Exercise Compiler Construction 9, improved version

Hans de Nivelle

Due: 10.01.2012

It has become clear that the implemenetation of the type checking algorithm is much harder than I initially imagined, so we have to try again. The main problem is that one needs polymorphic functions. For example, pointer addition has type

$$\forall X : \mathbf{func}(\mathbf{pointer}(X);\ \mathbf{pointer}(X), \mathbf{int}).$$

In order to apply it on a concrete type, (for example **pointer**(**double**)), one has to match **pointer**(X) into the concrete pointer type to obtain a value for X. (It would be $X =$ **double** in this case.) Polymorphism should not be confused with overloading. Polymorphism means that one function is strong enough to handle different types. Overloading means that there are different functions. In the classes **concretefunc** and **convfunc**, the list **typevars** is used for storing the quantified variables.
I believe in Object-Oriented programming, so I made 5 additional types, which are all members of **typesystem**.

1. Write the function **strictlymoregeneralthan** of class `typesystem::application`. The easiest way to do this, is to make use of the costs in the conversions.

2. Using **strictlymoregeneralthan**, implement `typesystem::mostgeneral( )`.

3. Implement `typesystem::allapplications`, which finds all the ways in which an overloading of function **func** could be applied on the conversions in **args**.

   Don't forget about polymorphic functions! You can see in `typesystem::allconversions` how they are used.

4. Now it should be easy to implement the function `typesystem::typecheck`.