A Guarded Syntactic Model of Gradual Dependent Types

Joey Eremondi

April 18, 2022

Overview

Two Problems - One Solution

Two Problems

- Implementing Gradual Dependent Types
- Denotational Semantics + Metatheory

One Soution

- Approximate Normalization + Translation to Static Dependent Types

Implementing Gradual Dependent Types

Don’t Reinvent the Wheel

Long-term goal: gradual types in a full-scale dependent language

Machinery

- Many parts to dependent type checking and compilation
  - Compile-time evaluation for comparisons
  - Unification/inference
  - Code generation/optimization
- Want to avoid re-implementing as much of this as possible

Efficiency

- Normalizing/comparing types is expensive
- Want to leverage existing techniques
  - E.g. Idris: experimental normalization by compilation to ChezScheme

Proposed Approach

Compile gradual dependent types to static dependent types *without changing the static core language*

- Can use existing normalization, unification, etc.

Challenge: Effects in gradual language

- Gradual languages: two effects
  - Failure (can just be modelled as special value in gradual language)
  - Non-termination
- Dependent languages restricted in how non-termination is used
  - Ensures consistency and decidablity of type checking

The Idris model of Non-termination

Functions marked as “partial”

- Are not checked for termination/productivity
- Allows for general recursion

At compile-time

- Partial functions never normalized
- Conservative: some equivalent partial-terms may rejected as non-equal
    - need to normalize to *see* that are equal
- Ensures type-checking terminates

Problem with the Idris model

Ad-hoc, hard to reason about

- e.g. it's hard to prove that every gradual program translates to a well-typed
- Want formalism of Idris-style non-termination, so can prove translation proves

We will use Guarded Type Theory as a theoretical model of Idris

- More on this later

Approximate Normalization