A type system with two kinds of identity types

by Vladimir Voevodsky
at Institute for Advanced Study in Princeton, NJ.

February 25 , 2013

Motivating example - semi-simplicial types

If we let ST_n denote the type of semi-simplicial types of dimension n in universe $\mathcal{U}$ then for small n we have intuitively simple explicit definitions:

$$ST_0 := \{X_0 : \mathcal{U}\}$$

$$ST_1 := \{X_0 : \mathcal{U}; X_1 : \prod(x_0\, x_1 : X), \mathcal{U}\}$$

$$ST_2 := \{X_0 : \mathcal{U}; X_1 : \prod(x_0\, x_1 : X), \mathcal{U};$$

$$X_2 : \prod(x_0\, x_1\, x_2 : X_0)(x_{01} : X_1\, x_0\, x_1)(x_{02} : X_1\, x_0\, x_2)(x_{12} : X_1\, x_1\, x_2), \mathcal{U}\}$$

So far no one was able to write a definition for ST_n in Coq!

Semi-simplicial types (cont.)

Several approaches lead to constructions which at some point require some object expression to have type $T(m)$ while its actual type is $T'(m)$. Here m is a natural parameter and remarkably for each individual m one has $T(m) \stackrel{d}{=} T'(m)$.

However the length of the reduction sequence one needs to perform to connect $T(m)$ with $T'(m)$ grows with m. Therefore in the context where m is a variable T is not definitionally equal to T'.

Possible solution: to make it possible to prove definitional equality by induction.

Attempt 1: definitional η-rule for natural numbers

One of the first ideas is to add definitional "η-rule" for the type of natural numbers $\mathbf{N}$:

$$\frac{\begin{array}{l}\Gamma, n : \mathbf{N} \vdash f : T \\ \Gamma, n : \mathbf{N} \vdash g : T \\ \Gamma \vdash f[O/n] \stackrel{d}{=} g[O/n] : T[O/n] \\ \Gamma, n : \mathbf{N} \vdash (f \stackrel{d}{=} g : T) \Rightarrow (f[Sn/n] \stackrel{d}{=} g[Sn/n] : T[Sn/n]) \\ \Gamma \vdash a : \mathbf{N}\end{array}}{\Gamma \vdash f[a/n] \stackrel{d}{=} g[a/n]}$$

Problem with definitional η-rule for natural numbers

As was pointed out by Peter Lumsdaine, implication in the premises of an inference rule leads to a structure which is not essentially-algebraic and hence, need not have an initial algebra. Therefore it is likely that

no precise meaning can be given to such a rule.

Another approach - two identity types

Id - an "extensional" identity type satisfying the "reflexion principle" i.e. such that if $Id\, x_1\, x_2$ is inhabited then $x_1 \stackrel{d}{=} x_2$.

Paths - a "univalent" identity type such that the system is consistent with the univalence axiom relative to a inverse $\mathcal{U}$ and $Paths$.

Problem: Using eliminators J_{Id} and J_{Paths} it is easy to prove that Id and $Paths$ will be logically equivalent. It is further easy to show that an identity type Id satisfying the reflexion rule also satisfies UIP. Hence the system will not be compatible with the univalence.

Third ingredient - notion of fibrant types

The solution to the problem suggested by the standard univalent model. If we introduce exact equality Id then we can use it to form display maps mapped by the model to arbitrary (up to an isomorphism) functions between simplicial sets corresponding to the contexts. Unless all contexts are mapped to discrete sets most such functions will not be fibrations.

Solution: Two identity types Id and $Paths$ and new class of judgements $\Gamma \vdash T \quad Fib$.

A comment on notations

I will sometimes use a technical system of notation for expressions with bound variables:

$$[name](x_1.\ldots.x_{n_1}.S_1, \ldots, x_1.\ldots.x_{n_m}.S_m)$$

is an expression with the head label $name$ and branches $S_1, \ldots, S_m$. The prefix $x_1.\ldots.x_{n_i}.$ in front of S_i means that variable names $x_1, \ldots, x_{n_i}$ are bound in S_i.

For example, instead of $\prod x : T_1, T_2$ I will sometimes write $[\prod](T_1, x.T_2)$.

This system of notation is has the following two main advantages:

1. Allows for constructors with complex schemes of variable bindings.
2. Reflects the way in which α-equivalence classes are represented through de Bruijn indexes.

Homotopy type system - HTS

Disclaimer: While I feel that some important pieces fall into place in this definition some are still missing.

Structural rules

The usual structural rules together with the rule

$$\frac{\Gamma \vdash T \quad Fib}{\Gamma, x : T \triangleright}$$

Rules for dependent products

The usual rules for $\prod$, ev, λ with the rules for the propagation of the definitional equality and β- and η- rules. In addition the rule:

$$\frac{\Gamma \vdash T_1 \quad Fib \qquad \Gamma, x : T_1 \vdash T_2 \quad Fib}{\Gamma \vdash [\prod](T_1, x.T_2) \quad Fib}$$

Rules for exact equality types Id

The usual rules with Id as the name of the types, $refl$ as the name of the constructor and J as the name for eliminator together with the reflexion rule

$$\frac{\Gamma \vdash o_1 : X \quad \Gamma \vdash o_2 : X \quad \Gamma \vdash o : [Id](o_1, o_2)}{\Gamma \vdash o_1 \stackrel{d}{=} o_2 : X}$$

Rules for path equality types Paths

The type forming rule of the form

$$\frac{\Gamma \vdash X \quad Fib \qquad \Gamma \vdash o_1 : X \qquad \Gamma \vdash o_2 : X}{\Gamma \vdash [Paths](o_1, o_2) \quad Fib}$$

followed by the usual rule for the constructor named $idpath$ and the eliminator rule of the form:

$$\frac{\begin{array}{ll} \Gamma \vdash X \quad Fib & \Gamma \vdash o_1 : X \\ \Gamma, x : X, x' : X, e : [Paths](x, x') \vdash P \quad Fib & \Gamma \vdash o_2 : X \\ \Gamma, x0 : X \vdash s0 : P[x0/x, x0/x', [idpath](x0)/e] & \Gamma \vdash eo : [Paths](o_1, o_2) \end{array}}{\Gamma \vdash [J_F](X, x.x'.e.P, x0.s0, o_1, o_2, eo) : P[o_1/x, o_2/x', eo/e]}$$

together with the computation rule of the standard form.

Rules for the universe of all types $\mathcal{U}$

Usual rules for universe $\mathcal{U}$ which is itself a fibrant type, the element forming constructor El, the operation $forall$ corresponding to $\prod$ and id corresponding to Id together with the rule

$$\frac{\Gamma \vdash o : \mathcal{U} \quad \Gamma \vdash o' : \mathcal{U} \quad \Gamma \vdash [El](o) \stackrel{d}{=} [El](o')}{\Gamma \vdash o \stackrel{d}{=} o' : \mathcal{U}}$$

Rules for the universe of fibrant types $\mathcal{U}_F$

Usual rules for universe $\mathcal{U}_F$ which is itself a fibrant type, the element forming constructor El_F, the operation $forall_F$ corresponding to $\prod$ and $paths$ corresponding to $Paths$ together with the rule

$$\frac{\Gamma \vdash o : \mathcal{U}_F \quad \Gamma \vdash o' : \mathcal{U}_F \quad \Gamma \vdash [El_F](o) \stackrel{d}{=} [El_F](o')}{\Gamma \vdash o \stackrel{d}{=} o' : \mathcal{U}_F}$$

Rules for universe inclusion

$$\frac{\Gamma \rhd}{\Gamma \vdash [j] : [\prod](\mathcal{U}_F, x.\mathcal{U})}$$

$$\frac{\Gamma \vdash o : \mathcal{U}_F}{\Gamma \vdash [El_F](o) \stackrel{d}{=} [El][ev]([j], o)}$$

Resizing rules

$$\frac{\Gamma \vdash T \quad Fib \quad \Gamma \vdash p : Weq(T, [El_F](t'))}{\Gamma \vdash [rr0](T, t', p) : \mathcal{U}_F}$$

$$\frac{\Gamma \vdash T \quad Fib \quad \Gamma \vdash p : Weq(T, [El_F](t'))}{\Gamma \vdash [El_F][rr0](T, t', p) \stackrel{d}{=} T}$$

$$\frac{\Gamma \vdash T \quad Fib \quad \Gamma \vdash p : Ishprop(T)}{\Gamma \vdash [rr1](T, p) : \mathcal{U}_F}$$

$$\frac{\Gamma \vdash T \quad Fib \quad \Gamma \vdash p : Ishprop(T)}{\Gamma \vdash [El_F][rr1](T, p) \stackrel{d}{=} T}$$

The system described above is the smallest system where some interesting mathematics can be developed. One can also consider additional constructions discussed in the following slides.

The unit type and dependent sums

The inference rules are the usual ones. The η-rules saying that $x \stackrel{d}{=} pair\,(pr1\,x)\,(pr2\,x)$ for x in a dependent sum and $x \stackrel{d}{=} x'$ for x, x' in the unit type are derivable.

The unit type is fibrant and the dependent sum where both arguments are fibrant is fibrant.

There are "unit" as object of $\mathcal{U}_F$ and dependent sum operation on families of objects of $\mathcal{U}$ (reps. $\mathcal{U}_F$) parametrized by types given by object of $\mathcal{U}$ (reps. $\mathcal{U}_F$).

The empty type and disjoint union

The inference rules are the usual ones. The η-rules are derivable.

The empty type is fibrant and the disjoint union of two fibrant types is fibrant.

There are "empty" as object of $\mathcal{U}_F$ and obvious disjoint union operation on objects of $\mathcal{U}$ (reps. $\mathcal{U}_F$).

Recursive types

As far as I understand the material of the last section of the "Notes on Type systems" can be made precise in this type systems. Namely, there is a way to express any strictly positive inductive definition as a combination of the constructions already discussed with *parametrized W-types*. The later correspond to the inductive definitions of Coq of the form

Inductive IC(A:Type)(a:A)(B:A →Type)(D:forall x:A, (B x → Type))(q:forall x:A, forall y:B x, forall z: D x y, A):= c: forall b:B a, forall f : (forall d: D a b, IC A (q a b d) B D q), IC A a B D q .

and introduced by the inference rule

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A, y : B, z : D \vdash q : A}{\Gamma, w : [IC](A, a, x.B, x.y.D, x.y.z.q)\triangleright}$$

Recursive types (cont.)

The inference rules for the constructor and for the eliminator can be more or less copied from then form of the corresponding definition in Coq and the type of IC_{rect} printed by Coq. Similarly one can write down the computation rule (definitional equality associated with the eliminator) from the general such rules used in Coq.

Recursive types (cont.)

The type IC is fibrant if all the arguments are fibrant i.e. if one has

$$\Gamma \vdash A \;\; Fib \quad \Gamma, x : A \vdash B \;\; Fib \quad \Gamma, x : A, b : B \vdash D \;\; Fib$$

There are obvious versions of IC on the level of the universes $\mathcal{U}$ and $\mathcal{U}_F$.

In particular, in this system the datatypes such as the types of natural numbers or binary trees can be expressed through the W-types obtaining types which have both the computation rules obtained in the strictly positive formalism as well as recursive η-rules which allow to prove definitional equality by induction.

It will be very interesting to extend to this system the formalism of more general inductive definitions (such as inductive-inductive and inductive-recursive).

Multiple universes

The system of multiple universes connected by explicit inclusions as in the specification of TS0 is easy to add to HTS. There will be two universes for each universe - one for all types and one for fibrant types.

Univalence axiom

Univalence axiom can be formulated in the usual way. It should use path equalities and $\mathcal{U}_F$.

Implementation notes

The type system HTS is partly extensional. In particular derivability of a sentence (a context together with a judgement) is not decidable. However it is clear that a proof assistant based on HTS should keep enough additional data with the sentences to make independent decidable proof verification possible.

One approach to this issue is to consider the system formed by *extended sentences*. Extended sentences are sequences of expressions of the form

$$\Gamma \triangleright$$

$$\Gamma \vdash T \quad Fib$$

$$\Gamma \vdash p : o : T$$

$$\Gamma \vdash p : T = T'$$

$$\Gamma \vdash p : o = o' : T$$

Implementation notes (cont.)

The additional components p in these extended sentences are *secondary witnesses* which are constructed in such a way that there is a bijection between α-equivalence classes of extended sentences and derivation trees. Dan Grayson and I are currently working on an implementation of simple test typing system TTS with secondary witnesses which is formally defined in the note called TTS which is now in the wiki.

A conjecture

Note that the subsystem of HTS generated by the inference rules which do not mention $Fib, \mathcal{U}$ and Id is just the usual Martin-Lof type system. The following conjecture makes sense both assuming univalence axiom in both systems and without it.

Conjecture Let T be a type expression in the empty context in the Martin-Lof type system. Let o be an object of HTS of type T. Then there exists a Martin-Lof object o' of type T and an HTS object of type $[Paths](o, o')$.