Category Theory and Computational Complexity

Marco Larrea Octavio Zapata

December 29, 2014

1 Introduction

1.1 First-order dependence

In this section every vocabulary τ will be referred as a similarity type or simply as a type.

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.

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 $i \leq |\bar{t}|$. 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 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 existential $\mathsf{SO}\exists$ model class [Vää07]. The intuitionistic dependence version ID has the same expressive power as full SO [Vää07]. It is a fact that MID-model checking is PSPACE-complete [Vää07] 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, $\mathsf{PSPACE} = \mathsf{IP} = \mathsf{QIP}$ [JJUW11], 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.

1.2 Ultraproduct

Ehrenfeucht-Fraïssé games characterise the expressive power of logical languages [Imm]. Every Ehrenfeucht-Fraïssé game is an ultraproduct [Vää11], a back-and-forth method for showing isomorphism between countably infinite structures, but only defined for finite structures in finite model theory. If F is an ultrafilter (i.e. $F \subseteq 2^{\mathbb{N}}$ such that $\forall X \subseteq \mathbb{N}(X \notin F \leftrightarrow \mathbb{N} \setminus X \in F)$ 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

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

This mathematical tool (the ultraproduct) is widely important because of results such as the following, from which proof we will delayed for the moment.

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.*

$$(\prod_i A_i/F, \alpha) \models \varphi \Leftrightarrow \{i \in I : (A_i, \alpha_i) \models \varphi\} \in F.$$

1.2.1 Ultrafilters

Given a nonempty set L define a binary relation $\succ$ on it by forcing the following formulas true:

1. Reflexivity: $\forall x(x \succ x)$
2. Transitivity: $\forall xyz(x \succ y) \wedge (y \succ z) \rightarrow (x \succ z)$
3. Antisymmetry: $\forall xy(x \succ y) \wedge (y \succ x) \rightarrow (x = y)$

The pair $\langle L, \succ \rangle$ is called a *poset* and $\succ$ a *partial order* on L. For every two-element subset $\{x, y\}$ of L we define its *meet* and *join* as $x \sqcap y := \inf_{\succ}\{x, y\}$ and $x \sqcup y := \sup_{\succ}\{x, y\}$, respectively. A *lattice* is a poset where every two-element subset has meet and join (i.e. $\forall\{x, y\} \in 2^L(\exists(x \sqcap y) \in L \wedge \exists(x \sqcup y) \in L)$ is satisfied).

A *filter* is a subset F of a lattice L which contains all the *successors* of any member of F (i.e. $\forall xy(y \in F) \wedge (x \succ y) \rightarrow (x \in F)$ holds). An *ultrafilter* U is a maximal filter with respect to the usual partial order relation that one can always define on any Boolean algebra (particularly on 2^L).

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*:

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

There are two introduction rules for types, these rules tell us how to construct new types from old ones. There is the introduction rule for *product types*:

$$\frac{A \ type \qquad B \ type}{A \times B \ type}$$

and the introduction rule for *function types*:

$$\frac{A \ type \qquad B \ type}{A \rightarrow B \ type}$$

Therefore the set of all types of the simply-typed lambda calculus is recursively generated from the set of basic types by applying the introduction rules of products and functions.

Now the set of types is defined we would like to define in a similar way the set of terms. As before we fix a set of *constant terms* or just *constants*. We also asume there are countable many variables (or as many as we might need), we'll name the variables $x, y, z, \ldots$. Just as types we will recursively generate the set of terms as follows:

$$t := [variables]|[constants]| * | < t, t' > |\pi_1 t|\pi_2 t|t(t')|\lambda x.t$$

References

[Ala13] Jesse Alama. The lambda calculus. http://plato.stanford.edu/archives/fall2013/entries/lambda-calculus/, 2013.

[APW13] Steve Awodey, Alvaro Pelayo, and Michael A. Warren. Univalence axiom in homotopy type theory. *Notices of the AMS*, 60(9):1164–1167, 2013.

[CF58] Haskell B Curry and Robert Feys. *Combinatory Logic, Studies in Logic and the Foundations of Mathematics*, volume 1. North-Holland, Amsterdam, 1958.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, Jun. 1940.

[Coq13] Thierry Coquand. Coq. coq.infra.fr, 2013.

[How95] W. A. Howard. The formuæ-as-types notion of construction. In *The Curry-Howard Isomorphism*. Academia, 1995.

[Imm] Neil Immerman. Descriptive complexity: a logician's approach to computation.

[JJUW11] Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. Qip = pspace. *J. ACM*, 58(6):30:1–30:27, December 2011.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. *arXiv preprint*, Nov. 2012.

[ML84] Per Martin-Löf. An intuitionistic theory of types: Predicative part. *The Journal of Symbolic Logic*, 49(1):311–313, Mar. 1984.

[Nor13] Ulf Norell. Agda. wiki.portal.chalmers.se/agda/pmwiki.php, 2013.

[Sch13] Urs Schreiber. Infinity groupoid. http://ncatlab.org/nlab/show/infinity-groupoid, 2013.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[Vää07] J. Väänänen. *Dependence Logic: A New Approach to Independence Friendly Logic*. London Mathematical Society Student Texts. Cambridge University Press, 2007.

[Vää11] J. Väänänen. *Models and Games*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2011.

[Voe06] Vladimir Voevodsky. Foundations of mathematics and homotopy theory. http://video.ias.edu/node/68, Mar. 2006.