CS 524 Lecture 8

José Meseguer

1. Natural Transformations

So, now that we know about functors, what is a natural transformation?

The definition can be given in three lines. But that does not mean that it can be understood in thirty seconds. I think that, before giving the definition, we can see some examples illustrating the general concept. Even better if the are CS examples anybody knows about.

The examples I will give are all about polymorphic functions, that anybody familiar with functional programs, or just with parameterized modules in Maude, can easily understand.

For example, in the Pair polymorphic type, which we could just write [define] as:

$$Pair(X,Y) =_{def.} \lambda\,(X,Y) \in \underline{Set} \times \underline{Set} \;.\; X \times Y$$

i.e., as the functor

$$\begin{array}{ccc} -\times- : \underline{Set} \times \underline{Set} & \longrightarrow & Set \\ (A, B) & & A \times B \\ (f,g)\downarrow & \mapsto & f \times g \downarrow \\ (A', B') & & A' \times B' \end{array}$$

The first and second projections $p_1 : X \times Y \to X$, $(x,y) \mapsto x$

$p_2 : X \times Y \to Y$, $(x,y) \mapsto y$ are polymorphic, i.e., for each

$(A,B) \in \mathbf{Set} \times \mathbf{Set}$ we get an actual function

$p_1(A,B) : A \times B \to A$, $(a,b) \mapsto a$, $p_2(A,B) : A \times B \to B$, $(a,b) \mapsto b$

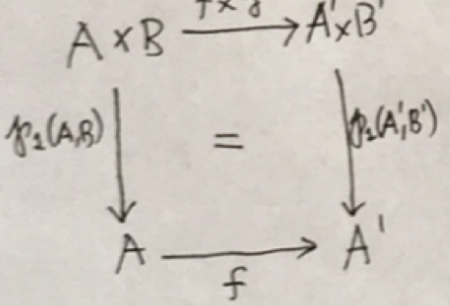
What does it mean that p_1 and p_2 are polymorphic?

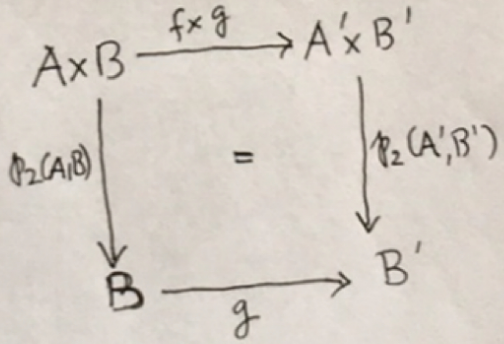
Of course, that the are families of functions

$$\{p_1(A,B)\}_{(A,B) \in \mathbf{Set}^2} \qquad \{p_2(A,B)\}_{(A,B) \in \mathbf{Set}^2}$$

but more than that: that they are coherent: do not involve any arbitrary choices for each (A,B), are natural in the parametric or polymorphic way the families have been chosen. This sounds all a bit woolly mumbo-jumbo. Can we cut through the BS?

Yes, of course:





It just means that p_1 and p_2 are operations of the polymorphic data type Pair(X, Y), and that for any $(f, g) : (A, B) \to (A', B')$ in $\underline{Set}^2$ the function $f \times g : A \times B \to A' \times B'$ is not just any function: it is a homomorphism preserving the data type operations p_1 and p_2 as instantiated in $A \times B$ and $A' \times B'$, respectively.

Let us look at a second example: the polymorphic type of non-empty strings:

$$\text{String}^+(X) = \lambda X \in \underline{Set}.\ X^* \in \underline{Set}$$

i.e., the functor [in fact a monad]

$$(\_)^+ : \underline{Set} \longrightarrow \underline{Set}$$

$$\begin{array}{ccc} A & \longmapsto & A^+ \\ f \downarrow & & \downarrow f^+ \\ B & & B^+ \end{array}$$

where, of course, $f^+ : A^+ \to B^+$

$$a_1 \cdots a_n \longmapsto f(a_1) \cdots f(a_n)$$

Three polymorphic functions, among others, for $\text{String}^+(X)$ are:

$first : X^+ \longrightarrow X$ $\quad x_1 \cdots x_n \longmapsto x_1$

$last : X^+ \longrightarrow X$ $\quad x_1 \cdots x_n \longmapsto x_n$

$reverse : X^+ \longrightarrow X^+$ $\quad x_1 \cdots x_n \longmapsto x_n \cdots x_1$

In Maude we would define them by the equations:

$first(x) = x = last(x)$

$first(x, \ell) = x = last(\ell\, x)$

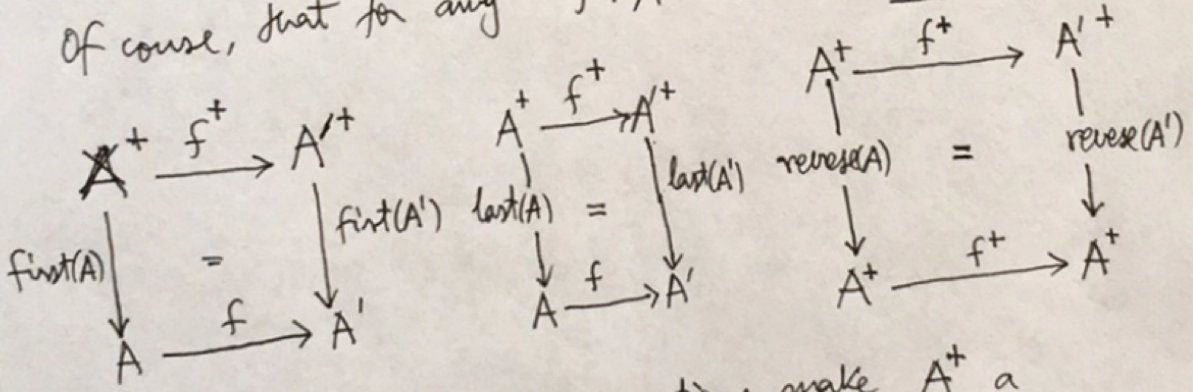
$reverse(x) = x$

$reverse(x\, \ell) = reverse(\ell)\, x$

where x ranges over the parameter X, and ℓ over X^+

What does it mean first, last and reverse are polymorphic?

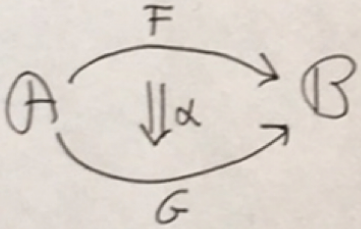
Of course, that for any $f : A \to A'$ in Set we have:



That is, the first and last operations make A^+ a two-sorted data type with sorts A^+ and A, and then, for any $f : A \to A'$ in Set, the two-sorted ~~homomorphism~~ function $(f, f^+) : (A, A^+) \longrightarrow (A', A'^+)$ in $\underline{Set}^2$ is a homomorphism preserving first, last, and reverse.

Can we say all this more succinctly? Yes, just use category theory!

Definition Let $\mathcal{A}$ and $\mathcal{B}$ be categories $\mathcal{A} = (O, A, \to_{\mathcal{A}}, -;-, id)$, $\mathcal{B} = (Q, B, \to_{\mathcal{B}}, -;-, id')$, and let $F, G : \mathcal{A} \to \mathcal{B}$ be two functors. Then, a natural transformation between F and G, denoted $\alpha : F \Rightarrow G$ and depicted



is a family of $\mathcal{B}$-arrows: $\alpha = \{\alpha(A) : F(A) \to G(A) \mid \alpha(A) \in B\}$ such that for any $f : A \to A'$ in $\mathcal{A}$ we have:

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(A') \\ {\scriptstyle \alpha(A)}\downarrow & = & \downarrow{\scriptstyle \alpha(A')} \\ G(A) & \xrightarrow[G(f)]{} & G(A') \end{array}$$

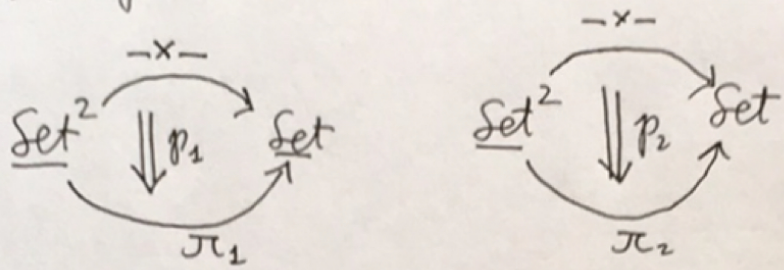
Examples:

1. Consider the projection functors

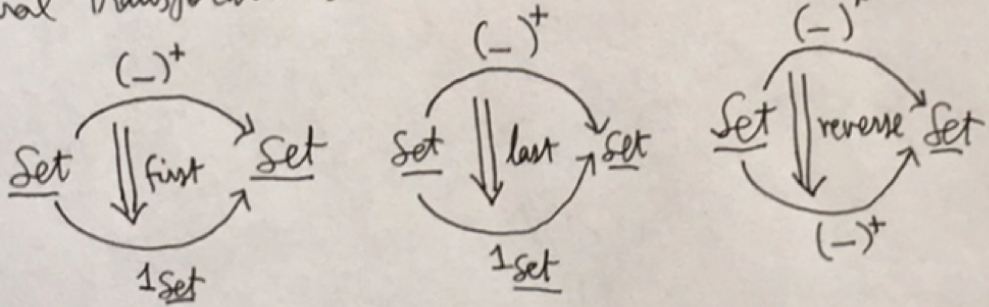
$$\pi_1 : \underline{Set}^2 \to \underline{Set} \qquad \pi_2 : \underline{Set}^2 \to \underline{Set}$$

$$\begin{array}{ccc} (A,B) & & A \\ (f,g)\downarrow & \mapsto & f\downarrow \\ (A',B') & & A' \end{array} \qquad \begin{array}{ccc} (A,B) & & B \\ (f,g)\downarrow & \mapsto & \downarrow g \\ (A',B') & & B' \end{array}$$

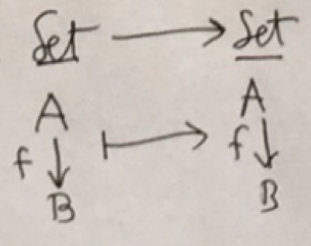
Then, the fact that p_1 and p_2 are polymorphic functions of the Pair(X,Y) [three-sorted] data type $[(X \times Y, X, Y)]$ is just the fact that we have natural transformations



2. Likewise, the fact that first, last, and reverse are polymorphic functions in the $String^+(X)$ [two-sorted] data type $[(X^+, X)]$ is just the fact that we have natural transformations:



where 1_{Set} is the identity functor 1_{Set} :



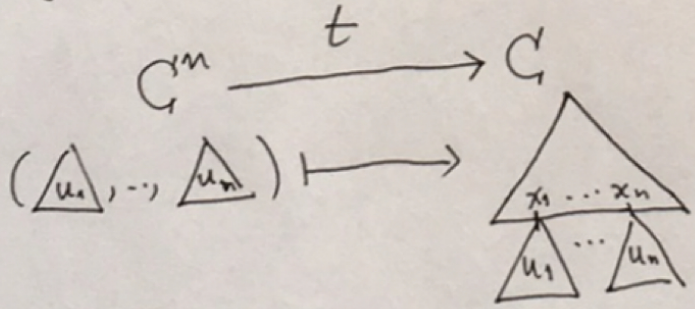
So, what does all this have to do with CL and the category $\mathcal{T}_{CL}$? Everything!

2. CL Reduction Rules as Natural Transformations

To simplify notation, let us assume that the variables ranging over CL are numbered as $X = \{x_1, x_2, \dots, x_n, \dots\}$ with $n \in \mathbb{N} - \{0\}$. This way, when we view a CL expression $t = t(x_1 .., x_n)$ as a function of n-arguments

$$t = \lambda (u_1 .., u_n) \in CL^n \cdot t\{x_1 \mapsto u_1, .., x_n \mapsto u_n\}$$

we have no problem making sense of which argument is which in the order of arguments. Recall that $\{x_1 \mapsto u_1, \dots, x_n \mapsto u_n\}$ is called a substitution, and then $t\{x_1 \mapsto u_1, .., x_n \mapsto u_n\}$ denotes the replacement of all occurrences of $x_1 .., x_n$ in t by, respectively, $u_1, .., u_n$. Graphically:



If we want to be more precise, CL may denote the CL-expressions without variables, and CL(X) the CL-expressions with variables $X = \{x_1, x_2, .., x_n, \dots\}$.

Then, for any $t \in CL(X)$, with $t = t(x_1, \ldots, x_n)$, we have a function

$$t : CL(X)^n \longrightarrow CL(X)$$

but, since $CL \subseteq CL(X)$, this restricts to a function

$$t : CL^n \longrightarrow CL$$

But recall that <u>application</u> is actually a <u>functor</u> [because of the <u>Functoriality</u> equation]

$$-- : \mathcal{T}_{CL}^2 \longrightarrow \mathcal{T}_{CL}$$

Therefore, by a trivial induction on n, any $t \in CL(X)$, $t = t(x_1, \ldots, x_n)$ defines a <u>functor</u>:

$$t : \mathcal{T}_{CL}^n \longrightarrow \mathcal{T}_{CL}$$

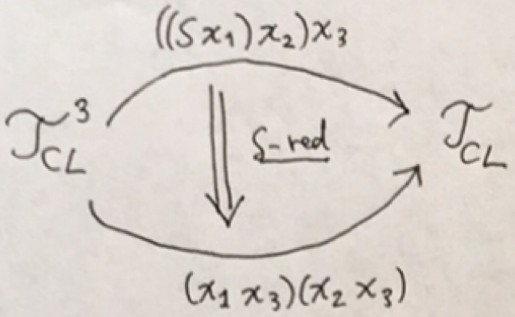
Consider now the S-red rule:

<u>S-red</u>: $((S\, x_1)\, x_2)\, x_3 \longrightarrow (x_1\, x_3)\, (x_2\, x_3)$

Then, the diagram in pg. 5 of Lecture 7b just means that:

Fact: $\underline{S\text{-red}} = \{ S\text{-red}(u,v,w) : ((S\,u)\,v)\,w \to (u\,w)(v\,w) \mid (u,v,w) \in CL^3 \}$

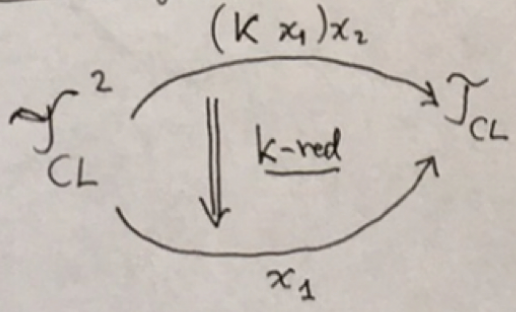
is a natural transformation



Likewise, again because of equivalences (5) and (6) in pg. 5 of Lecture 7 we have:

Fact $\underline{K\text{-red}} = \{ K\text{-red}(u,v) : (K\,u)\,v \to u \mid (u,v) \in CL^2 \}$

is a natural transformation



Fact $\underline{I\text{-red}} = \{ I\text{-red}(u) : Iu \to u \mid u \in CL \}$ is a natural transformation