Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Inductive Synthesis from Higher-Order Functions

Master Thesis

Alexandra Maximova

August 15, 2016

Advisors: Prof. Dr. Martin Vechev, Dimitar Dimitrov

Department of Computer Science, ETH Zürich

Abstract

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Contents

Chapter 1

Introduction

1.1 About program synthesis in general

put in some context, link 1

1.2 Problem definition

have many components, put them together into a program, no lambdas, no if-then-else, no recursion

1.3 Contributions

Evaluation, exploring the baseline algorithm, exploring the search space

Chapter 2

Related Work

Try to answer the following three question for each paper read:

1. What is new in this approach? Or better, what is the approach. Describe technically the approach, so that you can answer technical questions.
2. What is the trick? (Why are they better than others?)
3. Which examples they can do really well? What kind of examples do they target? What is the most complicated thing they can generate?

Nadia Polikarpova 2015
here is a talk: `http://research.microsoft.com/apps/video/default.aspx?id=255528&l=i`
and here is the code: `https://bitbucket.org/nadiapolikarpova/synquid`
In [4] SYNQUID is proposed. Refinement types (types decorated with logical predicates) are used to prune the search space. SMT-solvers are used to satisfy the logical predicates. The key is the new procedure for type inference (called modular refinement type reconstruction), which thank to its modularity scales better than other existing inference procedures for refinement types. Programs can therefore be type checked even before they are put together. Examples that this tool is able to synthesize include several sorting algorithms, binary-search tree manipulations, red-black tree rotation as well as other benchmarks also used by other tools (**TODO:** read about these benchmarks and write if there is something interesting). The user specifies the desired program by providing a goal refinement type.

Feser 2015
The tool proposed in [2] is called λ^2 and generates its output in λ-calculus with algebraic types and recursion. The user specifies the desired program providing input-output examples. No particular knowledge is required from the user, as was demonstrated using random input-output examples

The examples are inductively generalized in a type-aware manner to a set of hypotheses (programs that possibly have free variables). The key idea are the hard-coded deduction rules used to prune the search space depending on the semantics of some of the higher-order combinators (map, fold, filter and a few others). Deduction is also used to infer new input-output examples in order to generate the programs needed to fill in the holes in the hypotheses. This tool is able to synthesize programs manipulating recursive data structures like lists, trees and nested data structures such as lists of lists and trees of lists. The examples that require much more time to be synthesized than the others are dedup (remove duplicate elements from a list), droplast (drop the last element in a list), tconcat (insert a tree under each leaf of another tree), cprod (return the Cartesian product of a list of lists), dropmins (drop the smallest number in a list of lists), but all of them are synthesized under 7 minutes.

Kincaid 2013

In [1] Escher is presented, an inductive synthesis algorithm that learns a recursive procedure from input-output examples provided by the user. The user must provide a ”closed” set of examples, otherwise recursion cannot be handled properly The target language is untyped, first-order and purely functional. The algorithm is parametrized by components that can be instantiated differently to suit different domains. The approach combines enumerative search and conditional inference. The key idea is to use a special data structure, a *goal graph*, to infer conditional branches instead of treating `if-then-else` as a component. Observational equivalence is also used to prune the search space. Programs with the same value vectors (output of the program when applied to the inputs of the input-output examples) are considered equivalent and only one of them is synthesized. An implementation of the tool was tested on a benchmark consisting of recursive programs (including tail-recursive, divide-and-conquer and mutually recursive programs) drawn from functional programming assignments and standard list and tree manipulation programs. For all examples the same fixed set of components was used. The tool is able to synthesize all of them quickly. There is very little information on how many input-output examples were needed to synthesize the benchmarks and how difficult it is for a non-experienced user to come up with a ”closed” set of examples.

Osera 2015

The tool in [3] is called Myth and uses not only type information but also input-output examples to restrict the search space. The special data structure used to hold this information is the *refinement tree*. This system can synthesize higher-order functions, programs that use higher order functions and work with large algebraic data types.
There is an ML-like type system that incorporates input-output examples. Two pieces: a *refinement tree* and an enumerative search.

Two major operations: refine the goal type and the examples and guess a term of the right type that matches the examples.
A small example to show what does the procedure. The user specifies a goal type incorporating input-output examples as well as the ”background”: the types and functions that can be used.

```
stutter :
```

Chapter 3

Definitions

3.1 System F

Terms and Types

Evaluator

Maybe shift to the appendix the implementation of lists, booleans, integers and trees in System F?

3.2 Top down type driven synthesis, aka stupid queue

3.2.1 Successor rules

Write about stack vs queue of open holes

3.2.2 Type unification

3.3 Enhancements

3.3.1 Priority queue and cost functions

nof-nodes

nof-nodes-simple-type

no-same-component

length of the string

3.3.2 Black list

automatic generation of black list discussed in evaluation

3.3.3 Templates

Top-down type-driven synthesis, aka stupid queue

Successor rules

Do we really need to describe it? Well, I have code to generate templates, but it does not really do what it is meant to do

Chapter 4

Implementation

What could I talk about in this chapter?

- Programming language and compiler version
- Picture of the code??? (Diagram of modules or something like that)
- Library format??? (the not-really vim plugin) and the type-checking when added to the library?
- What are Fun, FUN and BuiltinFun?

Chapter 5

Evaluation

What I want to see in this chapter:

- Table of all components
- Table of synthesized programs with synthesis times for the different algorithms
- Figure of the synthesis times of synthesized programs
- Comparison to Feser
- Explain why automatic black list and templates perform so poorly
- Show advantages of using black lists (I know it's trivial)
- Talk about the trivial thing that if you have more functions with the same time you will need much more time to find the program you are looking for.
- Talk about the constants in the cost functions and how they affect the search space???
- Stack versus queue expanding $\rightarrow$ mention it in the definitions

5.1 Set up

Describe the machine and the testing set up, which components were used, what information did you give to the synthesizer, how many examples were given. What else?

5.2 Cost functions

Compare the different cost functions with each other, explain why are they good for some programs and bad for other. Table.

5.3 Black lists

5.3.1 Benefits of black lists

Table of you super manual black list. Figure that for each cost function compares time with and without black list. Trivial words about pruning search space and useless branches.

5.3.2 Shortcomes of automatically generated black lists

Maybe really bring the automatically generated table. Point at the repetitions. Say that automatically generated I/O-examples are bad and we need a lot more of them. Say that you tried only identity pruning, but one could also try to generate, say, the empty list or whatever.

5.4 Templates

Short section explaining that the templates you generate are not the templates you expect and why

Chapter 6

Conclusions

6.1 Conclusions

The baseline is not that bad. Gathered some data about the search space.

6.2 Future Work

Templates done well, augmented examples?

Appendix A

Dummy Appendix

You can defer lengthy calculations that would otherwise only interrupt the flow of your thesis to an appendix.

Bibliography

[1] Aws Albarghouthi, Sumit Gulwani, and Zachary Kincaid. Recursive program synthesis. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 934–950, Berlin, Heidelberg, 2013. Springer-Verlag.

[2] John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2015, pages 229–239, New York, NY, USA, 2015. ACM.

[3] Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2015, pages 619–630, New York, NY, USA, 2015. ACM.

[4] Nadia Polikarpova and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. *CoRR*, abs/1510.08419, 2015.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.