Adventures in Impredicative Semantics

Programming and Proving in Cedille

Aaron Stump
Computer Science
The University of Iowa
Motivation and background for Cedille
A little history
System F (Girard, Reynolds, early 1970s)

1969 Mercury Cyclone Spoiler II
System F (Girard, Reynolds, early 1970s)

- $\forall X : \ast. T$
- Raw power (impredicativity!)
- A little crude (no Curry-Howard)

1969 Mercury Cyclone Spoiler II
Calculus of Constructions (Coquand, Huet 1988)
Calculus of Constructions (Coquand, Huet 1988)

- Add dependent types: $\Pi x : T. T'$
- Imported from Automath/Martin-Löf type theory
- Curry-Howard!
- No induction. [Geuvers 2001]
1992 Hoffman-Markley Streamliner
Calculus of Inductive Constructions (Werner 1994)

- Add primitive inductive types
- Finally ready for constructive mathematics!
- Basis for Coq

1992 Hoffman-Markley Streamliner
But Coq ≠ CIC

- Coinductive types
- Universe hierarchy (Extended CC, Luo 1990)
- Proof-irrelevant universe $\text{Prop}$
- And we might want more:
  - definitional proof irrelevance
  - inductive-inductive types
  - inductive-recursive types

Similarly, Agda ≠ MLTT.
Issues and limitations, Coq and Agda

- No formal semantics/correctness proof
  - Despite a lot of interest: TT in TT

- (Hence!) bugs and surprises
  - Incompatibilities with various axioms
  - Actual contradictions!
  - Type soundness broken in Coq

- Commitment to a set of datatypes
  - Theory of datatypes not finished...
  - E.g., higher-order abstract syntax prohibited
Have we created a monster?

Schaufelradbagger 258
If I could turn back time...

Good-bye to:

▷ primitive datatypes
▷ (also universe hierarchy, my bias)

Hello to

▷ lambda-encodings of data
If I could turn back time...

Good-bye to:

▷ primitive datatypes
▷ (also universe hierarchy, my bias)

Hello to

▷ lambda-encodings of data
Wanted: a new type theory

where

▷ inductive datatypes are derived (lambda-encoded)
▷ impredicativity is central
▷ core theory is small and verifiable

Tooling goals:

▷ see all typing/inference information
▷ predictable inference
▷ elaborate to core with independent checker
∀ x : T. T′ implicit products (Miquel)
ι x : T. T′ dependent intersections (Kopylov)
{ t ≃ t′} untyped equality

▷ Small theory, formal syntax and semantics
▷ Core checker implemented in < 1000loc Haskell
▷ Logically sound
▷ Turing complete(!)
▷ Supports inductive lambda-encodings
Back the truck up
Back the truck up

Did you say lambda encodings?
Not your forebear’s lambda encodings

- Usual rap: inefficient accessors
- Corrected by Parigot 1988 for typed encoding
- **Perfect** untyped encoding Böhm et al. 1994
  - linear space
  - constant-time accessors
  - intrinsic support for iteration
- Cedille: perfect inductive (typed) encodings
How are inductive datatypes defined?

- Several variations (CPP ’18, ITP ’18), one theme: The type of \( d \) expresses an induction principle for \( d \)
- For Nat:

\[
n : \forall P : \text{Nat} \to \star. (\forall x : \text{Nat}. P x \to P (S x)) \to P \ Z \to P \ n
\]

- Essentially due to Leivant 1983
- With D. Firsov, generic derivations for classes of \( F : \star \to \star \)
What do we get from this?

- Freedom
- No pre-set datatype class
- Explore semantics of advanced datatypes
- Power of impredicativity
- So far: Functorial, Monotone, IR, II
What do we get from this?

Freedom

- No pre-set datatype class
- Explore semantics of advanced datatypes
- Power of impredicativity
  - So far: Functorial, Monotone, IR, II
What do we get from this?

**Freedom**

- No pre-set datatype class
- Explore semantics of advanced datatypes
- *Power of impredicativity*
- So far: Functorial, Monotone, IR, II
So which car are we?
So which car are we?
So which car are we?

High-altitude type-theory exploration
Terrestrially: Cedille 1.1

▶ Datatype notations convenient!
▶ Cedille 1.1 adds them
▶ With elaboration to Cedille Core
▶ Histomorphomic recursion
  ▶ subsumes nested patterns
  ▶ can iteratively match on pattern variable x,
  ▶ and then make a recursive call
  ▶ division (iteratively take predecessor)
Architecture of Cedille

- `.ced files`
- `.cdle files`
- `Emacs mode`
- `Backend`
- `Cedille core`
- `Ok`
- `Error`
data Nat: * =
  | zero: Nat
  | suc: Nat → Nat.

pred : Nat → Nat
  = λ n. μ' n \{zero → n | suc n → n\}.

add : Nat → Nat → Nat
  = λ m. λ n. μ addN. m \{zero → n | suc m' → suc (addN m')\}.

add-assoc : ∀ x : Nat . ∀ y : Nat . ∀ z : Nat .
  \{ add x (add y z) = add (add x y) z \} =
  λ x . λ y . λ z .
  μ add-assoc . x @ (λ x : Nat . \{ add x (add y z) = add (add x y) z \})
  \{ zero → β
  | suc x' → p+ (add-assoc x') - β \}.
data Nat: * = 
  | zero: Nat 
  | suc: Nat → Nat.

pred : Nat → Nat 
  = λ n. μ m' {zero → n | suc n → m}.

add : Nat → Nat → Nat 
  = λ m n. μ addN. m {zero → n | suc m' → suc (addN m')}.

add-assoc : Π x : Nat . ∀ y : Nat . ∀ z : Nat . 
  { add x (add y z) = add (add x y) z } 

6U:--- nat.ced<mar8-2019> 3% (11,4) (cedille navi) 10:38AM 0.42

add = λ m : Nat . λ n : Nat . IndFixM : NatF -NatFmap m · (λ _ : Nat . Nat) (λ Type-addN : * . λ to : Cast · Type-eaddN · (FixM · NatF Natmap)). λ out : Π o : Π _ : Type-addN . NatF · Type-addN . { o = outFix }. λ addN : Π x : Nat . Type-addN . { isType-addN = λ Y : * . λ x : Π _ : Π i : Π _ : Type-addN . Nat . λ y : NatF · Type-addN. { i = outFix } } . Y . x to out } - (φ ρ β < y > { λ x-x . x-x } @ x-x . { NatIndF y = x-x } - β < NatIndF y > { λ x-x . x-x }) - NatIndF · Type-addN y { y = y' } · (λ y'' : NatF · Type-addN . ∀ y'' : Nat . ∀ _ : { inFix y'' = y'' } . Nat) (λ y'' : Nat . λ e : { inFix (λ zero' · λ suc' · zero') = y'' } . ρ ρ e @ x . NatF · Type-addN . λ y'' : Nat . λ e : { inFix (λ zero' · λ suc' · suc' m') = y'' } . ρ ρ e @ x . Nat - suc (addN m')) - (inFix · NatF · Natmap (cast · (NatF · Type-addN) · (NatF · (FixM · NatF Natmap))) - (Natmap · Type-addN · NatF · Type-addN · Nat) · (isType-addN · (cast · Type-addN · Nat) (λ to : Π _ : Type-addN . Nat . { i = x . x } . λ out' : Π i : Π _ : Type-addN . NatF · Type-addN . { i = out' } )) · (Π Nat . Π _ : Type-addN .Nat . { i = x . x } . λ out' : Π i : Π _ : Type-addN . NatF · Type-addN . { i = out' }) · 6U:--- nat.cdle 100% (??,336) (Fundamental cdle) 10:38AM 0.42

Checks!
module cov-pattern-matching.
import datatypes.

predCV : \forall N: * . Is/Nat \cdot N \Rightarrow N \rightarrow N
  = \lambda N. \lambda is. \lambda n. \mu '<is> n {zero \rightarrow n | suc n \rightarrow n}.

minusCV : \forall N: * . Is/Nat \cdot N \Rightarrow N \rightarrow Nat \rightarrow N
  = \lambda N. \lambda is. \lambda m. \lambda n. \mu mMin. n {
    | zero \rightarrow m
    | suc n \rightarrow predCV -is (mMin n)
  }.
minus = minusCV -is/Nat.

lt : Nat \rightarrow Nat \rightarrow Bool
  = \lambda m. \lambda n. \mu' (minus (suc m) n) {zero \rightarrow tt | suc _ \rightarrow ff}.

ite : \forall X: *. Bool \rightarrow X \rightarrow X \rightarrow X
  = \lambda X. \lambda b. \lambda t. \lambda f. \mu' b {tt \rightarrow t | ff \rightarrow f}.

div : Nat \rightarrow Nat \rightarrow Nat
  = \lambda m. \lambda n. \mu divN. m {
    | zero \rightarrow zero
    | suc m \rightarrow
      [m' = to/Nat -isType/divN m] -- (1)
      [difMN = minusCV -isType/divN m (pred n)] -- (2)
      ite (lt (suc m') n) zero (suc (divN difMN)) -- (3)
  }.