PFPL Supplement: Dynamic Dispatch as an Abstract Type

Robert Harper

September, 2018

Dynamic dispatch may be seen as an abstract type of objects supporting two methods, creation
of an object of a class, and sending a message to an object to obtain a result. The textbook describes
an example with two classes of complex number, cart and pol, and two methods, dist and quad.

A dynamic dispatch scenario with classes C and methods D specified by instance types \( \tau_c \) for
each \( c \in C \), and result types \( \rho_d \) for each \( d \in D \), may be organized as implementation of the following
existential type:

\[
\tau_{dd} \triangleq \exists (t_{obj} \cdot (\text{new} \leftrightarrow \langle \tau_c \rightarrow t_{obj} \rangle_{c \in C}, \text{snd} \leftrightarrow \langle t_{obj} \rightarrow \rho_d \rangle_{d \in D})).
\] (1)

Given a package, call it \( x \), of this type, a client may open \( x \) to gain access to the object creation
and message send operations as follows:

\[
\text{open } x \text{ as } t_{obj} \text{ with } (\text{new} \leftrightarrow \text{new}, \text{snd} \leftrightarrow \text{snd}) \text{ in } e_{\text{client}}
\] (2)

Within \( e_{\text{client}} \) we may define

\[
\text{new}[c](e^c) \triangleq \text{new} \cdot c(e^c) \quad \text{snd}[e](d) \triangleq \text{snd} \cdot d(e)
\]

wherein \( e^c : \tau^c \) is instance data appropriate to class \( c \) and \( e : t_{obj} \). A new instance of class \( c \) is
created by the \text{new} operation applied to an argument of type \( \tau^c \). It creates an abstract object of
type \( t_{obj} \) to which one may send a message \( d \) to obtain a result of type \( \rho_d \). Importantly, the types \( \tau^c \)
and \( \rho_d \) are independent of the abstract type \( t_{obj} \); their values are meaningful outside of the object
abstraction. Moreover, the statics of the \text{open} expression ensures that the type of the client also be
meaningful apart from \( t_{obj} \) so as to ensure that abstraction is not violated.

For example, in the case of the abstract type of complex numbers, we may write

\[
z \triangleq \text{new}[\text{cart}](\langle x \leftrightarrow x, y \leftrightarrow y \rangle)
\]

\[
= \text{new} \cdot \text{cart}(\langle x \leftrightarrow x, y \leftrightarrow y \rangle)
\]

to create a complex number \( z \) with rectangular coordinates \((x, y)\). We may then write

\[
u \triangleq \text{snd}[\text{dist}](z)
\]

\[
= \text{snd} \cdot \text{dist}(z)
\]

to compute the squared distance of \( z \) from the origin, namely \( x^2 + y^2 \).

Two natural implementations of the type \( \tau_{dd} \) arise, one by taking an object to be a tuple of
methods, one for each method \( d \in D \), and one taking an object to be an instance datum labelled
with some class $c \in C$. In the former case creating an object requires some work, but sending a message is simply a projection. In the latter case creating an object is simply an injection, but sending a message requires a case analysis. More precisely, we may define two packages of type $\tau_{dd}$, namely

$$\text{pack } \tau_{\text{obj}} \equiv (\text{new } \mapsto e_{\text{new}}, \text{snd } \mapsto e_{\text{snd}})$$

and

$$\text{pack } \tau_{\text{obj}} \equiv (\text{new } \mapsto e_{\text{new}}, \text{snd } \mapsto e_{\text{snd}})$$

whose components are defined as follows:

$$\tau_{\text{obj}} \equiv (d \mapsto \rho_d)_{d \in D}$$

$$e_{\text{new}} \equiv (\lambda (x^c : \tau^c) (d \mapsto (\star))_{d \in D})_{c \in C}$$

$$e_{\text{snd}} \equiv (\lambda (x : t_{\text{obj}}) x \cdot d)_{d \in D}$$

and

$$\tau_{\text{obj}} \equiv [c \mapsto \tau_c]_{c \in C}$$

$$e_{\text{new}} \equiv (\lambda (x^c : \tau^c) c \cdot x^c)_{c \in C}$$

$$e_{\text{snd}} \equiv (\lambda (x : t_{\text{obj}}) \text{ case } x \{c \cdot x^c \mapsto (\star)\})_{c \in \rho_d}$$

In each case $(\star)$ stands for the expression $e_{\text{DM}} \cdot c \cdot d(x^c)$, which invokes the appropriate code from the dispatch matrix.

For example, in the case of the complex numbers an object is either a tuple of type

$$\tau_{\text{obj}} \equiv (\text{dist } \mapsto \rho_{\text{dist}}, \text{quad } \mapsto \rho_{\text{quad}}),$$

or an injection of type

$$\tau_{\text{obj}} \equiv [\text{cart } \mapsto \tau_{\text{cart}}, \text{pol } \mapsto \tau_{\text{pol}}].$$

The creation and message sending operations are defined according to the general case given above.

These two representations are “equivalent” in the sense that no client of the dynamic dispatch abstraction can distinguish between them; the client’s behavior is the same whichever form is used. The key to proving this is to observe that the typing rule for $\text{open}$ ensures that (a) the client is polymorphic in the abstract type $t_{\text{obj}}$ of objects, and (b) the client computes a value of an extrinsically meaningful type, one that does not involve $t_{\text{obj}}$. Taken together, this means that we may interpret the abstract type by a binary simulation relation that relates the two implementations. As long as the $\text{new}$ and $\text{snd}$ operations preserve this relation, we are ensured that the client behavior is the same, regardless of which implementation is chosen.

Thus, the first step is to define a simulation relation between the two implementation types, $\tau_{\text{obj}}$ and $\tau_{\text{obj}}$, that expresses when two values of disparate types are “equivalent” insofar as the $\text{new}$ and $\text{snd}$ operations are concerned. So, in what sense is a tuple of methods equivalent to an injected instance value? The question can only be answered by reference to the implementations of the associated operations. If we examine implementation (I), we see that the methods in the tuple are given by the code in the dispatch matrix, specialized to the instance data used to create the tuple. If we examine implementation (II), we see that the injection of the instance data is used to select
methods from the dispatch matrix appropriate to that instance. This suggests that we define the binary relation $R$ between the two implementation types as follows:

$$R(e^I, e^\Pi) \iff e^I \mapsto^* (e^I_{d})_{d \in D}, \quad e^\Pi \mapsto^* c \cdot e^\Pi_c, \quad \text{and for all } d \in D, \quad e^I_d = \rho_d e_{DM} \cdot c \cdot d(e^\Pi_c).$$

Because the dynamics is deterministic, and from its definition, we have that the relation $R$ respects evaluation in that $R(e^I, e^\Pi)$ iff $e^I \mapsto^* e$, $e^\Pi \mapsto^* e'$ and $R(e, e')$.

This relation is preserved by the new and snd operations. More precisely, interpreting the type $t_{obj}$ by the relation $R$, we have

1. If $e^I_c = \tau c \cdot e^\Pi_c$, then new$^I \cdot c(e^I_c) = t_{obj} new^\Pi \cdot c(e^\Pi_c)$, i.e., $R(new^I \cdot c(e^I_c), new^\Pi \cdot c(e^\Pi_c))$.

2. If $e^I = t_{obj} e^\Pi$, i.e., $R(e^I, e^\Pi)$, then snd$^I \cdot d(e^I) = \rho_d snd^\Pi \cdot d(e^\Pi)$

Herein we have used the variables new$^I$ and new$^\Pi$, and similar annotations for snd, to remind the reader to consider these in the sense given by appropriate implementation.

Let us consider the verifications required.

1. By definition

$$\text{new}^I \cdot c(e^I_c) \mapsto^* e^I_{\text{new}}(e^I_c) \mapsto^* (d \mapsto e_{DM} \cdot c \cdot d(e^I_c))_{d \in D},$$

and, similarly,

$$\text{new}^\Pi \cdot c(e^\Pi_c) \mapsto^* c \cdot e^\Pi_c.$$

For these to be related by $R$, it suffices to show for all $d \in D$,

$$e_{DM} \cdot c \cdot d(e^I_c) = \rho_d e_{DM} \cdot c \cdot d(e^\Pi_c).$$

Now, by the parametricity theorem, the $(c, d)$ entry of the dispatch matrix is related to itself by equality at type $\tau c \rightarrow \rho_d$. Because we are assuming $e^I_c = \tau c \cdot e^\Pi_c$, the desired equation follows directly from the definition of equality at a function type.

2. By definition of the implementation

$$\text{snd}^I \cdot d(e^I) \mapsto^* e^I_{\text{snd}} \cdot d(e^I) \mapsto^* e^I_{\text{snd}} \cdot d(e^I)$$

and

$$\text{snd}^\Pi \cdot d(e^\Pi) \mapsto^* e^\Pi_{\text{snd}} \cdot d(e^\Pi) \mapsto^* \text{case } e^\Pi \{ c \cdot \cdot x^c \leftarrow e_{DM} \cdot c \cdot d(x^c) \}_{c \in \rho_d} \text{ case } e^\Pi \{ c \cdot \cdot x^c \leftarrow e_{DM} \cdot c \cdot d(x^c) \}_{c \in \rho_d} \mapsto e^I_{\text{snd}} \cdot d(e^I)$$

Because we are assuming $R(e^I, e^\Pi)$, we have

$$e^I_{\text{snd}} \cdot d(e^I) \mapsto^* (e^I_d)_{d \in D} \cdot d \mapsto e^I_d$$

and

$$\text{case } e^\Pi \{ c \cdot \cdot x^c \leftarrow e_{DM} \cdot c \cdot d(x^c) \}_{c \in \rho_d} \mapsto^* \text{ case } c \cdot e^I_{\Pi} \{ c \cdot \cdot x^c \leftarrow e_{DM} \cdot c \cdot d(x^c) \}_{c \in \rho_d} \mapsto e_{DM} \cdot c \cdot d(e^I_c)$$

and

$$e^I_d = \rho_d e_{DM} \cdot c \cdot d(e^I_c),$$

which completes the proof.

References