Homotopy Type Theory in Isabelle

Joshua Chen
University of Nottingham
(prev. Innsbruck)

ITP '21

29 Jun 2021

Motivation

- Bring support for working with dependent types to the simply typed Isabelle prover.
- Why?
  - Update + improve on Isabelle/CTT [Pau20]
  - Modern case study on implementing a fully featured object logic with all the necessary infrastructure on top of a logical framework.
  - Lay groundwork for exploring logical frameworks as testbeds for prototyping type theories (c.f. Andromeda [Bau+21]; Isabelle more flexible with fewer guarantees).

The Isabelle logical framework

- Base logic Isabelle/Pure:

$$\text{rank-1 polymorphic simple type theory} + \\ \text{base type } \texttt{prop} \text{ of judgments} + \\ \text{implication} \Rightarrow\text{, universal quantification} \bigwedge \text{ and equality} \equiv.$$

- LCF-style proof assistant: kernel enforces that only valid `props` are derivable from a core set of axioms and inference rules.
- After setting up the logical rules, further infrastructure (typechecking, elaboration, definitions, proof methods, ...) is implemented using existing logical framework facilities, which all go through the kernel.

Embedding DTT into Isabelle/Pure

Intensional Martin-Löf type theory with ℕ-many cumulative Russell universes.

Type judgment

```
typedecl o

consts has_type :: ‹o ⇒ o ⇒ prop› ("(2_:/ _)" 999)
```

Universes

```
typedecl lvl

axiomatization
  O  :: ‹lvl› and
  S  :: ‹lvl ⇒ lvl› and
  lt :: ‹lvl ⇒ lvl ⇒ prop› (infix "<ᵤ" 900)
  where
  O_min: "O <ᵤ S i" and
  lt_S: "i <ᵤ S i" and
  lt_trans: "i <ᵤ j ⟹ j <ᵤ k ⟹ i <ᵤ k"

axiomatization U :: ‹lvl ⇒ o› where
  Ui_in_Uj: "i <ᵤ j ⟹ U i: U j" and
  U_cumul: "A: U i ⟹ i <ᵤ j ⟹ A: U j"

lemma Ui_in_USi:
  "U i: U (S i)"
  by (rule Ui_in_Uj, rule lt_S)

lemma U_lift:
  "A: U i ⟹ A: U (S i)"
  by (erule U_cumul, rule lt_S)
```

Embedding DTT into Isabelle/Pure

Type rule examples

```
axiomatization where
  PiF: "⟦A: U i; ⋀x. x: A ⟹ B x: U i⟧ ⟹ ∏x: A. B x: U i" and

  PiI: "⟦A: U i; ⋀x. x: A ⟹ b x: B x⟧ ⟹ λx: A. b x: ∏x: A. B x" and

  PiE: "⟦f: ∏x: A. B x; a: A⟧ ⟹ f `a: B a" and

  beta: "⟦a: A; ⋀x. x: A ⟹ b x: B x⟧ ⟹ (λx: A. b x) `a ≡ b a" and

  eta: "f: ∏x: A. B x ⟹ λx: A. f `x ≡ f" and

  Pi_cong: "⟦
    ⋀x. x: A ⟹ B x ≡ B' x;
    A: U i;
    ⋀x. x: A ⟹ B x: U j;
    ⋀x. x: A ⟹ B' x: U j
    ⟧ ⟹ ∏x: A. B x ≡ ∏x: A. B' x" and

  lam_cong: "⟦⋀x. x: A ⟹ b x ≡ c x; A: U i⟧ ⟹ λx: A. b x ≡ λx: A. c x"
```

```
axiomatization where
  SigF: "⟦A: U i; ⋀x. x: A ⟹ B x: U i⟧ ⟹ Σx: A. B x: U i" and

  SigI: "⟦⋀x. x: A ⟹ B x: U i; a: A; b: B a⟧ ⟹ <a, b>: Σx: A. B x" and

  SigE: "⟦
    p: Σx: A. B x;
    A: U i;
    ⋀x. x : A ⟹ B x: U j;
    ⋀p. p: Σx: A. B x ⟹ C p: U k;
    ⋀x y. ⟦x: A; y: B x⟧ ⟹ f x y: C <x, y>
    ⟧ ⟹ SigInd A (fn x. B x) (fn p. C p) f p: C p" and

  Sig_comp: "⟦
    a: A;
    b: B a;
    ⋀x. x: A ⟹ B x: U i;
    ⋀p. p: Σx: A. B x ⟹ C p: U i;
    ⋀x y. ⟦x: A; y: B x⟧ ⟹ f x y: C <x, y>
    ⟧ ⟹ SigInd A (fn x. B x) (fn p. C p) f <a, b> ≡ f a b" and
```

...

Embedding DTT into Isabelle/Pure

Note:

- Contexts are encoded as premises.

$$\frac{\Gamma \vdash A\colon U_i \qquad \Gamma, x\colon A \vdash b\colon B}{\Gamma \vdash \lambda(x\colon A).\, b\colon \Pi(x\colon A).\, B}\ \Pi_I$$

```
PiI: "⟦A: U i; ⋀x. x: A ⟹ b x: B x⟧ ⟹ λx: A. b x: ∏x: A. B x"
```

- Judgmental equality is shallowly embedded using Isabelle/Pure equality.

```
beta: "⟦a: A; ⋀x. x: A ⟹ b x: B x⟧ ⟹ (λx: A. b x) `a ≡ b a"
```

- Type families and function arguments to type eliminators are meta-functions.

```
SigE: "⟦
  p: Σx: A. B x;
  A: U i;
  ⋀x. x : A ⟹ B x: U j;
  ⋀p. p: Σx: A. B x ⟹ C p: U k;
  ⋀x y. ⟦x: A; y: B x⟧ ⟹ f x y: C <x, y>
  ⟧ ⟹ SigInd A (fn x. B x) (fn p. C p) f p: C p"
```

Short demo

Some Terminology

- *Theorem collection*: Data slot for storing `props` that have been certified by the kernel.
- *Schematic variables*: Metavariables. Prefixed with "?".
- *Resolution*: Given a goal P and an inference rule

$$\bigwedge \vec{x}.\ R_1(\vec{x}) \Rightarrow \cdots R_n(\vec{x}) \Rightarrow Q(\vec{x}),$$

  abstract P over $\vec{x}$, unify with $Q(\vec{x})$, and return the state with the new subgoals $\bigwedge \vec{x}.\ R_i(\vec{x})$ replacing P.

Type Checking

Isabelle's higher-order unification and resolution tactic + its simplifier can be used to implement a type checker.

We maintain theorem collections for type inference rules and known type judgments: `[form]`, `[intr]`, `[elim]`, `[comp]`, `[type]`.

Type Checking

To solve $(t\colon ?T)$ where the head of t is not schematic, we

- Unify the goal with some unconditional type judgment $s\colon S$ in `[type]`. If successful, we're done. Else,
- Resolve with a rule from `[form]`, `[intr]`, or `[elim]`. This is syntax-directed since the head of t is a constant. If successful, start from the top with each new subgoal in turn. Else,
- (Currently unimplemented) Attempt unfolding the definition of the head of t and start from the top. Else,
- Resolve with a unifying rule from `[type]`. This time, conditional rules are allowed. This is no longer necessarily syntax-directed since the user is allowed to add arbitrary rules to `[type]`, but backtracking is performed. If successful, start from the top. Else,
- Resolve with the change of direction rule $\bigwedge a\,A.\ a\colon A \Rightarrow A \equiv B \Rightarrow a\colon B$, and on the two newly arising subgoals run the typechecker and the simplifier, respectively.

Typechecker also hooked in to the simplifier to discharge ancillary typing conditions.

Implicits and Elaboration

- Implicits are metavariables that are to be elaborated.
- They appear everywhere, so in general proofs are always in "schematic mode".
- Isabelle's default goal statements don't support these very well—even the `schematic_goal` command converts metavariables to fixed free variables in crucial cases!
- Had to define modified goal statement commands (`Theorem`, `Lemma` etc.) to keep metavariables across subgoals, and also to hook the typechecker in to elaborate proof premises. (Example: horiz_pathcomp)

Definitions

- Currently, rudimentary “definitional package”: basically a modified version of the goal commands.
- No syntactic support for pattern matching or recursion yet (such cases are defined by constructing the terms manually).
- Implicit arguments denoted by `{}`. A syntax phase translation converts these to schematic variables in goal statements. (Example: idtoeqv)

Induction

Some work needed to integrate the propositions-as-types paradigm with the Isar structured proof style.

Example: Natural numbers

$$\frac{n\colon \mathbb{N} \qquad c_0\colon C(0) \qquad k\colon \mathbb{N},\ c\colon C(k) \vdash f(k,c)\colon C(\mathsf{suc}(k))}{\mathsf{NatInd}(C, c_0, f, n)\colon C(n)}\ \mathsf{NatE}$$

Induction

Lemma
  assumes "$n\colon \mathbb{N}$" and "$2 \mid n$"
  shows "$2 \mid \mathsf{suc}(\mathsf{suc}(n))$"

By induction on n, but naive application of NatE results in inductive goal

$$2 \mid \mathsf{suc}(\mathsf{suc}(k)) \Longrightarrow 2 \mid \mathsf{suc}(\mathsf{suc}(\mathsf{suc}(k)))$$

Want to induct on "$2 \mid n \Longrightarrow 2 \mid \mathsf{suc}(\mathsf{suc}(n))$" instead!

Induction

Push context assumptions involving n into the goal type:

$$\prod_{r\,:\,2|n} \left[2 \mid \mathsf{suc}(\mathsf{suc}(n))\right]$$

and *then* apply NatE (pull them out again after).

General `elim` tactic does this. Equality induction eq is a special case.

Next steps and Future possibilities

- Improve the type checker: better normalization and definitional unfolding (currently still needs a lot of manual tweaking to make computations go through)
- Universe level inference
- Definition infrastructure: more convenient pattern matching, inductive type definitions, HITs...

Might need to revisit some earlier design decisions (untyped Isabelle/Pure equality, Russell universes...)

Prototype two-level type theory?

References I

[Bau+21] Andrej Bauer et al. *Andromeda 2*. 2021. URL: https://www.andromeda-prover.org/ (visited on 28/06/2021).

[Pau20] Lawrence C. Paulson. *Constructive Type Theory*. 2020. Chap. 5, pp. 63–86. URL: https://isabelle.in.tum.de/website-Isabelle2020/dist/Isabelle2020/doc/logics.pdf.