Category Theory and Computational Complexity

Marco Larrea Octavio Zapata

December 23, 2014

1 Ponle Nombre a Tu Seccion wey

A first-order dependence logic D is a class which consists of all D-definable properties where $\mathsf{D} := (\mathsf{FO} + \mu.\bar{t})$ and $\mu.\bar{t}$ denotes that term $t_{|\bar{t}|}$ is functionally dependent on t_i for all $i \leq |\bar{t}|$. Recall that a P-definable property is a closed under isomorphism class P of structures. The model class of a first-order sentence φ is the class of structures $\mathsf{Mod}(\varphi) := \{A : A \models \varphi\}$ where the notation $A \models \varphi$ refers to the *'the snow is white' iff the snow is white* type of Tarskian semantics, of φ being true when interpreted in A under all α assignment functions (i.e. $(\forall \alpha \in [A]^{Var})\ A \models \varphi[\alpha]$ with Var some usable variables).

The model class FO is, as always, defined as the class of models of all first-order sentences (i.e. $\mathsf{FO} := \{S : (\exists \tau)(\exists \varphi \in L(\tau))\ S = \mathsf{Mod}(\varphi)\}$ where $L(\tau)$ is a first-order language of type τ) and $\mu.\bar{t}$ is interpreted as a recursively generated tuple of terms $(t_1, \ldots, t_{|\bar{t}|})$ which we naturally identify with the set $[|\bar{t}|] := \{1, 2, \ldots, |\bar{t}|\}$. D sentences are capable to characterise variable dependence and in general they are proven to be as expressive as the sentences of the second order Σ^1_1 fragment [**?**]. The intuitionistic dependence version ID has the same expressive power as full SO [**?**]. It is a fact that MID-model checking is PSPACE-complete [**?**] where MID is the intuitionistic implication fragment of the modal dependence logic MD which contains at least two modifiers. Hence, $(\mathsf{FO} + \mu.\bar{t}) = \mathsf{NP}, \mathsf{ID} = \Sigma_*\mathsf{P}$ and $\mathsf{MID} = \mathsf{PSPACE}$. On the other hand, $\mathsf{PSPACE} = \mathsf{IP} = \mathsf{QIP}$ [**?**], and so $\mathsf{MID} = \mathsf{QIP}$ which is the quantum version of the interactive polytime class IP.

We shall try to cook up a purely algebraic definition for the class of structures MID and extend such categorical logic in order to capture other quantum and classical complexity classes.

Ehrenfeucht-Fraïssé games characterise the expressive power of logical languages [**?**]. Every Ehrenfeucht-Fraïssé game is an ultraproduct [**?**]. A back-and-forth method for showing isomorphism between countably infinite structures. If F is an ultrafilter (i.e. $F \subseteq 2^{\mathbb{N}}$ and $\forall X \subseteq \mathbb{N}(X \notin F \Leftrightarrow \mathbb{N} \setminus X \in F)$ holds) then the reduced product $\prod_i M_i/F$ is an ultraproduct of the sets M_i, $i \in I$. Recall that

$$f \sim g \Leftrightarrow \{i \in I : f(i) = g(i)\} \in F$$

for all infinite sequences $f, g \in \prod_i M_i$ and any index set I is the relation which induces the equivalence classes that conform the ultraproduct

$$\prod_i M_i/F = \{[f] : f \in \prod_i M_i\}.$$

Lemma 1.1 (Łoś Lemma)**.** *If F is an ultrafilter and φ a first-order formula, then the ultraproduct of models of φ indexed by any index set $I \in F$ is a model of φ, i.e.*

$$\prod_i A_i/F \models \varphi \Leftrightarrow \{i \in I : A_i \models \varphi\} \in F.$$

2 Categorical Semantics of the Lambda Calculus

The $\lambda-$calculus is an abstraction of the theory of functions, in the same way group theory is an abstraction of the theory of symmetries. There are two basic operation on function we would like to formalize, *application* and *abstraction*.

Application refers to the operation performed by a function on a given term or expression. For example, if *double* is the function that multiplies by two, then for any given natural number n, we can

apply *double* to n to form the new natural number $double(n) = 2n$. Note that in order to be consistent one should define the type of arguments a function can take, for instance, it makes no sense to apply *double* to a string "*string*" of characters.

Abstraction is the operation of introducing new functions. Given a term t which (possibly) depends on a variable x, we can form a new function by abstracting the variable x from the term t in such a way that the application of this function on a term u is given by substituting in t the variable x by u. So for example, if we have the term $t = x * 2$ which depends on x, we form the function $\lambda x.t$ which extensionally is the same as the function *double* from above, that is $\lambda x.t(n) = double(n) = 2n$ for all natural number n.

The *simply-typed lambda calculus* is a form of type theory that interpretes the λ-calculus. Types are used in order to improve the consistency of the originally untyped theory.

The first step to define the simply-typed lambda calculus is to fix a set β whose elements we name *basic types* or *atomic types*. We express the fact that an object is a *type* by the judgment:

$$A \ type$$

We want every element of β to be a type, for this we introduce an *axiom* which is a special kind of *deduction rule* for which there are no assumptions. So for each $A \in \beta$ we have the rule:

$$\frac{}{A \ type}$$

which is read "A is a type". We'll also want to have a special type with only one term which we shall name the *unit type*:

$$\frac{}{1 \ type}$$