Towards a mathematical science of programming

Noam Zeilberger

February 11, 2016

1 Overview

In many areas of science there is a tension between the desire to introduce new ideas to solve today's problems, and the ability to analyze these ideas using rigorous mathematical foundations. Usually this tension gives rise to a healthy "competition" between these two aspects of science, but sometimes one side so outweighs the other that it is difficult to make meaningful progress: good ideas get forgotten in the absence of a rigorous framework for explaining them, while overly rigid foundations can induce blindness to problems lying outside their formal scope. At such times, something is required in order to upset the balance of power – either bold new ideas, or bold new mathematical foundations (though often one closely follows the other).

For a long time, I have been convinced that the field of programming languages is at such an intellectual blocking point, in need of a major revision to its mathematical foundations. Category theory, a richly-developed branch of mathematics, has been successfully applied to explain some aspects of *typed* programming languages, yet it is far from an everyday tool in reasoning about computational systems or even in the analysis of type systems (as, say, algebra and differential equations are in the analysis of physical systems). Instead, programming language researchers usually develop formalisms on an ad hoc basis, and study them using techniques which remain somewhat isolated from the mainstream of mathematics, and difficult to communicate to outsiders. I believe that this is not an accident or due to purely social factors, but instead is a result of a fundamental mismatch between the intuitive conception of type systems and the way that they are usually explained via category theory.

In long-running joint work with Paul-André Melliès, I have been developing a categorical foundation for programming languages that rejects the established wisdom of interpreting a type system as a category of well-typed terms. A starting point for understanding our work is the idea that

functors are type refinement systems.

I am convinced that this basic idea provides the right conceptual basis for thinking about types through the lens of category theory, and indeed unifies type theory with other approaches to reasoning about computational systems (such as Hoare logic). Developing the applications of this framework – as well as understanding its limits – will take a serious and prolonged study.

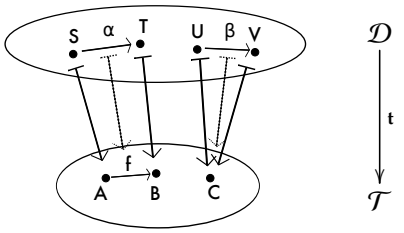
An advantage of this approach is that it provides new ways of looking at familiar objects, and one completely unexpected outcome of this work was the discovery of

a correspondence between normal planar lambda terms and rooted planar maps.

This is a striking new link between the field of programming languages and a very active area of combinatorics (enumeration of graphs on surfaces), which itself has far-reaching links to domains such as algebraic geometry and mathematical physics. Moreover, the early indications are that that this is not just a miraculous coincidence, but instead a sign of a deeper connection. Since the potential for knowledge transfer in both directions is real and quite exciting, I am eager to pursue such connections over the coming years.

2 Functors are type refinement systems

Type theory is a bit unusual as a mathematical theory in that the word "type" does not have a well-accepted meaning, and is actually used informally in two very different senses, sometimes called "types à la Church" and "types à la Curry" (or the *intrinsic* and the *extrinsic* views of typing by Reynolds [14]). One basic contribution of my paper with Paul-André Melliès, "Functors are Type Refinement Systems" [9], was to give a mathematical articulation of this informal distinction, and in particular to place typing "à la Curry" solidly within the framework of categorical logic. It is important to note that the standard dogma which interprets a typed programming language as a category of well-typed terms (see, e.g., Lambek and Scott [7]) is inherently biased towards the "types à la Church" view, indeed is *formally incompatible* with typing à la Curry. This is because by definition, in a category, any morphism f is intrinsically associated with a unique pair of types, namely its domain $\mathrm{dom}(f)$ and codomain $\mathrm{cod}(f)$. The idea alluded to in the title of our paper is to instead interpret a type system as a *functor* from a category of typing derivations to a category of terms:



In this picture, objects of $\mathcal{T}$ represent types intrinsically associated to terms (hence, "types à la Church"), while objects of $\mathcal{D}$ represent an extra layer of more refined typing information (hence, "types à la Curry"), with the type refinement relation formally represented by the functor $\mathbf{t}$. (So in the diagram above the types S and T refine the types A and B, respectively, while U and V both refine the same type C.) A *typing judgment* is defined as a triple (S, f, T) of a morphism f in $\mathcal{T}$ together with a pair of objects S and T in $\mathcal{D}$ which refine its domain and codomain, $\mathbf{t}(S) = \mathrm{dom}(f)$, $\mathbf{t}(T) = \mathrm{cod}(f)$. A *derivation* of a typing judgment (S, f, T) is just a morphism $\alpha : S \to T$ in $\mathcal{D}$ mapped by the functor $\mathbf{t}$ to the morphism f.

Now, on the one hand, many type systems can be naturally modelled this way as functors, with different features of the type system corresponding to different properties of the corresponding functor. For example, in [9] we showed that the property of being a *Grothendieck bifibration* [3] has a natural type-theoretic interpretation, with the categorical definition unwinding into a certain collection of typing rules for *image* and *inverse image* types. On the other hand, this idea can also serve as a working mathematical *definition* of "type system": indeed, *any* functor between two categories can be interpreted as a type system in a sufficiently abstract sense. Such a definition allows one to study type systems from a much more elementary perspective, and identify common patterns that occur across a range of computational settings even beyond the traditional scope of type theory. For example, in [9] we explained how Hoare logic may be considered as a type system in this sense, and used this fact to derive an analysis of the so-called "frame rule" in separation logic. In a more recent paper [10], we developed this approach considerably further by proving a suite of general representation theorems for type systems-as-functors (or "type refinement systems"), and in particular an Isbell-style duality theorem which we showed how to apply, for example, to the computation of strongest postconditions in Hoare logic. In our most recent article [11], we applied insights derived from this work to obtain a solution to a longstanding open problem in categorical logic originally set out by Lawvere, related to defining equality in the hyperdoctrine of presheaves.

These results are part of a longer term research project to develop practical mathematical foundations for the study of programming languages and proof systems. One of my original, personal motivations for beginning this line of work with Melliès over five years ago was to provide a rigorous framework in which to explain various intuitions I had developed during my thesis work [17], which connected phenomena that arise in the type-theoretic study of computational effects to the proof theory of linear logic. I am confident that this type of research will continue to provide a clarifying perspective on type theory within the wider world of mathematics and computer science, and pave the way for interesting and useful new discoveries.

3 A leap from λ-calculus into the theory of embedded graphs

One basic "folklore" observation in lambda calculus is that the β-normal forms can be characterized inductively, as a syntactic category (R) defined in mutual recursion with an auxiliary syntactic category (B) of "neutral" terms. This division shows up in many places in proof theory and programming languages, for example in proofs of strong normalization [6] and in the formulation of bidirectional type checking [4]. Based on ideas of Pfenning [13], it also provides a primordial example of a *refinement type signature*, with the syntactic categories R and B now seen as types refining the universal type U of pure lambda terms.

Motivated by my work with Melliès on the categorical semantics of type refinement systems (Section 2), I decided to study this signature in the *linear* case where every free or bound variable is used exactly once, since the categorical axiomatics for that situation are particularly simple and elegant. As a first step, I thought it could be useful to *count* normal linear lambda terms, with the hope that this would provide a guide towards the construction of interesting models. Using a simple recurrence formula, I obtained (in May 2014) the initial sequence

$$1, 2, 9, 54, 378, 2916, 24057$$

counting the number of distinct underlying *shapes* of closed normal linear lambda terms of a given size, where the shape of a lambda term abstracts away from its variable binding.

Quite surprisingly, this sequence was already listed in the OEIS [12] as entry A000168, and is a well-known series in combinatorics (see, e.g., [5, VII.8.2]) counting *rooted planar maps* by number of edges. Planar maps (= graphs embedded in a sphere) are a very old subject, but the idea of "rooting" a map was introduced in the 1960s by Bill Tutte in order to simplify the problem of counting isomorphism classes of maps. Tutte wrote two classic papers on the subject in the '60s [15, 16] where he even gave a beautiful closed formula for the number of rooted planar maps with n edges ($\frac{2(2n)!3^n}{n!(n+2)!}$). Since then, people have counted rooted maps on surfaces other than the sphere and with many different kinds of constraints – it's an active subfield of combinatorics that overlaps with numerous other areas of mathematics as well [8].

In an arXiv draft posted in August 2014 (later published in LMCS [18]), Alain Giorgetti and I gave a size-preserving bijection to account for this numerical coincidence by showing how to replay Tutte's 1968 analysis [16] of rooted planar maps in lambda calculus. Although the deeper meaning of this bijection is still not completely clear, one reason it suggests tantalizing directions for research is that rooted planar maps are naturally seen as a corner of a much larger mathematical universe. For example, there is an elegant, purely algebraic way of representing an embedding of a graph into an oriented surface called a *combinatorial map*. It is known that general linear lambda terms (considered up to α-equivalence) are in bijection with the subclass of *trivalent* rooted combinatorial maps [2, 20], and moreover this restricts to a bijection between shapes of linear lambda terms and trivalent rooted planar maps, giving us the following table:

linear lambda terms	rooted trivalent maps on oriented surfaces
shapes of linear lambda terms	rooted trivalent maps on the sphere
shapes of β-normal linear lambda terms	rooted maps on the sphere

Looking at the table, it is natural to ask about what happens when one considers β-normal linear lambda terms in general and not just their underlying shapes. I recently made important progress towards resolving this question, by proving [19] that if one counts certain *isomorphism classes* of closed normal linear lambda terms, then the sequence one obtains is precisely the sequence (A000698) counting rooted combinatorial maps by number of edges – and even better, the two-variable generating function $\tilde{L}_B(z, x)$ counting isomorphism classes of *neutral* linear lambda terms (by size and number of free variables) is equal to the two-variable generating function counting isomorphism classes of rooted maps on oriented surfaces (by number of edges and vertices) derived by Arquès and Béraud [1]. As a corollary, one can show that the generating function $\tilde{L}_R(z, 0) = z + 2z^2 + 10z^3 + 74^4 + \dots$ counting isomorphism classes of closed normal linear lambda terms satisfies the following remarkable continued fraction: $\tilde{L}_R(z, 0) = \cfrac{z}{1-\cfrac{2z}{1-\cfrac{3z}{1-\ddots}}}$.

Much more can be said, but let me conclude by mentioning that it is even possible to apply the correspondence between linear lambda terms and rooted trivalent maps to give a lambda calculus reformulation of the Four Color Theorem [20]. By all indications, we are just at the outset of an exciting journey!

References

[1] Didier Arquès and Jean-François Béraud. Rooted maps on orientable surfaces, Riccati's equation and continued fractions. *Discrete mathematics* 215:1–12, 2000.

[2] O. Bodini, D. Gardy, and A. Jacquot. Asymptotics and random sampling for BCI and BCK lambda terms. *Theoretical Computer Science*, 502:227–238, 2013.

[3] Francis Borceux. *Handbook of Categorical Algebra 2: Categories and Structures*. Cambridge University Press, 1994.

[4] Joshua Dunfield and Neel Krishnaswami. Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism. In *Proceedings of the 18th ACM SIGPLAN Int. Conf. on Functional Programming*, Boston, USA, 2013.

[5] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[6] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1990.

[7] Joachim Lambek and Philip Scott. *Introduction to Higher-order Categorical Logic.* CUP, 1986.

[8] Sergei K. Lando and Alexander K. Zvonkin. *Graphs on Surfaces and Their Applications*, Encyclopaedia of Mathematical Sciences 141, Springer-Verlag, 2004.

[9] Paul-André Melliès and Noam Zeilberger. Functors are Type Refinement Systems. In *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming*, Mumbai, India, 2015.

[10] Paul-André Melliès and Noam Zeilberger. An Isbell Duality Theorem for Type Refinement Systems. Submitted for publication. July 31, 2015. arXiv:1501.05115

[11] Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of Lawvere's presheaf hyperdoctrine. Submitted for publication. January 22, 2016. arXiv:1601.06098.

[12] OEIS Foundation Inc. (2011), The On-Line Encyclopedia of Integer Sequences. See entries A000168 and A000698.

[13] Frank Pfenning. Refinement Types for Logical Frameworks. In *Informal Proceedings of the Workshop on Types for Proofs and Programs* (ed. Herman Geuvers), 285–299, Nijmegen, The Netherlands, May 1993.

[14] John C. Reynolds. *Theories of Programming Languages.* CUP, 1998.

[15] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.

[16] W. T. Tutte. On the enumeration of planar maps. *Bulletin of the American Mathematical Society*, 74:64–74, 1968.

[17] Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*, PhD thesis, Carnegie Mellon University, Computer Science Department, April 2009.

[18] Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Logical Methods in Computer Science*, 11(3:22)2015:1-39, 2015.

[19] Noam Zeilberger. Counting isomorphism classes of β-normal linear lambda terms. Submitted for publication. September 25, 2015. arXiv:1509.07596

[20] Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. Submitted for publication. December 21, 2015. arXiv:1512.06751