Typed Assembly: From Function to Text

Daniel Nikpayuk

August 5, 2021

What is typed assembly? The easiest way to explain is it's a deconstruction of the idea of function **composition**. With composition, you take functions $f_0, f_1, \ldots, f_{n-1}, f_n$ and compose them together:

$$f_n \circ f_{n-1} \circ \ldots \circ f_1 \circ f_0$$

Simple enough, but this only works if the functions have composable domains/codomains:

$$\begin{array}{rcll}
f_0 & : & X_0 & \to & X_1 \\
f_1 & : & X_1 & \to & X_2 \\
 & & & \vdots & \\
f_{n-1} & : & X_{n-1} & \to & X_n \\
f_n & : & X_n & \to & X_{n+1}
\end{array}$$

What if your functions instead have arbitrary domains?

$$\begin{array}{rcll}
f_0 & : & X_0 & \to & Y_0 \\
f_1 & : & X_1 & \to & Y_1 \\
 & & & \vdots & \\
f_{n-1} & : & X_{n-1} & \to & Y_{n-1} \\
f_n & : & X_n & \to & Y_n
\end{array}$$

From a category theory perspective, you *lift* these functions from their home category into a *typed assembly* category:

$$\begin{array}{rcrcl}
\mathrm{lift}(f_0) & : & Z & \rightarrow & Z \\
\mathrm{lift}(f_1) & : & Z & \rightarrow & Z \\
 & & & \vdots & \\
\mathrm{lift}(f_{n-1}) & : & Z & \rightarrow & Z \\
\mathrm{lift}(f_n) & : & Z & \rightarrow & Z
\end{array}$$

where our typed assembly category object Z in effect can be defined as:

$$Z \quad := \quad X_0 \times \ldots \times X_n \times Y_0 \times \ldots \times Y_n$$

By *refactoring* the domains and codomains, we can now compose our lifted functions:

$$\mathrm{lift}(f_n) \circ \mathrm{lift}(f_{n-1}) \circ \ldots \circ \mathrm{lift}(f_1) \circ \mathrm{lift}(f_0)$$

This idea is borrowed from *register machine theory*, and in fact is how register machines are created if we were to explain how they work to an audience of mathematicians. The refactored common *signature* forms the registers, and *lifted composition* in effect is the continuation passing monad. Each lifted function represents an instruction. Finally, as we may desire *recursion* we extend the register space Z such that each *typed* register is given its own *stack* so you can save and restore values.

facade:	**x**	**y**		
signature:	**x** : **int**	**y** : **int**	**z** : **int**	
constant			2	// $z \leftarrow 2$
multiply		$\lambda_2, \rightarrow$	λ_1	// $y \leftarrow 2y$
square	λ			// $- \ \leftarrow x^2$
add		λ_2		// $- \ \leftarrow x + y$
return				// $x^2 + 2y$

This page is temporary to make editing easier.