Abstract type systems

Vladimir Voevodsky

Started June 17, 2011

Contents

1 Introduction

It is well known to many specialists in type theory how difficult it is to discuss type systems with mathematicians. One of the key reasons is that up to now the only answer to the natural question "And what exactly is a type system?" consisted in listing a number of more or less complicated examples. In this paper we give a formal definition of an abstract type system in the language of modern mathematics. Showing that a particular type system in the usual utilitarian sense is a type system in the sense of our definition requires proving a list of properties for the corresponding classes of contexts, judgements and reduction rules.

A particular type system is usually defined in two or three steps. First one defined a syntactic system of expressions with free and bound variables which underlies the type system. Then one defines the class of "valid" contexts and judgements of the type system which are, mathematically speaking, certain sequences (lists) of the expressions of the underlying syntactic system. Finally one defines a class of transformations between expressions called "reductions" or more generally "re-writing rules" which is used to consider certain contexts and judgements to be "definitionally equal" to each other. We will proceed by introducing abstract mathematical concepts corresponding to each step of this process.

2 Abstract systems of expressions

In this section we give a mathematical meaning to syntactic systems of expressions with free and bound variables. We start with the following definitions. Recall that a monad or triple on a category C is a functor $F : C \to C$ together with natural transformations $F \circ F \to F$ and $Id \to F$ which satisfy appropriate associativity and unity axioms (see e.g. [2, p.137]). Recall further that a functor between categories with filtered colimits is called cocontinuous if it preserves such colimits. We will be interested in particular in cocontinuous functors $F : Sets \to Sets$ from the category of sets to itself. Since any set is in a canonical way a filtered colimit of its finite subsets there is a natural equivalence between such functors and functors from the category $FSets$ of finite sets to the category of all sets.

Definition 2.1 *An abstract system of expressions is a cocontinuous monad E on the category of sets.*

We claim that any syntactic system of expressions such as a first order language, language of the λ-calculus etc. defines an abstract system of expressions in our sense and that this abstract system of

expressions contains a sufficient amount of essential information about the corresponding syntactic system to be used in the forthcoming definition of an abstract type system.

Let us start with the following construction. Let F, B, S and Q be four sets of which B is countably infinite. The sets F, S and Q can be arbitrary but in most examples they will be finite or countably infinite. Let further a be a function $S \amalg Q \to \mathbf{N}$. The reader is suggested to think of F as the set of possible names of free variables, B as the set of possible names of bound variables, S as the set of names of functional symbols and Q as the set of names of quantifiers. The value of a on $s \in S$ or $q \in Q$ is the number of "arguments" or "arity" of the corresponding function or quantifier.

For a finite rooted tree T we write $V(T)$ for the set of vertices of T, $val(v)$ for the number of edges leaving v and $v' < v$ if v' is above v (the root of the tree is assumed to be the lowest vertex) . For $v \in V(T)$ let $[v] = [v]_T$ be the subtree in T which consists of v and all the vertices above v. We say that vertices of T are labeled by elements of a set X if we are given a function $l : V(T) \to X$. We denote by $T(X)$ the set of finite rooted trees labeled by elements of X given together with the orderings on the sets of edges leaving each vertex ("planar trees").

Let $Exp_0(F, B, S, Q, a)$ be the subset of $T(F \amalg B \amalg S \amalg Q \times B)$ which consists of trees satisfying the following conditions:

1. if $l(v) \in F \amalg B$ then $val(v) = 0$,
2. if $l(v) \in S$ then $val(v) = a(l(v))$, if $l(v) = (q, x) \in Q \times B$ the $val(v) = a(q)$,
3. if $l(v) = (q, x)$, $l(v') = (q', x')$ and $v \neq v'$ then $x \neq x'$,
4. if $l(v) \in B$ then there exists v' such that $v' > v$ and $l(v') = (q, l(v))$.

The first condition says that only the leaves of the tree can be labelled by names of variables, the second has obvious meaning, the third requires all the names of all bound variables to be different from each other. This condition can be weakened but for our purposes such a strong but simple form is sufficient. Finally the fourth condition says that a name of bound variable can only appear above a quantifier which binds it.

Definition 2.2 *Two elements of $Exp_0(F, B, S, Q, a)$ are called α-equivalent if they can be obtained from each other by the change of labeling induced by a permutation on B ("renaming of bound variables").*

We let $Exp(F, S, Q, a)$ denote the set of equivalence classes in $Exp_0(F, B, S, Q, a)$ modulo α-equivalence. Note that this set does not depend up to a canonical isomorphism from the choice of B and we omit B from its notation.

Consider now $Exp(S, Q, a) := Exp(-, S, Q, a)$ as a covariant functor in F from $Sets$ to $Sets$ with the functoriality defined by the obvious change of names of free variables corresponding to a map $F \to F'$. The following proposition is straightforward although a detailed proof would take some space.

Proposition 2.3 *The functor $Exp(S, Q, a)$ is cocontinuous. The substitution of expressions for free variables defines a structure a monad on $Exp(S, Q, a)$.*

We have constructed an abstract system of expressions starting from a "signature" (S, Q, a). We will call such systems of expressions "free" or "syntactic" systems of expressions. Let us now consider some examples.

Example 2.4 [**lambda**] The set of α-equivalence classes of terms of the untyped λ-calculus with free variables from F is naturally isomorphic to $Exp(F, S, Q, a)$ where $S = \{eval\}$, $Q = \{\lambda\}$, $a(eval) = 2$ and $a(\lambda) = 1$. The monad structure corresponds to the substitution on λ-terms.

Example 2.5 [**propositional**]The set of expressions of propositional calculus with variables from F is naturally isomorphic to $Exp(F, S, Q, a)$ where $S = \{\neg, \vee, \wedge, \Rightarrow\}$, $Q = \emptyset$, $a(\neg) = 1$ and $a(\vee) = a(\wedge) = a(\Rightarrow) = 2$.

Example 2.6 [**multisorted**] Consider the language of multi-sorted first order logic with the set of sorts SR, set of generating predicates GP with arities given by a function $a : GP \to \mathbf{N}$ and the set generating functions GF with arities given by a function $GF \to \mathbf{N}$ which we also denote by a. Let $Q = \{\forall, \exists\}$ and $S = SR \amalg GP \amalg GF$. For $q \in Q$ set $a(q) = 2$ and for $s \in SR$, $a(s) = 1$.

Now we can embed the well formed expressions our language with free variables from F considered modulo the α-equivalence to $Exp(F, S, Q, a)$. Vertices which are labeled by $(\forall, x)$ and $(\exists, x)$ have valency two. For such a vertex v, the first branch of $[v]$ is one vertex labeled by an element of SR giving the sort over which the quantification occurs and the second branch is the expression which is quantified. Unlike in the previous examples not all elements of $Exp(S, Q, a)$ are well-formed expressions and, moreover, the subsets of well-formed expressions do not form a sub-monad. In this particular example we could still obtain a monad but now on sorted sets. However, in the more complex situation of dependent type systems the sorts themselves can be complex expressions involving variables and such an approach does not work. Instead it turns out to be convenient to consider the whole monad $Exp(S, Q, a)$ and distinguish "meaningful" expressions only on the next stage when we introduce the concepts of contexts and judgements.

Example 2.7 [**betareduction**] The following example provides us with an abstract system of expressions which is obtained from a syntactic one but which is not itself syntactic. Let us start with the system of expressions of λ-terms described in an example given above. Let us denote the corresponding sets of terms with free variables from F by $E\Lambda(F)$. Consider the equivalence relation on $E\Lambda(F)$ generated by the β-reduction of λ-terms. This equivalence relation is known to commute with substitution. Therefore, the quotient sets with respect to this relation again form a monad which is still cocontinuous. The corresponding abstract system of expression is denoted by $E\Lambda_\beta$. We can further add the η-reduction and obtain another system of expressions $E\Lambda_{\beta,\eta}$.

Syntactic systems of expressions admit a construction which does not apply to general systems of expressions but which is very important in formalizations of the descriptions of particular type systems. This construction was invented by de Bruijn (see [1]) in the context of λ-terms but applies equally well to any syntactic system of expressions.

If one wants to formalize the mathematics of the previous paragraphs using a proof assistant then formalization of Exp_0 is straightforward - it is just an inductive type easily written in terms F, B, S, Q and a. The first problem one will encounter is that our definition of $Exp(F, S, Q, a)$ required taking the quotient of, say, $Exp_0(F, \mathbf{N}, S, Q, a)$ by the α-equivalence relation. From the point of view of constructive mathematics taking a quotient set is a complicated operation. For example it can lead from a set (or type) with decidable equality to one where equality is undecidable. De Bruijn's construction provides an explicit embedding of $Exp(F, S, Q, a)$ into $Exp_0(F \amalg \mathbf{N}, \emptyset, S \amalg Q, \emptyset, a)$ and describes the substitution operation directly in the language of this representation. Let us give a short exposition of this important construction here.

Let E be an element of $Exp_0(F, B, S, Q, a)$. We will construct, following de Bruijn, and element $dB_F(E)$ in $Exp_0(F \amalg \mathbf{N}, \emptyset, S \amalg Q, \emptyset, a)$ from which one can construct an explicit representative of the α-equivalence class of E in $Exp_0(F, \mathbf{N}, S, Q, a)$. The construction is very simple.

The underlying planar tree remains the same. The labels from F and S remain the same. The labels from $Q \times B$ are replaced by their first components lying in Q. Consider now a leaf v labelled by an element x of B. By the fourth condition there exists a vertex v' upstream from v labelled by (q, x) for some $q \in Q$. By the third condition such a vertex is unique. Let us consider the unique path in the tree leading from v' to v and lets count the number of vertices on this path labelled by the quantifiers including v' itself. Let us now label v by the number which we obtain. The resulting labelled tree $dB_F(E)$ lies in $Exp_0(F \amalg \mathbf{N}, \emptyset, S \amalg Q, \emptyset, a)$. It is easy to explicitly describe this embedding and to reconstruct the α-equivalence class of E from $dB_F(E)$.

There is a slight modification of this construction which is used to establish a bijection dB between $Exp(\mathbf{N}, S, Q, a)$ and $Exp_0(\mathbf{N}, \emptyset, S \amalg Q, \emptyset, a)$. Let us fix a linearly ordered set B with elements $b_1, b_2, \ldots$ and consider $Exp(\mathbf{N}, S, Q, a)$ as a quotient of $Exp_0(\mathbf{N}, B, S, Q, a)$. Let us define a map from $Exp_0(\mathbf{N}, \emptyset, S \amalg Q, \emptyset, a)$ to $Exp_0(\mathbf{N}, B, S, Q, a)$ as follows. Let E be an element of the first set. Let N be the total number of vertices in E labelled by elements of Q. We choose an ordering on these vertices and add to their labels second components $b_1, \ldots, b_N$ using this ordering. Now consider a leaf v labelled by $n \in \mathbf{N}$. If the number of quantifiers m upstream of v is less than n then we re-label it by $m - n - 1 \in nn$. It will be a vertex corresponding to a free variable. Otherwise the number of quantifiers greater or equal to n and we may consider the quantifier which is n quantifiers upstream of v. It is labelled by some b_i and we re-label v with b_i. It will be a vertex corresponding to a bound variable.

References

[1] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem [MR0321704 (48 #71)]. In *Selected papers on Automath*, volume 133 of *Stud. Logic Found. Math.*, pages 375–388. North-Holland, Amsterdam, 1994.

[2] S. MacLane. *Categories for the working mathematician*, volume 5 of *Graduate texts in Mathematics*. Springer-Verlag, 1971.