Dependently-typed systems and their models.

Talk at IHP by Vladimir Voevodsky

June 6 , 2014

Today I will be thinking mostly in terms of what Erik Palmgren in his talk called "beta-models".

I would suggest to reserve the word "model" for such "beta-models" and use the word "interpretation" for "alpha-models".

By many of the speakers before me the phrase "model of type theory" was assigned provisional meaning by analogy with the concept "model of set theory".

A model of set theory (see some classical book on set theory) is a set together with some operations on this set.

So, by analogy the phrase "models of type theory" was used to denote some, as yet not fully determined, mathematical objects (usually expected to be categories with extra structures, maybe infty-categories with extra structure) which will play the same role for "the" type theory as models of set theory play for the set theory.

We see however, for example from the Steve Awodey list, that instead of one kind of mathematical structure which suffices for the discussion of models of set theory the work on models of type theory led to the discovery of many new kinds of mathematical structures.

I think that this fact is not a consequence of our inability to invent something but a forceful suggestion that type theory is different from the set theory and that the theory of models for type theory is much richer and more complex than the theory of models of set theory.

So in my talk today I will try to use a different terminology to emphasize that I am not thinking in the framework of the analogy "type theory"/"set theory".

Firstly, I will use the name "dependently-typed systems" for the class of objects which we are concerned with.

Secondly, I will use the phrase "theory of models of dependently-typed systems" not in the narrow sense suggested by the mathematical theory of models of set theory but in the sense of the field of mathematics which is concerned with the connections between the mathematical theory of dependently-typed systems and other areas of mathematics.

Let us consider the following examples:

- Is "the simple theory of types" in Church's formulation ("Church's type theory") a dependently-type system?

- Is the dependently-sorted first order logic of Makkai a dependently-typed system?

- Is logic-enriched system of Aczel-Gambino a dependently-typed system?

- Is the system suggested by Brunerie to describe infty-groupoids which has "Gamma is a contractible context" as a judgement a dependently-typed system?

To be able to answer such questions I suggest that we assume that the necessary components of any dependently-typed system are:

1. syntax with bound variables and alpha-equivalence
2. use of contexts, context extensions (weakening) and well-typed substitutions
3. inference rules
4. computation rules (?)

Using these criteria we see that "Church's type theory" is not a dependently-typed system (it has no contexts) while the remaining three examples are.

Do we have a clear (formalizable) understanding of what is each of the three components of a dependently-typed system?

I would argue that the answer is "yes" for the first component, "maybe" for the second and "not yet" for the third and the fourth.

We know what "syntax with bound variables and alpha-equalivalence" is thanks to the concepts of:

Dependent arity and dependent signature (Peter Aczel, published version in Aczel-Gambino, 2000).

A dependent arity with the set of expression-kinds EK is a pair which consists of a list of pairs (n,e) where n:nat and e:EK and an element of EK. We may write a dependent arity as (((n1,e1),...,(nl,el)),e). Let AG(EK) be the set of dependent arities.

A dependent signature with the set of symbols S and the set of expression-kinds EK is a (dependent) pair (S, ar:S ->AG(EK)) or, alternatively, a list of pairs of the form (s,alpha) where s:S and alpha:AG(EK).

Examples:

1. In the type system of Coq or more generally in type systems which use Russell style universes EK usually has only one element.

2. In the type systems where there is a clear distinction between objects and types and in particular where universes, if they are present, are Tarski style (e.g. in the description of Calculus of Constructions given in Streicher's book) EK is a two element set (e.g. {0,1}) with 0-expressions being objects and 1-expressions being types.

3. In the logic-enriched type theory of Aczel-Gambino one has EK={0,1,2} where 2-expressions are propositions "about" types and objects.

Given a dependent signature SA and two sets V and X one can define a set Exp0(SA, V, X) of expressions made from free variables from X using constructors from SA and bound-variable names from V as the set of labelled rooted trees satisfying some simple conditions on valencies and compatibility with labels from EK (the names of bound variables are attached to the symbol nodes as additional branches ending in leaves).

One can further define Exp1(SA,V,X) as the subset of Exp0(SA,V,X) which consists of trees satisfying the non-shadowing conditions for bound variables.

Then one can define the set Exp(SA,V,X) as the quotient of Exp0(SA,V,X) by alpha-equivalence.

The set Exp(SA,V,X) does not depend of V in the sense that permutations on V act on it trivially and we can denote it simply by Exp(SA,X).

Theorem. If S, EK and X have decidable equality then so do the sets Exp(SA,X).
Proof. De Bruijn indexes.

At this point we need to describe, in the similarly explicit and general manner what are the entities of our system of the next level - the ones directly composed from expressions. Here we are not even even sure yet what are contexts. Clearly a context contains the declarations of types of the free variables used in the expressions "under" this context.

But one also considers definitions, which provide not only the type but also the value of a variable.

May be definitions are actually computation rules and computation rules should be included in the context?

At this point let me assume a minimalistic position on contexts and say that any dependently-typed system should have in its contexts the type declarations for the variables and under contexts it should be possible to include the descriptions of objects of types.

Let me assume further that objects of types can be substituted for variables both in contexts and in the descriptions of other objects.

Having restricted myself to these features of dependently-typed systems I can perform an abstraction step.

Let me show that the only structure on the family of sets Exp(SA,-) which we need to describe the basic behavior of contexts and substitutions is the structure of a finitary monad.

A finitary monad on Sets is a monad which commutes with colimits. Since any set is, canonically, a colimit of a diagram of finite sets a finitary monad can be described as a collection of data (dependent tuple) of the following form:

1. A function Exp from finite sets to sets. Elements of Exp({x1,...,xn}) can be written as E(x1,...,xn).

2. A dependent function which assigns to finite sets X, X' and a function X' -> Exp (X) a function Exp(X') -> Exp(X) which we may call substitution and write as E(E1'(x1,...,xn),...,Em'(x1,...,xn)).

3. A collection of equations which these functions satisfy.

Note: the structure of the finitary monad on Exp(SA,-) will be sufficient for our discussion of B-systems but there are clearly other structures on Exp(SA,-) which arise from the fact that free variables can become bound variables through operations corresponding to symbols from S. These other structures are probably important for the formulation of a definition of what an inference rule is.

Question: how can we describe Exp(SA,-) as a free structure with operations given by SA?

Given a finitary monad M we can define a dependently-typed system as follows:

1. The set of names of free variables of the system is the set nat of natural numbers.
2. Contexts of the system Ctx(M) are sequences (E1,…,En) where Ei is an element of M({1,…,i-1}).
3. Type expressions in the context (E1,…,En) are elements of M({1,…,n})
4. Descriptions of objects of type E in the context (E1,…,En) are elements of M({1,…,n})
5. Weakening is achieved through shift of numbers (using the substitution structure of M).
6. Substitutions are achieved through the use of the substitution structure of M.

We add the inference rule saying that any context and any object description are derivable from the empty set of premises.

We say that the set of computations is empty.

If we had a good definition of what an inference rule is we could, at this point, to define a computation-free type system as a pair which consists of a dependent signature SA and a (finite) set of inference rules relating the contexts, type expressions in contexts and object descriptions in contexts for the finitary monad Ext(SA,-) **such that** the system of derivable contexts, types and objects generated by this set of inference rules is closed under weakening, substitution and such that variables declared in the context are derivable objects under the context.

Unfortunately I do not have a definition of what inference rule is at this point.

Still, we can advance one step further in mathematical theory of computation-free dependently-typed systems.

I think there is a growing agreement that, in the terminology of the set-theoretic foundations, the output of the next step should be an essentially algebraic theory together with a proof of the theorem that the objects defined through the syntactic means together with the operations provided by the inference rules give an initial model of this theory.

Even this however brings to us new questions which we have to address.

What is an essentially algebraic theory? Is a clear and precise definition (without using phrases such as "… and similarly for … ") given anywhere in the literature?

What is the notion of equality for essentially algebraic theories?

In what sense is a category with a functor from this category to sets "the same as" a model of an essentially algebraic theory which has along with the sorts of objects and morphisms of the category the sorts of objects with a distinguished section of the functor and of compatible morphisms?