Interpreting Univalent Type Theory in Nuprl

- Voevodsky proposed type theory with *univalence axiom* as a foundation for mathematics that could be checked by computer.
- He showed consistency by giving a model using *Kan simplicial sets*.
- His classical construction requires the axiom of choice so it was not known if a fully constructive model of univalent type theory could be made.
- Bezem, Coquand, & Huber made a model using *uniformly Kan cubical sets* working constructively in set theory (on paper).
- We formalized much of this model in Nuprl, but three main results, *composition for* $\Pi(A, B)$, *composition for* $\mathbb{U}$, and *univalence*, were too time consuming to finish (composition for $\mathbb{U}$ took twenty pages in Simon Huber's thesis).

Cubical Type Theory

- Cohen, Coquand, & Mörtberg made *cubical type theory* (cTT) and a model for it using a variant of the BCH model.
- In this model, composition for $\mathbb{U}$ follows more easily (than in BCH) using *composition for Glue*, a new type.
- Univalence is proved inside cTT using the other rules.
- Coquand suggested that we try to formalize the model of cTT in Nuprl.
- NSF funded us to do that. It took about one year and *composition for Glue* is the longest proof ever done in Nuprl.
- We now know that there is a fully constructive model of a univalent type theory.

Formalization Effort

1. lattice theory (87 definitions, 264 lemmas)
2. category theory (93 definitions, 183 lemmas)
3. presheaf models of type theory (70 definitions, 267 lemmas)
4. cubical type theory (348 definitions and 1,395 lemmas)
5. A very rough measure of the effort involved is the total number of *interactive proof steps* in the proofs. This is the number of user submitted tactics (some of them fairly long composite tactics) that are in the final proofs—so it does not count false starts, failures, and reorganizations. For the cubical type theory this total was 19,215.

Why use Nuprl?

- Coquand et. al. worked "on paper" using set theory with constructive logic.
- The first step is to build a *presheaf model* of Martin-Löf type theory.
- Coquand told me that such a model can't be defined in intensional type theory. (?)
- An *extensional* theory where the types are more like classical sets is needed (?). Nuprl is an extensional theory.
- Nuprl also has *quotient types* and *nominal logic* using *atoms*. Both were used in the BCH model, but turn out not to be essential.

Outline for this lecture

- What is univalence?
- What makes Nuprl an extensional theory.
- What is a *presheaf model MLTT*?
- What are (some of) the types in cubical type theory?
- What is *composition* for a type?

Univalence

- Singleton(T) $==$ $\exists x : T.\ \forall y : T.\ (y = x \in T)$
- For $f : A \to B$, PreImage(f,b) $==$ $a : A \times (f(a) = b \in B)$.
- Equiv(A,B) $==$ $\exists f : A \to B.\ \forall b : B.$ Singleton(PreImage(f, b)).
- Univalence: $\forall A, B : \mathbb{U}.$ Equiv(Equiv(A, B), $(A = B \in \mathbb{U})$)
- In cubical type theory we use $\mathrm{Path}_T(x, y)$ instead of $(x = y \in T)$.
- So univalence is $\forall A, B : \mathbb{U}.$ Equiv(Equiv(A, B), $\mathrm{Path}_{\mathbb{U}}(A, B)$)

Judgements of Nuprl's Type Theory

- The Nuprl *type system* inductively defines $T_1 = T_2 \in \text{Type}$ and $t_1 = t_2 \in T_1$. (Allen's PER semantics)
- *Only one kind of judgement*: $H \vdash C$ ext t
- Context H is $v_1 : T_1, v_2 : T_2, \ldots, v_k : T_k$
- Substitutions σ and τ are *consistent for* H if
  $\sigma(T_i) = \tau(T_i) \in \text{Type}$ and
  $\sigma(v_i) = \tau(v_i) \in \sigma(T_i)$
- $H \vdash C$ ext t is *true* if for all σ, τ consistent for H,
  $\sigma(C) = \tau(C) \in \text{Type}$ and
  $\sigma(t) = \tau(t) \in \sigma(C)$
- This definition of truth (formally defined in Nuprl-in-Coq) is very subtle. The meaning of equality in the type system is built in.

Extensionality in Nuprl

- Two uses of the adjective *extensional*
- *function extensionality*
- `H ⊢ f = g ∈(x:A → B)`
  `BY functionExtensionality !parameter{i:l} u`
  `H u:A ⊢ f(u) = g(u) ∈B[u/x]`
  `H ⊢ A = A ∈𝕌{i}`
  (when conclusion is equality, extract is always Ax so omitted.)
- *propositional equality = definitional equality*
- `H x:A, J ⊢ C ext t`
  `BY hyp_replacement #j B !parameter{i:l}`
  `H x:B, J ⊢ C ext t`
  `H x:A, J ⊢ A = B ∈𝕌{i}`
  (note that the extract does not change, no "transport")

Abstract Martin-Löf Type Theory

- (H $\vdash$) means H is a (well-formed) *context*
- (H $\vdash$ T) means T is a (well-formed) *type* in context H
- (H $\vdash$ t:T) means t is a (well-formed) *term* of type T
- Normal syntax like n:$\mathbb{N}$, n $\leq$ 17, v:Vector(n) $\vdash$ P(n,v)
- With *name free* syntax context is A.B.C and variables are replaced by *DeBruijn indices* (awkward but easier to model).
- n:$\mathbb{N}$, n $\leq$ 17, v:Vector(n) $\vdash$ P(n,v) becomes
  $\mathbb{N}$.(q $\leq$ 17).Vector(S(q)) $\vdash$ P(S(S(q)), q)
  (except S(q) is written (q)p, and S(S(q)) is ((q)p)p, etc.)

Categories in Nuprl

- To define *presheaf* in Nuprl we need categories, functors, natural transformations, etc.
- SmallCategory{i} ==

```
{cat:ob:Type{i} × ar:ob →ob →Type{i}
× x:ob →ar(x, x)
× x:ob→y:ob→z:ob →ar(x,y)→ar(y,z)→ar(x,z) |
let ob,ar,id,comp = cat
in ∀x,y:ob.  ∀f:ar(x,y).  comp(x,x,y,id(x),f) = f
                        ∧ comp(x,y,y,f,id(y)) = f
∧ ∀x,y,z,w:ob.∀f:ar(x,y)∀g:ar(y,z).∀h:ar(z,w).
                comp(x,z,w,comp(x,y,z,f,g),h) =
                comp(x,y,w,f,comp(y,z,w,g,h))
}
```

- Functors, natural transformations defined similarly.

Presheaf

- Instead of category Set, we use category of Nuprl types at level i
- `TypeCat{i} ==`
  `Cat(ob= Type{i};`
  `arrow(I,J)= (I ⟶ J);`
  `id(I)= λx.x;`
  `comp(I,J,K,f,g)= (g o f) )`
- Let C be any category.
- `Presheaf(C){i} == Functor(op-cat(C);TypeCat{i})`
- For example, the *representable presheaf* for `X ∈ ob(C)` is

  ```
  Yoneda(X) ==
  Presheaf(Set(I) = arrow(C)(I,X)
           Morphism(I,J,f,a) = comp(C)(J,I,X,f,a)
  )
  ```

Yoneda lemma

- *Yoneda embedding*:

```
Functor(ob(X) = Yoneda(X)
        arrow(X,Y,f) = A |→ λ g. comp(C)(A,X,Y,g,f)
        )
```

(A |→F(A) is display form for a natural transformation.)

- The *Yoneda lemma* states that this is full and faithful functor from C to Presheaf(C).
- Proof straightforward in Nuprl.
- Nuprl proof system tells us which rules were used. Both of Nuprl's extensional rules were used in proof of Yoneda lemma.

Presheaf models of MLTT

- Choose any base category C.
- $(H \vdash) == H \in \text{Presheaf(C)}$
- Context H is a family of Nuprl types $H(I)$ for $I \in ob(C)$ together with maps $\overline{f} : H(J) \to H(I)$ when $f : I \to J$ is an arrow in C.
- The *elements* of H are $I : ob(C) \times H(I)$. They are the objects of a category el(H).
- $(H \vdash T) == T \in \text{Presheaf(el}(H))$, so type T is another family of Nuprl types with maps between them.
- $(H \vdash t : T) == t$ assigns to (I, α) a member of $T(I, \alpha)$ and commutes with the maps. So term t is a family of Nuprl terms.

Other parts of a model of abstract MLTT

- Context map $\sigma : H \to G$ is a natural transformation from H to G. Such maps play the role of substitutions.
- Define how σ acts on types and terms. So if $G \vdash t : T$ then $H \vdash (t)\sigma : (T)\sigma$
- For $H \vdash T$, define context $H.T$, context map $p : H.T \to H$, and term q with $H.T \vdash q : (T)p$.
- Define types $\Pi(A, B)$, $\Sigma(A, B)$, and terms for pair, fst, snd, lambda, apply.
- Prove 39 rules of basic MLTT follow from these definitions.
- Every Nuprl type T becomes a *discrete* type in the presheaf model (i.e. a constant family).

Specialize general presheaf model to cubical type theory

- Use the base category CubeCat:
  - Objects are finite sets of names. We use fset($\mathbb{N}$) which is a quotient of list($\mathbb{N}$), but we could have used sorted lists and done without quotients.
  - Arrow (f:I $\to$ J) is a function from J to dM(I) the *free DeMorgan algebra* generated by the names in I. This lifts to a homomorphism f:dM(J) $\to$ dM(I) using freeness of dM(J), so we can compose arrows.
  - dM(I) is a free distributive lattice on I+I. We used quotient types to define general free distributive lattices, but for generators that have decidable equality, quotient types are probably not needed.
- The types dM(I) with lifted maps form a presheaf $\mathbb{I}$ over CubeCat, and gives a cubical type $\mathbb{I}$ (the *interval type*).

The Face Lattice

- In dM(I), for $i \in I$, there are generators inl(i) and inr(i) that we call $\langle i \rangle$ and $\langle 1 - i \rangle$.
- The *face lattice* fL(I) has the same generators, but we call them $(i = 0)$ and $(i = 1)$ and we form the free distributive lattice mod the constraints that $(i = 0) \wedge (i = 1) = 0$.
- The fL(I) with appropriate maps form a presheaf $\mathbb{F}$ over the CubeCat, and also a cubical type, $H \vdash \mathbb{F}$.
- If we were using variable names, we get formulas like $i : \mathbb{I},\ j : \mathbb{I} \vdash (i = 0) \vee (j = 1) : \mathbb{F}$
- A term ϕ where $H \vdash \phi : \mathbb{F}$ defines a constraint on the interval parts of context H.
- H, ϕ is a *subcontext* of H where ϕ "holds".
- In general, $G \subseteq H$ if $1 \in G \to H$.

Composition function for $G \vdash A$

- $H \vdash a : A[\phi \mapsto t]$ means $H \vdash a : A$ and $H, \phi \vdash a = t$.
- A *composition function* cA for type A in context G is a function such that for all contexts H and context maps $\sigma : H.\mathbb{I} \rightarrow G$
- For all formulas $H \vdash \phi : \mathbb{F}$ and terms $H, \phi.\mathbb{I} \vdash u : (A)\sigma$
- For any term a_0 such than $H \vdash a_0 : (A)\sigma[0][\phi \mapsto u[0]]$
- $a_1 = cA(H, \sigma, \phi, u, a_0)$ satisfies $H \vdash a_1 : (A)\sigma[1][\phi \mapsto u[1]]$
- cA must also satisfy a certain *uniformity condition*
- A big(!!) part of interpreting cTT in Nuprl is defining and proving correct the uniform composition functions for $\Pi(A, B)$, $\Sigma(A, B)$, (if ϕ then A else B), $(Glue[\phi \mapsto (T, f)]A)$, and $c\mathbb{U}$.

Composition for Glue

```
comp(Glue [phi ↦ (T, f)] A) == λH,s,psi,b,b0.
let a = unglue(b) in
let a0 = unglue(b0) in
let a'1 = comp (cA)s [psi ↦ a] a0 in
let t'1 = comp (cT)s [psi ↦ b] b0 in
let w = pres (f.1)s [psi ↦ b] b0 in
let st = (t'1 ∨ (b)[1]) in
let sw = (w ∨ <>((app((f.1)s; b)[1])p)) in
let cF = fiber-comp(H, ((phi)s)[1];((T)s)[1];((A)s)[1];
                       ((f)s[1]).1;a'1;(cT)s[1]; (cA)s[1]) in
let z = equiv (f)s[1] [(∀(phi)s ∨ psi) ↦ (st, sw)] a'1 in
let t1,pth = z in
let pth' = (pth)p @ q ∨ (a[1])p in
let a1 = comp (cA)s[1]p [(phi)s[1] ∨ psi ↦ pth' ] a'1 in
glue [((phi)s)[1] ↦ t1] a1
```

Details Online

www.nuprl.org/wip/Mathematics/cubical!type!theory