

Re: Formal and informal type systems?

Source: <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2004-09/1136.html>

From: Mark Lorenzen (mark.lorenzen_at_ofir.dk)

Date: 09/29/04

Date: 29 Sep 2004 19:53:30 +0200

"Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> writes:

```
> But this is already possible. Consider the following:
>
> type Optional_Integer (Defined : Boolean) is tagged record
> case Defined is
> when True => Value : Integer;
> when False => null;
> end case;
> end record;
>
> type Defined_Integer is
> new Optional_Integer (True) with null record;
>
> type Undefined_Integer is
> new Optional_Integer (False) with null record;
>
> procedure Do_Something (I : Defined_Integer) is
> begin
> Put_Line ("Defined:" & Integer'Image (I.Value));
> end;
>
> procedure Do_Something (I : Undefined_Integer) is
> begin
> Put_Line ("Undefined");
> end;
>
> The point is that there is no need to invent extra parameter matching
> mechanism. One should use and possibly extend the existing one based on
> inheritance.
```

In your example, the correct function is chosen by dispatching (or?). This would work, but it does not bind any variables in the argument pattern. It can of course always be done in Ada, I just think that pattern matching is a very elegant technique, that is often unknown to programmers.

comp.lang.ada: Re: Formal and informal type systems?

>
> > *The pattern matching could also be extended to case statements such as:*
> >
> > *procedure Do_Something (I : Optional_Integer) is*
> > *begin*
> > *case I is*
> > *when Optional_Integer'(Defined => False) => null;*
> > *when Optional_Integer'(Defined => True, Value) =>*
> > *Do_Something_More(Value);*
> > *end case;*
> > *end Do_Something;*
> >
> > *(Credit goes to one of my colleagues for the idea of using the type*
> > *name as constructor in the patterns.)*
>
> *I don't understand this example. Looks just like a class-wide of*
> *Optional_Integer. Am I right?*

The proposal does not have anything to do with tagged types, but discriminated records where Defined is the discriminant.

> *To start with, we could introduce function types. I cannot understand why*
> *we have access-to-function types and still have to types for the target.*

Function types would be very cool, but it is probably be a huge change in the language. We would end up with a language based more or less directly on the lambda calculus and the applicative languages are much better at that.

> *That requires structured type matching, which is a bad idea, in general.*

Generally yes.

>
> --
> *Regards,*
> *Dmitry A. Kazakov*
> *<http://www.dmitry-kazakov.de>*

Regards,
– Mark Lorenzen