CSCI5535: Final Project and Literature Survey

Jack Martin & Michael Dresser

March 27, 2020

1 Literature Review

1.1 Type-and-Example-Directed Program Synthesis

Osera & Zdancewic

Why did you select this paper?
It is the first attempt at a types-first approach to program synthesis in a style that would be familiar to anyone that has programmed in a polymorphic functional language. It addresses the "why do these programs seem to write themselves" question.
What is the "main idea" of this paper?
Using types and examples as a specification, Functional Programs can be synthesized for some small problems very quickly.
How well is this main idea communicated to you when you read the first two sections and conclusion of paper, and skimmed the rest?
The main idea is communicated concisely for the initiated reader. Section two lays out each aspect of the problem and solution very succinctly at a high level before diving into the details. One weakness of the result is that it fails to use any library code in the solutions. This ignores one of the strengths of function programming, the use of generic code. One strength is the pushing of examples into the leaves of the typing derivation trees as the program is synthesized.

1.2 Example-Directed Synthesis: A Type Theoretic Interpretation

Frankle et al.

Why did you select this paper?
One of the avenues for our project includes using type information and related properties during synthesis. This paper converts example-based synthesis to refinement type-based synthesis and could inform our work.
What is the "main idea" of this paper?
Example-based synthesis can be portrayed, instead, as refinement-type based synthesis. With that information, a synthesis engine can be developed that takes advantage of the large corpus of existing work in type theory/intuitionistic

logic.
How well is this main idea communicated to you when you read the first two sections and conclusion of paper, and skimmed the rest?
I had some difficulty teasing out the really valuable contributions of this work, likely because of my lack of familiarity with what a ”sequent calculus” is and with type theory in general. In my view, the most interesting results of this paper are a more formal take on MYTH [Osera and Zdancewic 2015] and the earlier-mentioned parallel drawn between example-directed synthesis, refinement type systems, and intuitionistic logic.

1.3 Constraint-Based Type-Directed Program Synthesis

Why did you select this paper?
A likely avenue for our project involves synthesis using types/type systems. This paper expands existing type-directed synthesis work.
What is the “main idea” of this paper?
Use constraint-based inference techniques (like those in many type inference systems) but flip them to produce potential programs based on types, an extension of similar work that uses only the type system of the language to synthesize programs.
How well is this main idea communicated to you when you read the first two sections and conclusion of paper, and skimmed the rest?
It takes a bit of setup to get to the main idea in the introduction, but I found the setup to be very helpful to end up with a clear understanding of the problem that is trying to be addressed. I believe the important contributions of the paper are the usage of constraints in type-directed synthesis and the usage of that system to approach synthesis of programs with complicated polymorphic types. The author takes slightly too long to explain the usage of constraint-based typing in type inference - it would go much better in an earlier section while the concept of flipping typing judgement for synthesis is fresh in the reader’s mind.

1.4 Augmented Example-based Synthesis using Relational Perturbation Properties

Why did you select this paper?
It presents an approach that we may wish to build off of, namely augmenting example-based synthesis using specified or inferred properties. Our work could take this a step further and maintain/use the properties beyond simply using them to generate more examples.
What is the “main idea” of this paper?
Using ”relational perturbation properties,” which are essentially relations between changes of inputs and outputs, to produce a larger set of examples for example-based synthesis.
How well is this main idea communicated to you when you read the

first two sections and conclusion of paper, and skimmed the rest?
The main idea is communicated extremely clearly early on. It also nicely lays out which sections go into more depth on specifics of the work.

Important: "relational perturbation properties" can be inferred just from example sets and can be used to significantly improve the performance of an existing synthesis program. This can also be done with user-specified properties.

Clear: The reason to pursue the work, the high-level explanation of the system that was designed, the components (levels of UI, perturbation inference, end-result), and analysis of results were all presented clearly and concisely early on and in the conclusion of the paper.

1.5 Dimensions of Program Synthesis

Sumit Gulwani

Why did you select this paper?
It is a survey of work in the field of program synthesis. It should provide a good high-level overview of how researchers approach problems in synthesis as well as potential references to inform our work.
What is the "main idea" of this paper?
To describe the synthesis problem at a high level and provide examples of various research projects that currently exist in the field.
How well is this main idea communicated to you when you read the first two sections and conclusion of paper, and skimmed the rest?
The main idea is communicated very well. The three dimensions of program synthesis (type of constraints, search space, and search technique) are introduced early and explained concisely. The majority of the rest of the paper is dedicated to exploring each dimension in further depth. Important aspects: dimensions of the synthesis problem - clear. Weaknesses: While most of the paper is clear, occasional stumbles of grammar do break flow and cause brief confusion.

1.6 The Hoogle Report

Neil Mitchell

This paper was selected because of Hoogle's status as the first-in-class search tool for functional languages. The main idea in this report is of a technical and historical nature, but did lead to some papers for how to approach equality over types, which will be one of the main problems to tackle in our project.

1.7 Isomorphisms of Types

Roberto de Cosmo

This is a textbook/survey of methods for showing isomorphisms between types. It is considered the "right" formalism of equality over types for the purpose of searching. This is considered in the text as well as in the Hoogle

Report. The main idea is that some types will be the same functionally as others. The most notable example is between curried and un-curried functions, but many other cases exist. We will need to consider how much we want to deal with isomorphisms in our search procedure, as the problem is undecidable in the general case for dependent types.

1.8 Using Types as Search Keys in Functional Libraries

Mikael Rittri

This is, historically, the first paper to use type isomorphisms for the purpose of searching for functions. It was read mainly for historical and inspirational reasons and to get a general sense for how Rittri thought about the problem. The paper itself wont lend itself very useful though since its results are better presented in “Isomorphisms of Types”.

2 Project Proposal

2.1 Problem Description

We will write a tool for searching Agda libraries for functions which satisfy a search specification including:

1. Type
2. Input-Output Examples
3. (Stretch Goal) Proven Relational Properties

This will be the first tool to search an Agda code base using examples and properties. Some examples for this program are:

```
Query                                | Results
-------------------------------------|-------------------------
Type : Nat -> Nat -> Nat             | _+_ : Match
Examples : 2 , 2 => 4                | _x_ : Match
Properties : Associative, Identity   |       ...
                                     | _divmod_ : n% Consistent
```

We will introduce some notion of consistency to allow for partial matches to a query. The amount that a query matches can be used to inform the user how "off" it is by some ratio of matched properties to those that couldn't be found.

A key theoretical observation here is the relation of program search to program synthesis and a novel type-theoretic interpretation of program search(more on this in the paper).

2.2 Approach

Our approach to solve this problem will rely on past research in program search(Hoogle etc.) for how to best structure the program data. Our main contribution will be to formalize a notion of consistency for functions being searched over. We will need to work out how to best organize the data such that it is easy to find which programs satisfy queries.

We will use example-based search as a concrete case of the more general property-based search that we are confident could become a near-future extension to the project.

2.3 Feasibility

This seems to be a problem that is fairly easy to solve using known methods along with common-sense extensions for supporting dependent-types. The main theoretical work will be in making such a tool efficient for the large domains it will eventually need to handle. We will use the technical report on Hoogle for inspiration and see if we can adapt their methods to the domain of agda programs.

2.4 Demonstration and Evaluation

If the project is successful we would like to be able to support search over the Agda Standard Library for common properties. The demonstration should be able to handle queries such as the one given above, but also for multiple i/o examples and other properties than just algebraic ones. We will develop a test-bed of example queries before developing the tool so we have an objective standard to hold ourselves up to.

3 Section Drafts

3.1 Related Work

Program Search Program Search, or the search of functions in some library, has led to such tools as Hoogle, and is the main category that this work could be included under. In the past, none of these tools use any functional properties or examples to help guide the search. Ours is the first tool to provide this functionality.

draft note: make sure to look into the search programs for other languages.

Program Synthesis As was mentioned in Section # above, program search can be viewed as a special case of the more general program synthesis problem. It seems hopeful that program search can be later used as a tool in program synthesis by treating it as an atomic operation for larger synthesis procedures.

Type Isomorphism Isomorphisms over types acts as the equivalence relation that we use for search in our approach. In reality our approach defines a slightly more robust isomorphism relation over types *and* examples. This is slightly more powerful than just the types themselves. In future work we will look at the theoretical basis for this.

3.2 Abstract

The problem of Program Search - looking for a function which satisfies a query - is one of the more practical tools a developer can use when programming. Search tools such as Hoogle provide an entrypoint for a language eco-system, but unfortunately often don't accept query information other than types. Here we present a search tool for searching over types and examples as well as a general framework for doing program search over dependent types. We also provide a type-theoretic interpretation of the program search problem as a special case of the more general synthesis problem and discuss how this presents a tool for furthering the solution to both problems.