Dependent types I

Guest lecture (Mark Barbone)

Previously in 4110

$$\frac{\vdots}{\Gamma, \alpha \text{ type} \vdash \lambda x.x : \alpha \to \alpha}$$

^^^^^^^^^^ a type…

referencing the context! ^^ ^^

→ “α type” is second-class: expressions can’t appear in types

→ Strict separation between types and values

What about mixing terms and types?

$$\frac{\vdots}{\Gamma, n : \mathtt{nat} \vdash e : A(n)}$$

^^^^^^^^^

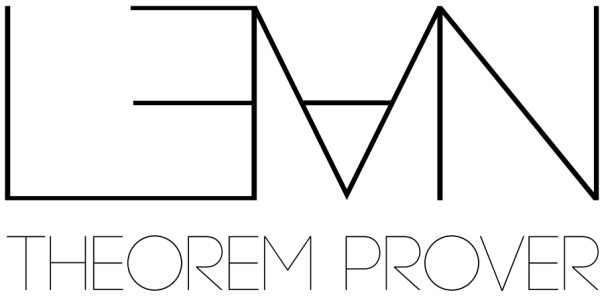
type depends on the value of n!

Dependent types

- Unified syntax of types and terms
- “Types are programs, too”
- Make your type system simpler* and more powerful!

* Terms and conditions apply

Dependent types



Demo: length-indexed lists in Rocq

Let’s do some type theory!

New features we want to support

Dependent function type

$$\Pi(x : A) \to B(x)$$

Type of functions f such that
if a : A, then f(a) : B(a)

"Pi type"

Dependent pair type

$$\Sigma(x : A) \times B(x)$$

Type of pairs (a,b) such that
a : A and b : B(a)

"Sigma type"

Example: zeros : Π(n : nat) → vect nat n

Pi types are functions

$$\frac{}{\Gamma} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B} \qquad \frac{}{B}$$

^^ ^^

the variable x can appear in B

Syntactic sugar: can just write A → B if x is not used in B

Pi types are “forall”

System F

$$\frac{\Delta, \alpha; \Gamma \vdash e : B \quad \alpha \notin FV(\Gamma)}{\Delta; \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\ B}$$

- New syntax and rules
- Extra contexts
- Extra side conditions

Dependent types

$$\frac{\Gamma, A : \mathsf{Type} \vdash e : B}{\Gamma \vdash \lambda A.e : \Pi(A : \mathsf{Type}) \to B}$$

+ Only one kind of lambda
+ Just an instance of the lambda rule

Syntactic sugar: can write ∀ x. B if the type A is clear from context

Challenge: equality of types

Is vect A e_1 the same type as vect A e_2?

e_1 / e_2 may have free variables — e.g., vect A (n+1), if n is in scope

Need to check equivalence of programs e_1 and e_2 — how?

Challenge: equality of types

Need to check equivalence of programs e_1 and e_2 — how?

Challenge: equality of types

Need to check equivalence of programs e_1 and e_2 — how?

- Evaluate to the same value on all inputs — undecidable
- Exactly the same AST — way too restrictive

Challenge: equality of types

Need to check equivalence of programs e_1 and e_2 — how?

- Evaluate to the same value on all inputs — undecidable
- Exactly the same AST — way too restrictive
- Equal up to α/β/η equalities — convenient and decidable

Restricting to pure, total functional programs makes this easier

Martin-Löf type theory

$$e, A, B ::= x \mid \lambda x.e \mid e_1 e_2 \\ \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e) \\ \mid \mathsf{Type} \mid \Pi(x : A) \to B \mid \Sigma(x : A) \times B$$

$$\Gamma ::= \cdot \mid \Gamma, x : A$$

Typing judgment

$$\Gamma \vdash e : A$$

Equality judgment

$$\Gamma \vdash e_1 \equiv e_2 : A$$

Basic typing rules

- Type is a type
- Variables from the context

$$\frac{}{\Gamma \vdash \mathbf{Type} : \mathbf{Type}}$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

Basic equality rules

- Replacing equal types
- ≡ is an equivalence relation

$$\frac{\Gamma \vdash A \equiv A' : \mathsf{Type} \qquad \Gamma \vdash e : A}{\Gamma \vdash e : A'}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash e \equiv e : A}$$

$$\frac{\Gamma \vdash e_1 \equiv e_2 : A \qquad \Gamma \vdash e_2 \equiv e_3 : A}{\Gamma \vdash e_1 \equiv e_3 : A}$$

$$\frac{\Gamma \vdash e_1 \equiv e_2 : A}{\Gamma \vdash e_2 \equiv e_1 : A}$$

Pi types

Formation

Introduction

Elimination

Computation

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \Pi(x : A) \to B : \mathsf{Type}}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : \Pi(x : A) \to B}$$

$$\frac{\Gamma \vdash e_1 : \Pi(x : A) \to B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 e_2 : B[e_2/x]}$$

$$\frac{\Gamma, x : A \vdash e_1 : B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash (\lambda x.e_1)e_2 \equiv e_1[e_2/x] : B[e_2/x]}\ (\beta)$$

Sigma types

Formation

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \Sigma(x : A) \times B : \mathsf{Type}}$$

Introduction

$$\frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B[e_1/x]}{\Gamma \vdash (e_1, e_2) : \Sigma(x : A) \times B}$$

Elimination

$$\frac{\Gamma \vdash e : \Sigma(x : A) \times B}{\Gamma \vdash \pi_1(e) : A} \qquad \frac{\Gamma \vdash e : \Sigma(x : A) \times B}{\Gamma \vdash \pi_2(e) : B[\pi_1(e)/x]}$$

Computation

$$\frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B[e_1/x]}{\Gamma \vdash \pi_2(e_1, e_2) \equiv e_2 : B[e_1/x]}$$

That's it!

- A few basic rules
- Rules for each feature:
  - Formation
  - Introduction
  - Elimination
  - Computation

Deciding equality and type checking

Proposition. For every equivalence class of well-typed terms, you can compute a canonical representative, called the *normal form*.

Proof sketch: by induction on the typing derivation, with a fairly complicated induction hypothesis.

→ Equality of terms (in particular, types) is decidable

→ Type checking is decidable

→ fails when you have Type : Type! (related to Russell's paradox)
Solution: use a hierarchy $\text{Type}_0 : \text{Type}_1 : \text{Type}_2 : \ldots$

Conclusion

Dependent types let you…

- Pass around types as values
- Put more interesting properties in types

… but checking equality of types becomes harder.

Friday: Propositions as types, Proofs as programs

Where to learn more about dependent types

To learn a dependently typed theorem prover:

- The Natural Number Game teaches Lean https://adam.math.hhu.de/#/g/leanprover-community/nng4
- Software foundations teaches Rocq https://softwarefoundations.cis.upenn.edu/
- PLFA teaches Agda https://plfa.github.io/

To learn type theory:

- Daniel Gratzer and Carlo Angiuli's book https://www.danielgratzer.com/papers/type-theory-book.pdf
- Chapter 2 of the HoTT book has a good intro https://homotopytypetheory.org/book/