Toward naive type theories

David Ripley

University of Connecticut
http://davewripley.rocks

Types

Why type theory?

Type theories are a family of formalisms with a wide range of uses.

Invented to block paradox, they have taken on a life of their own.

They have long been used in the semantics of natural languages and programming languages.

Their connection to proofs ties them to formalized, constructive, and proof-relevant mathematics as well.

Types

Simple types and propositions

The connection to proofs turns on the ability to see propositions as types, and vice versa.

Think of each type *A* as the proposition '*A* is inhabited'.
Each inhabitant of *A* is a proof of it.

It is usual to suppose we can form function types:
given types A and B there is a type $A \to B$ of functions from A to B.

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A).M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Simultaneously a logic and a theory of functions.

(The logic is $\mathrm{Int}_{\to}$.)

Can add other connectives/type formers.

For example, conjunctions/pairs:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathtt{fst}(M) : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathtt{snd}(M) : B}$$

Can get to full propositional Int this way.

But to capture even moderately-complex reasoning,
we must go beyond propositional.

Types

Dependent types

How to say ‘Every natural number is ≥ 0’?

We want a function type with input type $\mathbb{N}$.
But what is its output type?

When given a natural $n : \mathbb{N}$, the function should produce a proof of the proposition that $n \geq 0$.

That is, the output type depends on the input.
(Not on the input's type; on the input itself!)

We need what's called a dependent function type.

Dependent function types are usually written with a $\prod$.

The proposition that every natural is ≥ 0, then, is the type

$$\prod_{n:\mathbb{N}} n \geq 0$$

Its inhabitants are functions that, when given an $n : \mathbb{N}$, yield a proof that $n \geq 0$.

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A)M : \prod_{(x:A)} B} \qquad \frac{\Gamma \vdash M : \prod_{(x:A)} B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

When x is not free in B, this is just $\to$ again.

But it can be free in B, and then this is something new.

Dependent function types give restricted universal quantification.

(Similarly, dependent pair types give
restricted particular quantification.)

So with dependent types, quantification is in the mix;
the resulting logic is much more expressive.

Types

Where do types come from?

This is all well and good as a way of forming new types from old.

But it doesn't get us anywhere without some types to start with.

One option (Martin-Löf's original formulation) is to start by assuming `Type` : `Type`.

This is surprisingly powerful.

If we have a type, we can assume a variable in it:

$$\frac{\Gamma \vdash A : \mathtt{Type}}{\Gamma, x : A \vdash x : A}$$

If we have $\mathtt{Type} : \mathtt{Type}$, then, we can assume our way to types.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathtt{Type} : \mathtt{Type}}}{X : \mathtt{Type} \vdash X : \mathtt{Type}}}{X : \mathtt{Type}, y : X \vdash y : X}}{X : \mathtt{Type} \vdash \lambda(y : X).y : \prod_{(y:X)} X}}{\vdash \lambda(X : \mathtt{Type}).\lambda(y : X).y : \prod_{(X:\mathtt{Type})} \prod_{(y:X)} X}$$

This gives the polymorphic identity function.

Given any type as an argument,
it returns the identity function on that type.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathtt{Type} : \mathtt{Type}}}{X : \mathtt{Type} \vdash X : \mathtt{Type}}}{X : \mathtt{Type}, y : X \vdash y : X}}{X : \mathtt{Type} \vdash \lambda(y : X).y : X \to X}}{\vdash \lambda(X : \mathtt{Type}).\lambda(y : X).y : \prod_{(X:\mathtt{Type})} X \to X}$$

This gives the polymorphic identity function.

Given any type as an argument,
it returns the identity function on that type.

Moreover, with $\texttt{Type} : \texttt{Type}$, the four judgment forms:

- A type
- $A = B$ type
- $a : A$
- $a = b : A$

can be reduced to the latter two.

Types

Meaning explanations

So `Type` : `Type` is convenient.

But can it be justified?

One attempt to provide an understanding of types:
Martin-Löf's meaning explanations.

ME1: To define a type A: say how an object of type A is formed, and how an equality between two objects of type A is formed.

ME2: Types A and B are equal when they have the same objects and the same equalities between objects.

ME1 examples:

- to give the type $\mathbb{N}$, we say how to form a natural and when two are equal
- to give the type $\mathbb{N} \to \mathbb{N}$, we say how to form a function from $\mathbb{N}$ to $\mathbb{N}$ and when two are equal
- to give the type $\mathtt{Type}$, we say how to form a type and when two are equal.

But ME1 and ME2 do say how to form a type
and when two are equal.

According to ME1, then,
the two together define `Type`,
the type of types.

The inhabitants of `Type` are then those things formed via ME1 and ME2.

Since `Type` is so formed, we have `Type` : `Type`.

Thus, `Type` : `Type` follows from these meaning explanations.

It is not only convenient, but justified.

Paradoxes and naivete

Paradox

Few existing dependent type theories allow `Type` : `Type`.

It tends to lead to paradoxes.

In Martin-Löf's original `Type` : `Type` system,
every type is inhabited.

That is, every proposition is provable.

So in applications to proofs, `Type` : `Type` is usually jettisoned.

One common replacement is a cumulative hierarchy.

Paradoxes and naivete

Set-theoretic paradox

This should all smell familiar to set-theoretic paradoxers.

Playing the role of the meaning explanations,
we have either Basic Law V or naive comprehension.

These are part of a fine story about sets
(sets are the extensions of predicates/open sentences),
but they lead to paradoxes.

Sophisticated set theories are built to avoid these paradoxes.

(Many involve cumulative hierarchies.)

There are heterodox traditions in set theory, though.

Some hold to Basic Law V or naive comprehension.
Paradoxes are either blocked or ameliorated by a shift in logic.

These can be split into four rough categories:

- Paraconsistent
- Paracomplete
- Noncontractive
- Nontransitive

Towards naive type theories

Where to make the change?

What I want to suggest, then, is a heterodox type theory.

The aim is an approach that holds to `Type` : `Type`,
while blocking or ameliorating the paradoxes elsewhere.

There is much to be learned from existing naive set theories.

But the type-theoretic situation is much more constrained.

The logic and the type theory can't be split off from each other;
the logic is built from the types in the first place!

There are fewer places to tinker,
and each place has broader ramifications.

Paraconsistent and paracomplete approaches work with distinctive theories of negation.

Here, negation is $\to \bot$.

Paracomplete theories work by blocking reductio.
So this must go:

$$\frac{\Gamma, x : A \vdash M : \bot}{\Gamma \vdash \lambda(x : A).M : A \to \bot}$$

Seems unpromising:
needs to restrict λ abstraction.

Analogs to other naive set theories need to reject:

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

How are we to see $\to$ as a function type, then?
We need some replacement for this rule.

Paraconsistent approaches have nothing to offer here.

This leaves noncontractive and nontransitive approaches.
Both must reject:

$$\frac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash N[M/x] : B}$$

Seems awkward:
can we really not substitute terms for variables?

According to nontransitive approaches, really.

This leaves noncontractive and nontransitive approaches.

$$\frac{\Gamma \vdash M : A \qquad \Delta, x : A \vdash N : B}{\Gamma\Delta \vdash N[M/x] : B}$$

Noncontractive approaches have another option.

A different option for function application as well:

$$\frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma\Delta \vdash MN : B}$$

Instead of sharing left sides, this rule combines them.

By taking special care to avoid using any assumption multiple times, these set theories avoid paradox-based trouble.

Can this adapt to type-theoretic paradoxes?

Noncontractive simple type theories are straightforward.

What about noncontractive dependent type theories?

Alas, here we reach the state of the art.

For now, I'm cautiously optimistic.

Appendix

Naive homotopy type theory?

Homotopy type theory adds an axiom to a dependently-typed base.

Given types A and B, we have further types
$A \simeq B$ and $A =_{\mathbf{Type}} B$.

Univalence is an inhabitant of $(A \simeq B) \simeq (A =_{\mathtt{Type}} B)$.

It is broadly like an extensionality principle.

Here's a worry:
noncontractive set theories don't do well with extensionality.

So it's possible that the kind of type theory I'm after
won't do well with univalence.

As usual, the devil is in the pudding.

It's the combination of three factors
that causes trouble in the set-theoretic case:

naive comprehension
extensionality
weakening

To see whether the same tricks can be pulled in the target type theory would require actually having the target type theory.

So: too early to say. But keep an eye on weakening.