2-Categories of Computation

We want to understand "the process of computation" in the λ-calculus via:

monoidal closed categories $\rightsquigarrow$ monoidal closed 2-categories

but first let's do something simpler:

monoids $\rightsquigarrow$ monoidal categories

A monoidal category is a 2-category with one object, just as a monoid is a category with one object, so we are already seeing 2-categories in this baby example.

We'll study "rewrite rules" for monoids. Consider a presentation P of a monoid:

$$P = \langle G \mid R \rangle$$

where G is some set of "generators" and R is some set of relations of the form

$$g_1 \cdots g_n = g_1' \cdots g_m' \qquad g_i, g_j' \in G$$

For example:

$$P = \langle a, b, c, d \mid ab = a,\ ac = da \rangle$$

From any presentation P we get a monoid M_P by forming the free monoid FG on the generators (consisting of

$\left\{\begin{matrix}\text{"words"}\\ \text{"strings"}\end{matrix}\right.$ in the $\left\{\begin{matrix}\text{"alphabet"}\\ \text{"symbols"}\end{matrix}\right.$ G) & modding out by the congruence relation generated by R. Then we get a puzzle: the <u>word problem</u>: can you tell if two words in G define the same element of our monoid. E.g.:

$$\text{does } dacbab = acca \text{ ?}$$

In general, this problem is not solvable by any algorithm. But some presentations are nice — sometimes we can solve the word problem by interpreting the relations as rewrite rules:

$$P = \langle a, b, c, d \; : \; ab \xrightarrow{\alpha} a \, , \; ac \xrightarrow{\beta} da \rangle$$

Now we're treating the relations as <u>morphisms</u> rather than equations. We can then try to check equations in M_P (solve the word problem) by applying the rewrite rules to both sides:

$$\begin{tikzcd}
 & dacbab \arrow[dl, "d\beta bab"'] \arrow[dr, "dacb\alpha"] & & acca \arrow[d, "\beta ca"] \\
ddabab \arrow[d, "dd\alpha ab"'] \arrow[dr, "ddab\alpha"] & & dacba \arrow[dl, "d\beta ba"] & daca \arrow[ddl, "d\beta a"] \\
ddaab \arrow[drr, "dda\alpha"'] & ddaba \arrow[dr, "dd\alpha a"] & & \\
 & & ddaa &
\end{tikzcd}$$

So $dacbab = acca$!

In the above example, things worked very nicely, but they don't always. Let's describe the Diamond Lemma, which clarifies what's nice about this example.

What did we actually do? We took P and thought of it as generators (objects) and rewrite rules (morphisms) and used P to generate a monoidal category, $\widetilde{M}_P$. Then we drew a diagram in $\widetilde{M}_P$. In more detail, $\widetilde{M}_P$ is the strict monoidal category freely generated by the objects G & morphisms R:

$$\begin{cases} \alpha : ab \longrightarrow a \\ \beta : ac \longrightarrow da \end{cases} \quad \text{really meant} \quad \begin{cases} \alpha : a\otimes b \longrightarrow a \\ \beta : a\otimes c \longrightarrow d\otimes a \end{cases}$$

We get the objects of $\widetilde{M}_P$ by forming all tensor products of objects in G (unparenthesized, since $\widetilde{M}_P$ is <u>strict</u>!). We get the morphisms by tensoring and composing the morphisms in R — subject to equations in the defn. of strict monoidal category, e.g. the interchange law:

$$\begin{tikzcd}
 & dacbab \arrow[dl, "d\beta bab"'] \arrow[dr, "dacb\alpha"] & \\
ddabab \arrow[rr, phantom, "="] \arrow[dr, "ddab\alpha"'] & & dacba \arrow[dl, "d\beta ba"] \\
 & ddaba &
\end{tikzcd}$$

← really means $1_{d\otimes a\otimes c\otimes b}\otimes\alpha$

In our particular example, $\widetilde{M}_p$ is "terminating"

Def: A category is <u>terminating</u> if there does not exist an infinite diagram in it like this:

$$x_1 \xrightarrow{f_1} x_2 \xrightarrow{f_2} x_3 \xrightarrow{f_3} \cdots$$

where no f_i is an identity morphism.

If we had used

$$\gamma : a \longrightarrow ab$$

instead of $\alpha : ab \longrightarrow a$, $\widetilde{M}_p$ would not be terminating:

$$a \xrightarrow{\gamma} ab \xrightarrow{\gamma b} abb \xrightarrow{\gamma bb} abbb \xrightarrow{\gamma bbb} \cdots$$

In our example, $\widetilde{M}_p$ has another nice property that lets us solve the word problem by blindly applying rewrite rules:

Def: A category is <u>confluent</u> if given any diagram in it like this:

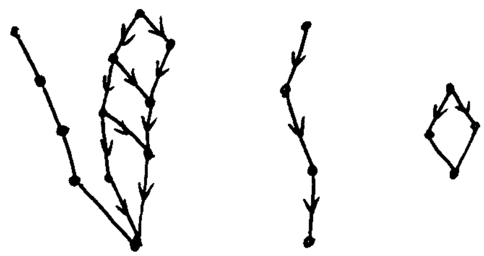
$$\begin{tikzcd} & x \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\ y_1 & & y_2 \end{tikzcd}$$

then $\exists\, z, g_1, g_2$ s.t. there is a <u>not</u>-necessarily-commuting diagram:

$$\begin{tikzcd} & x \arrow[dl] \arrow[dr] & \\ y_1 \arrow[dr] & & y_2 \arrow[dl] \\ & z & \end{tikzcd}$$

The "diamond property" or "Church-Rosser property

Diamond Lemma: A terminating confluent category is the coproduct (think "disjoint union") of categories with terminal objects.



So a terminating confluent category C is of the form $\sum_i C_i$ where C_i has a terminal object x_i. Given $x \in C_i$ we say x_i is its normal form: using rewrite rules however you want until you can't do any more, you get from x to x_i. So to solve word problems you just compare the normal forms.

So what's the relation between our monoid M_p & our monoidal category $\widetilde{M}_p$? Note $M_p \cong M_{p'}$ $\not\Rightarrow \widetilde{M}_p \simeq \widetilde{M}_{p'}$. But there is a map

$$\sigma : \widetilde{M}_p \longrightarrow M_p$$

Note: a monoid is the same as a monoidal category with only identity morphisms (the equations). σ is a monoidal functor that "squashes" all morphisms in $\widetilde{M}_p$ to identity morphisms.