A Guraded Syntactic Model of Gradual Dependent Types

Translation to Support Implementation and Metatheory

ANONYMOUS AUTHOR(S)

TODO write abstract

1 INTRODUCTION

Gradual dependent give principled static and dynamic semantics to programs where part of a type or term is missing. By allowing imprecision, gradual dependent types allow for smooth migration of code from non-dependently typed languages, or even untyped languages, to full dependent types, allowing the programmer run and test their code even when the full type details haven't been figured out. This migration is easiest in languages fulfilling the *gradual guarantees* of Siek et al. [2015], which state replacing part of a program with ? creates no new static or dynamic type errors. The gradual guarantees ensure that, when an error is encountered, the problem is never too few types, but that two types in the program are fundamentally incompatible.

However, the benefits of gradual dependent types have not been realized, since existing developments have enabled the gradual guarantees at the expense of other desirable properties. Eremondi et al. [2019] presented a dependent calculus supporting the gradual guarantees, but relied on a termination argument that does not scale to inductive types. Lennon-Bertrand et al. [2022] presented two extensions of the Calculus of Inductive Constructions (CIC) that satisfy the gradual guarantees, but one has undecidable type checking and the other rejects some well-typed static CIC programs. They show that, to a degree, these sacrifices are unavoidable, to a degree: no dependently typed language can satisfy all of strong normalization, conservative extension of CIC, and the Embedding-Projection Pairs (EP-pairs) property, a strengthening of the gradual guarantees.

Another obstacle to the adoption of gradual dependent types is that gradual dependent types have not yet been meaningfully implemented. Constructing a compiler for a dependently typed language is a massive engineering effort, and involves writing a type checker, a convertability check for terms, and unification engine for inference, in addition to the code generation and optimization. Writing a compiler for a gradual dependently typed language involves all of this work, plus extra handling to ensure safety in the presence of type imprecision.

We address both these shortcomings in GrInd, a Gradual language with Inductive types. GrInd sacrifices strong normalization and EP-pairs, but keeps the properties that we actually want: decidable type checking, (weak) consistency and canonicity, the gradual guarantees, and conservatively extending CIC. Because type checking is decidable, GrInd can be translated into the core calculi of existing dependently typed compilers. Moreover, we show that GrInd does not violate static reasoning principles: propositionally-equal CIC terms embedded in GrInd are observationally-equivalent, and casts only change the error-behavior of terms, not the concrete results produced.

Our main contribution is a translation from GrInd to a static type theory:

- For implementation, the translation means that existing normalizers and code generators can be used "off-the-shelf" to compile GrInd programs;
- For metatheory, the translation serves as a syntactic model in the style of Boulier et al. [2017], which we use to prove the gradual guarantees and other metatheoretic properties;
- To enable decidable type checking, we adapt approximate normalization from Eremondi et al. [2019] to a cast calculus, using the syntactic model to prove termination;

- To model run-time semantics, we translate to *guarded type theory* [?], whose non-positive recursive types allow the non-termination of gradual types to be exactly represented in a consistent target language;
- Our translation and the theorems about it have been mechanized in Guarded Cubical Agda [?]

2 TWO PROBLEMS, ONE SOLUTION

2.1 Translation to Support Implementation

2.1.1 *Don't Reinvent The Wheel.*

2.1.2 *An Implementation Strategy.*

2.1.3 *The Idris Model of Non-Termination.*

2.1.4 *Translating Approximate Normalization.*

2.2 Metatheory

2.2.1 *Extinguishing the Fire Triangle.*

2.2.2 *Static Reasoning in Gradual Code.*

2.3 Modelling Gradual Dependent Types

2.3.1 *Modelling Approximate Normalization.*

2.3.2 *Guarded Type Theory.*

2.3.3 *Relating Approximate and Exact Normalization.*

REFERENCES

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The next 700 Syntactical Models of Type Theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs* (Paris, France) *(CPP 2017)*. Association for Computing Machinery, New York, NY, USA, 182–194. https://doi.org/10.1145/3018610.3018620

Joseph Eremondi, Éric Tanter, and Ronald Garcia. 2019. Approximate Normalization for Gradual Dependent Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 88 (July 2019), 30 pages. https://doi.org/10.1145/3341692

Meven Lennon-Bertrand, Kenji Maillard, Nicolas Tabareau, and Éric Tanter. 2022. Gradualizing the Calculus of Inductive Constructions. *ACM Trans. Program. Lang. Syst.* 44, 2, Article 7 (apr 2022), 82 pages. https://doi.org/10.1145/3495528

Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined Criteria for Gradual Typing. In *1st Summit on Advances in Programming Languages (SNAPL 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Thomas Ball, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morrisett (Eds.), Vol. 32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 274–293. https://doi.org/10.4230/LIPIcs.SNAPL.2015.274