# The Name of the Title is Hope

TBD and TBD, University of Colorado Boulder

TBD

**ACM Reference Format:**
TBD and TBD. 2020. The Name of the Title is Hope. CSCI 5535, Spring 2020 (April 2020), 2 pages.

## 1 INTRODUCTION

One of the more powerful tools in use by functional programmers are engines for searching over code bases given the type signature of the desired function. Such tools provide the entry-point for practical code reuse and management of large code bases. Tools such as Hoogle(for Haskell) provide a best-in-class search functionality to polymorphic functional languages, but the same tools are not as prevalent for dependently-typed languages.

There are a number of factors that have prevented robust search features from entering the dependently-typed landscape. The primary setback is the undecidability of type equality(isomorphism) in the presence of dependent types. For search over types to be performed there must be some way of saying whether two types are equal or not. This introduces many of the nuances of equality into the confusion of trying to build a practical tool.

We attempt to solve this problem by defining an abstraction which defines a search engine for dependent types, that can scale and adapt to appropriate definitions of type equality. It is reasonable for a user to want some control over how weak/strong the equalities for search are. Some may want strong definitional equalities, others may accept the more computationally burdensome(and semi-decidable) isomorphisms. We give an approach that leaves this control to the user, and also leaves open the possibility for very powerful forms of program search.

Another deficiency of current search engines for program source is the lack of being able to specify complex functional properties over the domain of search. Input-Output examples and algebraic properties offer two great sources of specification for search, yet they are not used, typically because of the lack of some method for verifying these properties. Fortunately, a dependently typed language offers a natural way of handling these properties.

Practically, we provide all of the abstractions formalized in Agda and presented as concrete tool for search over functions with input-output examples. The implementation is capable of being extended to support more expressive properties which in turn will provide more power to a user to specify complex queries.

## 2 OVERVIEW

Let's start with an example of a search query over a non-dependent type, given as a type:

$$\text{query} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

Solutions for this query would obviously include the usual arithmetic operations. But let's say that we wanted to further specify the query by providing some input-output examples:

Authors' address: TBD, tbd@colorado.edu; TBD, tbd@colorado.edu, University of Colorado Boulder.

| input 1 | input 2 | output |
|---|---|---|
| 0 | 0 | 0 |
| 2 | 2 | 4 |
| 1 | 2 | 3 |
| … | … | … |

We could devise the obvious procedure of checking the resulting functions against these examples and discarding those that are not consistent. In fact, this is exactly what we will do. But to jump to the algorithm would fail to realize the deeper type-theoretic possibilities here. See, the examples above can be encoded quite easily as a type for our query. The type below is consistent with the query:

$$\Sigma[\mathrm{f} \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}]((f\ 0\ 0\ \equiv 0) \times (f\ 2\ 2\ \equiv 4) \times (f\ 1\ 2\ \equiv 3))$$

This is a *dependent product* type that encodes our query as searching for a term that satisfies the type we gave along with proofs that the provided function produces certain results under evaluation.

## 3 (CONTRIBUTION 1)

## 4 (CONTRIBUTION 2)

## 5 EMPIRICAL EVALUATION

## 6 RELATED WORK

## 7 CONCLUSION

## ACKNOWLEDGMENTS

TBD

## REFERENCES