational Induction

We will somewhat generalize the problem and consider methods for proving that the least fix-point solution of the functional program $F \leftarrow T[F]$ satisfies a predicate $\lambda f : \Phi(f)$. Partial correctness is a special case of this more general problem by taking the predicate to be $\Phi = \lambda f : f \subseteq f_{\text{spec}}$.

All the proof methods we will present are based on the following approach: We will prove by induction on $i = 0, 1, \ldots$ that $T^i(\Omega) \models \Phi$, where $T^i(\Omega)$ are the successive approximations to the fix-point $f_T = \text{lub}\{T^i(\Omega)\}$. From this, we wish to infer that $f_T \models \Phi$.

Consequently, we say that a predicate $\lambda f : \Phi$ is admissible if, for every $S$ a directed set of functions, $\forall f \in S : \Phi(f)$ implies $\Phi(\text{lub}(S))$.

Not every predicate is admissible. Consider the predicate

$$\Phi(f) : \exists x \in \mathbb{N} : f(x) = \bot$$

and the directed set $S = \{f_i(x) : \text{if } x < i \text{ then } x! \text{ else } \bot\}$. Clearly $\Phi(f_i) = \text{true}$ for every $i \geq 0$, yet $\Phi(\text{lub}(S)) = \Phi(\lambda x : x!) = \text{false}$. 
Sufficient Conditions for Admissibility

Claim 37.
Every predicate of the form \( \bigwedge_{i=1}^{n} \alpha_i[F] \sqsubseteq \beta_i[F] \), where \( \alpha_1, \ldots, \alpha_n \) are continuous functionals and \( \beta_1, \ldots, \beta_n \) are monotonic functionals, is admissible.

Proof:
Let \( \Phi \) be of the form required by the claim, and let \( S \) be a directed set of functions such that \( \alpha_i[f] \sqsubseteq \beta_i[f] \) for every \( i \in [1..n] \) and \( f \in S \), i.e. \( \Phi(f) = \text{true} \) for every \( f \in S \). Let \( h = \text{lub}(S) \). Consider a specific \( i \in [1..n] \). Since \( f_i \sqsubseteq h \) and \( \beta_i \) is monotonic, we have \( \alpha_i[f] \sqsubseteq \beta_i[f] \sqsubseteq \beta_i[h] \). Thus, \( \beta_i[h] \) is an upper bound of the set \( \{ \alpha_i[f] \mid f \in S \} \). It follows that the least upper bound of this set should be bounded by \( \beta_i[h] \), i.e., \( \text{lub}\{\alpha_i[f] \mid f \in S\} \sqsubseteq \beta_i[h] \). Since \( \alpha_i \) is continuous, we have that \( \text{lub}\{\alpha_i[f] \mid f \in S\} = \alpha_i[\text{lub}\{f \mid f \in S\}] = \alpha_i[h] \) and can, therefore conclude that \( \alpha_i[h] \sqsubseteq \beta_i[h] \). Since this holds for every \( i \in [1..n] \), it follows that \( \Phi(h) = \text{true} \).
Stepwise Computational Induction

Claim 38. [deBakker and Scott]
Let $P : F \leftarrow T[F]$ be a functional program over $[(D^\perp)^n \rightarrow D^\perp]$, and $\Phi$ be an admissible predicate. If $\Phi(\Omega) = \text{true}$ and $\Phi(f) \rightarrow \Phi(T[f])$ for every $f \in [(D^\perp)^n \rightarrow D^\perp]$, then $f_P \models \Phi$.

Obviously, the two assumptions of the claim imply that $T_i[\Omega] \models \Phi$ for every $i = 0, 1, \ldots$. Due to the admissibility of $\Phi$, it follows that $f_P \models \Phi$. 
Example 1

Consider the functional program

\[ P : F(x) \iff \text{if } p(x) \text{ then } x \text{ else } F(F(h(x))) \]

where \( p \) and \( h \) are monotonic functions, and \( p \) is not a constant function.

We wish to prove the equivalence \( f_P \circ f_P = f_P \). To do so, we take the admissible predicate \( \Phi(F) : \forall x : (f_P(F(x)) = F(x)) \). It is admissible since it can be presented as the conjunction \((f_P \circ F \subseteq F) \land (F \subseteq f_P \circ F)\).

We start by showing that \( \Omega \models \Phi \). First, we observe that since \( f_P \) is a fix-point of the functional \( \text{if } p(x) \text{ then } x \text{ else } F(F(h(x))) \), it satisfies

\[ f_P(\bot) = \text{if } p(\bot) \text{ then } \bot \text{ else } f_P(f_P(h(\bot))) = \text{if } \bot \text{ then } - \text{ else } - = \bot \]

since \( p \) is assumed to be a monotonic non-constant function. It follows that

\[ f_P \circ \Omega = \Omega \]

which establishes \( \Phi(\Omega) \).

Next, we establish that \( \Phi(f) \) implies \( \Phi(\mathcal{T}[f]) \), i.e. \( f_P \circ f = f \) implies \( f_P \circ \mathcal{T}[f] = \mathcal{T}[f] \). We compute as follows:

\[
\begin{align*}
  f_P(\mathcal{T}[f](x)) &= &
  f_P(\text{if } p(x) \text{ then } x \text{ else } f(f(h(x)))) &= &
  \text{Definition of } \mathcal{T} \\
  \text{if } p(x) \text{ then } f_P(x) \text{ else } f_P(f(f(h(x)))) &= &
  \text{Distributing } f_P \text{ over conditional} \\
  \text{if } p(x) \text{ then } (\text{if } p(x) \text{ then } x \text{ else } f_P(f_P(h(x)))) \text{ else } f_P(f(f(h(x)))) &= &
  \text{Substituting } \mathcal{T}[f_P] \text{ for } f_P \\
  \text{if } p(x) \text{ then } x \text{ else } f_P(f(f(h(x)))) &= &
  \text{Simplification.} \\
  \text{if } p(x) \text{ then } x \text{ else } (f(f(h(x)))) &= &
  \text{Induction hypothesis.} \\
  \mathcal{T}[f] &= &
  \text{Definition of } \mathcal{T}.
\end{align*}
\]
Example 2

Consider the functional program

\[ P : F(x, y) \leftarrow \text{if } p \, x \text{ then } y \text{ else } h \, F(k \, x, y) \]

where \( p, h, \) and \( k \) are monotonic non-constant functions. For this program we would like to prove \((\forall x, y) \, h \, f_P(x, y) = f_P(x, h \, y)\). As the admissible predicate, we take

\[ \Phi(F) : (\forall x, y) \, h \, F(x, y) = F(x, h \, y) \]

- For the base step, we have to show that \( \Phi(\Omega) \) holds, i.e. \( h \, \Omega(x, y) = \Omega(x, h \, y) \). This holds because \( h \bot = \bot \).

- For the induction step we assume that \( h \, f(x, y) = f(x, h \, y) \) for every \( x \) and \( y \), and prove that \( h \, T[f](x, y) = T[f](x, h \, y) \). We compute as follows:

\[
\begin{align*}
  h \, T[f](x, y) &= h \, (\text{if } p \, x \text{ then } y \text{ else } h \, f(k \, x, y)) \\
  &\quad = \text{if } p \, x \text{ then } h \, y \text{ else } h \, h \, f(k \, x, y) \\
  &\quad = \text{if } p \, x \text{ then } h \, y \text{ else } h \, f(k \, x, h \, y) \\
  &\quad = T[f](x, h \, y) \\
\end{align*}
\]

Distribution of \( h \) over conditional

Induction hypothesis for \( k \, x \) and \( y \)

Definition of \( T \)
Example 3: Append

Consider the Lisp function defined by

\[ P : F(x, y) \leftarrow \text{if } x = \bot \Lambda \text{ then } y \text{ else } \text{hd}(x \cdot F(tl(x), y)) \]

where \( \Lambda \) denotes the empty list, \( \text{hd} \) and \( \text{tl} \) denote the head and tail functions (\text{car} and \text{cdr}, respectively), and \( \cdot \) denotes the \text{cons} function. This program defines the \text{append} function, often denoted by \( \ast \). We wish to prove that the \text{append} function is associative, that is \((x \ast y) \ast z = x \ast (y \ast z)\) for every lists \( x, y \) and \( z \). To do so, we define the predicate \( \Phi(F) : (\forall x, y, z) \ f_P(F(x, y), z) = F(x, f_P(y, z)) \). We then reason as follows:

- The base step is established by
  \[ f_P(\Omega(x, y), z) = f_P(\bot, z) = \bot = \Omega(x, f_P(y, z)) \]
  using the fact that \( \bot = \bot \Lambda \) evaluates to \( \bot \).

- For the induction step, we assume \( f_P(f(x, y), z) = f(x, f_P(y, z)) \) and establish
  \[ f_P(\mathcal{T}[f](x, y), z) = \mathcal{T}[f](x, f_P(y, z)) \]. Compute as follows:

\[
\begin{align*}
\text{if } x = \bot \Lambda \text{ then } y \text{ else } \text{hd}(x \cdot f(tl(x), y)), z) &= \text{Definition of } \mathcal{T} \\
\text{if } x = \bot \Lambda \text{ then } f_P(y, z) \text{ else } f_P(\text{hd}(x \cdot f(tl(x), y)), z)) &= \text{Distribution of } f_P \text{ over conditional, noting that } f_P(\bot, y) = \bot. \\
\text{if } x = \bot \Lambda \text{ then } f_P(y, z) \text{ else } \text{hd}(x \cdot f_P(f(tl(x), y), z)) &= \text{Definition of } f_P + \text{simplification} \\
\text{if } x = \bot \Lambda \text{ then } f_P(y, z) \text{ else } \text{hd}(x \cdot f(tl(x), f_P(y, z))) &= \text{Induction hypothesis} \\
\mathcal{T}[f](x, f_P(y, z))
\end{align*}
\]
Systems of Recursive Definitions

The theory of fix-points already allowed functionals mapping functions in \([(D^\perp_1)^n \mapsto (D^\perp_2)^m]\) into the same domain. To make use of this general capabilities, we add to the set of constructions defining \textit{constructible functionals} also the \textit{tuple formation} construction, which allows combining constructible functionals \(T_1, \ldots, T_k\) each mapping \([(D^\perp_1)^n \mapsto (D^\perp_2)^m]\) into itself, into the tuple functional \((T_1, \ldots, T_k)\) of the same type.

In practice, we use \([(D^\perp_1)^n \mapsto (D^\perp_2)^m]\) functionals in a recursive definition of a \textit{system of functions}, having the following form:

\[
F_1(\bar{x}) \iff T_1[F_1, \ldots, F_m](\bar{x}) \\
\ldots \\
F_m(\bar{x}) \iff T_m[F_1, \ldots, F_m](\bar{x})
\]

which simultaneously defines \(m\) functions of type \([(D^\perp)^n \mapsto D^\perp]\).

For example, we may consider the following recursive system definition:

\[
F_1(x) \iff \text{if } x > 100 \text{ then } x - 10 \text{ else } F_1(F_2(x + 11)) \\
F_2(x) \iff \text{if } x > 100 \text{ then } x - 10 \text{ else } F_2(F_1(x + 11))
\]

whose least fix-point solution is given by \((f_{91}(x), f_{91}(x))\).
Example: Proving Equivalence between Functions

The proof method of stepwise computational induction can be easily extended to deal with recursive system definitions of the form \((F_1, \ldots, F_m) \iff (T_1[F], \ldots, T_m[F])\). Assume that \(\Phi(F)\) is an admissible predicate. Then the extended method requires showing that \(\Phi(\Omega, \ldots, \Omega) = \text{true}\) and that, for an arbitrary tuple of monotonic functions \(\bar{f} : (f_1, \ldots, f_m)\), \(\Phi(\bar{f})\) implies \(\Phi(T_1[\bar{f}], \ldots, T_m[\bar{f}])\).

We can use this proof method to establish equivalence between functions. Consider, for example, the following two recursive programs:

\[
\begin{align*}
P_1: \quad F_1(x, y) & \iff \text{if } p(x) \text{ then } y \text{ else } hF_1(kx, y) \\
P_2: \quad F_2(x, y) & \iff \text{if } p(x) \text{ then } y \text{ else } F_2(kx, hy)
\end{align*}
\]

where \(p\), \(h\), and \(k\) are monotonic non-constant functions. To apply the method, we reconsider these two definitions as constituting a single system of two recursively defined functions. As the admissible predicate we take \(\Phi(F_1, F_2) : (\forall x, y)(F_1(x, y) = F_2(x, y)) \land (F_2(x, hy) = hF_2(x, y))\), which strengthens our desired goal of proving that the two functions are equivalent. We proceed in two steps:

Base step: \(\Phi(\Omega, \Omega) = \text{true}\)

Substituting \(\Omega\) for both \(F_1\) and \(f_2\), the first conjunct yields \(\Omega(x, y) = \Omega(x, y)\) which trivially holds. The second conjunct yields \(\Omega(x, hy) = h\Omega(x, y)\) which also holds since both terms evaluate to \(\bot\), based on the fact that \(h(\bot) = \bot\).
Example Continued

**Induction step:** \( \Phi(f_1, f_2) \) implies \( \Phi(T_1[f_1], T[f_2]) \)

We assume that \( (f_1(x, y) = f_2(x, y)) \land (f_2(x, h\, y) = h\, f_2(x, y)) \) holds for every \( x \) and \( y \), and proceed to prove that the same equalities hold when we replace \( f_1 \) and \( f_2 \) by \( T_1[f_1] \) and \( T_2[f_2] \), respectively. For the first conjunct we compute:

\[
\begin{align*}
T_1[f_1](x, y) & = \text{If } p(x) \text{ then } y \text{ else } h\, f_1(k\, x, y) \quad = \text{Definition of } T_1 \\
& = \text{Induction hypothesis } f_1 = f_2 \\
& = \text{Induction hypothesis } f_2(x, h\, y) = h\, f_2(x, y) \\
& \text{Definition of } T_2
\end{align*}
\]

The second conjunct leads to:

\[
\begin{align*}
T_2[f_2](x, h\, y) & = \text{If } p(x) \text{ then } h\, y \text{ else } f_2(k\, x, h\, h\, y) \quad = \text{Definition of } T_2 \\
& = \text{Induction hypothesis } f_2(x, h\, u) = h\, f_2(x, u) \text{ for any } u \\
& = h\, \text{If } p(x) \text{ then } y \text{ else } f_2(k\, x, h\, y) \quad = \text{Distribution of } h \text{ over conditional } (h(\bot) = \bot) \\
& \quad \text{Definition of } T_2 \\
& \quad \text{Definition of } T_2
\end{align*}
\]
Complete Computational Induction

Stepwise computational induction can be viewed as an induction establishing \( \Phi(T^{i+1}[\Omega]) \) based on \( \Phi(T^i[\Omega]) \). A more general induction principle, often referred to as complete induction establishes \( \Phi(T^i[\Omega]) \) based on the hypothesis that \( \Phi(T^j[\Omega]) \) holds for all \( j < i \). This leads to the following proof principle:

**Claim 39. [Complete Computational Induction]**

Let \( P : F \Leftarrow T[F] \) be a functional program and \( \Phi \) be an admissible predicate. If by assuming that \( \Phi(T^j[\Omega]) \) holds for all \( j < i \), we can establish \( \Phi(T^i[\Omega]) \), then \( \Phi \) holds for the least fix-point \( f_P \).

The claim can also be stated for the more general case of recursive system definition.

Reconsider the mutually recursive system:

\[
F(x, y) \Leftarrow \text{if } p(x) \text{ then } y \text{ else } h F(k x, y) \\
G(x, y) \Leftarrow \text{if } p(x) \text{ then } y \text{ else } G(k x, h y)
\]

we will use complete computational induction in order to prove that \( f_P = g_P \), where \( f_P \) and \( g_P \) are the least fix-point solutions of this system. We will use the notation \( f_i, g_i \) to stand for the successive approximations \( T^i_F[\Omega], T^i_G[\Omega] \). As the admissible functional we take \( \Phi(F, G) : F = G \). Note that, in comparison with the stepwise computational induction proof, the admissible functional here is much simpler. We proceed in three steps:
Example Continued

Step 0: Establish $\Phi(f_0, g_0)$
$
\Phi(\Omega, \Omega)$ reduces to $\Omega = \Omega$ which obviously holds.

Step 1: Establish $\Phi(f_1, g_1)$

We compute

$$f_1(x, y) = \text{if } p(x) \text{ then } y \text{ else } h f_0(k x, y) =$$

$$\text{if } p(x) \text{ then } y \text{ else } h \Omega(k x, y) = \text{Definition of } f_0$$

$$\text{if } p(x) \text{ then } y \text{ else } \bot = \text{Sine } h \bot = \bot$$

$$\text{if } p(x) \text{ then } y \text{ else } \Omega(k x, h y) =$$

$$\text{if } p(x) \text{ then } y \text{ else } g_0(k x, h y) = \text{Definition of } g_0$$

Definition of $g_1$

Induction Step: Assuming $f_i = g_i$ and $f_{i+1} = g_{i+1}$, establish $f_{i+2} = g_{i+2}$

$$f_{i+2} =$$

$$\text{if } p(x) \text{ then } y \text{ else } h f_{i+1}(k x, y) = \text{Definition of } f_{i+2}$$

$$\text{if } p(x) \text{ then } y \text{ else } h g_{i+1}(k x, y) = \text{Induction hypothesis } f_{i+1} = g_{i+1}$$

$$\text{if } p(x) \text{ then } y \text{ else } h (\text{if } p(x) \text{ then } y \text{ else } g_i(k k x, h y)) \text{ Definition of } g_{i+1}$$

$$= \text{Induction hypothesis } f_i = g_i$$

$$\text{if } p(x) \text{ then } y \text{ else if } p(x) \text{ then } h y \text{ else } h f_i(k k x, h y) = \text{Distribution of } h \text{ over conditional}$$

$$\text{if } p(x) \text{ then } y \text{ else } f_{i+1}(k x, h y) = \text{Definition of } f_{i+1}$$

$$\text{if } p(x) \text{ then } y \text{ else } g_{i+1}(k x, h y) = \text{Induction hypothesis } f_{i+1} = g_{i+1}$$

Definition of $g_{i+2}$
Fix-Point Induction

Claim 40. [Fix-point Induction, Park]
Consider the functional program $P : F \leftarrow \mathcal{T}_P[F]$ over $(D^\bot)^n \rightarrow D^\bot$ and let $g \in [(D^\bot)^n \rightarrow D^\bot]$. If $\mathcal{T}_P[g] \subseteq g$, then $f_P \subseteq g$.

Proof:
We apply stepwise computational induction to the admissible functional $\Phi(F) : F \subseteq g$. Obviously $\Phi(\Omega)$ holds, since $\Omega \subseteq g$. Assuming $f \subseteq g$, we may apply the monotonic functional $\mathcal{T}_P$ to both sides to obtain

$$\mathcal{T}_P[f] \subseteq \mathcal{T}_P[g] \subseteq g$$

It follows that $f \subseteq g$ implies $\mathcal{T}_P[f] \subseteq g$.

Consider, for example, the 91 functional program

$$P_{91} : \quad F(x) \leftarrow \text{if } x > 100 \text{ then } x - 10 \text{ else } F(F(x + 11))$$

and the proposed specification $f_{91} : \text{if } x > 100 \text{ then } x - 10 \text{ else } 91$. Since $f_{91}$ satisfies $\mathcal{T}_{91}[f_{91}] = f_{91}$ (implying $\mathcal{T}_{91}[f_{91}] \subseteq f_{91}$), we conclude that the fix-point of $P_{91}$ must be contained in $f_{91}$, i.e. satisfies the specification $f_{91}$.

Corollary 41.
For a given functional program $P : F \leftarrow \mathcal{T}_P[F]$, it is sufficient to characterize the fix-point $f_P$ as the least function $f$ satisfying the inclusion $\mathcal{T}_P[f] \subseteq f$.

Proof:
Let $f_P$ be the least fix-point solution of $F = \mathcal{T}_P[f]$. Obviously, $f_P$ satisfies the inclusion $\mathcal{T}_P[f_P] \subseteq f_P$. According to Claim 40, it is also the least function satisfying this inclusion.
Fix-point Preserving Transformations

Consider the two functional programs $P : F \leftarrow \mathcal{T}_P[F]$ and $Q : F \leftarrow \mathcal{T}_Q[F]$, where we are interested in proving the equivalence $f_P = f_Q$. A trivial case is when $\mathcal{T}_P$ is equivalent to $\mathcal{T}_Q$, because in this case, the two programs are equivalent. In more interesting cases, $\mathcal{T}_P$ is not directly equivalent to $\mathcal{T}_Q$, but we may apply to $\mathcal{T}_P$ fix-point preserving transformations which will cause $\mathcal{T}_P$ to become equivalent to $\mathcal{T}_Q$.

**Claim 42. [Fix-point Preserving Transformations, Vuilleman]**

Let $P : F \leftarrow \mathcal{T}_P[F]$ and $Q : F \leftarrow \mathcal{T}_Q[F]$ be two functional programs such that $\mathcal{T}_Q$ is obtained from $\mathcal{T}_P$ by replacing some occurrences of the function symbol $F$ by either $\mathcal{T}_P[F]$ or $f_P$. Then $f_P = f_Q$.

**Proof:**

Let $\mathcal{T}[F]$ be a functional which we present as $\mathcal{T}[F, F]$ in order to distinguish the occurrence of $F$ which will be rewritten from all other occurrences of $F$ in $\mathcal{T}$. Let $f_\mathcal{T}$ denote the least fix-point of $\mathcal{T}$. Let us define two new functionals

$$\mathcal{T}_1[F] = \mathcal{T}[F, \mathcal{T}[F]] \quad \text{and} \quad \mathcal{T}_2[F] = \mathcal{T}[F, f_\mathcal{T}]$$

Note that $\mathcal{T}_1$ and $\mathcal{T}_2$ are obtained from $\mathcal{T}$ by the two rewriting transformations described in the claim. Let $f_{\mathcal{T}_1}$ and $f_{\mathcal{T}_2}$ be the least fix-point solutions of the functionals $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively.

our goal is to prove that $f_\mathcal{T} = f_{\mathcal{T}_1} = f_{\mathcal{T}_2}$, and we will show this in two steps.
Proof Continued

Step 1: \( f_{\tau_1} \subseteq f_\tau \) and \( f_{\tau_2} \subseteq f_\tau \). 

By definition of \( \tau_1 \) and \( \tau_2 \), \( f_\tau = \tau_1[f_\tau] = \tau_2[f_\tau] \); that is, \( f_\tau \) is a fix-point of both \( \tau_1 \) and \( \tau_2 \) and is, therefore, more defined than the respective least fix-points of these functionals, namely \( f_{\tau_1} \) and \( f_{\tau_2} \).

Step 2: \( f_\tau \subseteq f_{\tau_1} \) and \( f_\tau \subseteq f_{\tau_2} \).

This is proved by stepwise computational induction using

\[
\Phi(F, F_1, F_2) : \ (F \subseteq F_1) \land (F \subseteq F_2) \land (F \subseteq f_\tau) \land (F \subseteq \tau[F, F])
\]

Obviously \( \Phi(\Omega, \Omega, \Omega) \) holds. Let us assume that \( \Phi(f, f_1, f_2) \) holds and show that \( \Phi(\tau[f, f], \tau_1[f_1], \tau_2[f_2]) \) must also hold. From the induction hypothesis \( f \subseteq \tau[f, f] \) and the monotonicity of \( \tau \), it follows that \( \tau[f, f] \subseteq \tau[f, \tau[f, f]] = \tau_1[f] \).

From the induction hypothesis \( f \subseteq f_1 \) and the monotonicity of \( \tau_1 \), we conclude

(1) \( \tau[f, f] \subseteq \tau_1[f_1] \). Similarly, from the induction hypotheses \( f \subseteq f_\tau \) and \( f \subseteq f_2 \), we obtain

\[
\tau[f, f] \subseteq \tau[f_\tau, f_\tau] \subseteq \tau[f_2, f_\tau] = \tau_2[f_2]
\]

leading to (2) \( \tau[f, f] \subseteq \tau_2[f_2] \). From the induction hypothesis \( f \subseteq f_\tau \) we obtain

\[
\tau[f, f] \subseteq \tau[f_\tau, f_\tau] = f_\tau
\]

leading to (3) \( \tau[f, f] \subseteq f_\tau \). Finally, from the induction hypothesis \( f \subseteq \tau[f, f] \), we obtain (4) \( \tau[f, f] \subseteq \tau[\tau[f, f], \tau[f, f]] \).

Altogether, these establish \( \Phi(\tau[f, f], \tau_1[f_1], \tau_2[f_2]) \).
**Example 4**

Consider the two functional programs over $\mathbb{N}$:

\[
P : \quad F(x) \leftarrow \text{if } x > 10 \text{ then } x - 10 \text{ else } F(F(x + 13))
\]

\[
Q : \quad F(x) \leftarrow \text{if } x > 10 \text{ then } x - 10 \text{ else } F(x + 3)
\]

We wish to prove that $f_P = f_Q$.

Consider the transformed functional obtained from $\mathcal{T}_P$ by replacing the occurrence $F(x + 13)$ by $\mathcal{T}_P[F](x + 13)$. It is given by

\[
\mathcal{T}_R[F](x) : \quad \text{if } x > 10 \text{ then } x - 10 \text{ else } F(\text{if } x + 13 > 10 \text{ then } x + 13 - 10 \text{ else } F(F(x + 13 + 13)))
\]

Since $x$ is natural, $x + 13 > 10$ is always true. Therefore the second line in this definition can be simplified to $F(x + 13 - 10) = F(x + 3)$ which makes $\mathcal{T}_R$ identical to $\mathcal{T}_Q$. This establishes that $f_P = f_Q$. 