Generating Examples for Comprehension of Synthesized Programs

ZITENG WANG, UC San Diego, USA

1 INTRODUCTION

Consider the task of implementing a function `mapEither f xs` with the following type signature:

$$\texttt{mapEither :: (a → Either b c) → [a] → ([b], [c]).} \tag{1}$$

This function applies `f` to each element of list `xs` and partitions the results into two lists, based on whether `f` returned `Left` or `Right`. Here, we want to implement Eq. 1 by combining library functions to prevent code duplication, but how to discover such an implementation?

Hoogle+. To address this issue, Guo et al. have recently developed Hoogle+ [Guo et al. 2020; James et al. 2020], a component-based synthesis engine for Haskell. In this case, Hoogle+ produces the desired result

$$\texttt{\\f xs → partitionEithers (map f xs).} \tag{2}$$

However, the type signature alone cannot precisely capture programmer intent; synthesized results can be undesired even under the same type. To be specific, some results are never useful as they always throw an exception; another class of unhelpful results are duplicates, where several syntactically distinct results have the same behavior.

2 PRIOR WORK: CHECK+ FOR RESULT FILTERING

Check+. Our previous work on improving the quality of results is Check+ [James et al. 2020], an extension to Hoogle+ to filter out classes of unhelpful results identified above. To be specific, Check+ tests each synthesized result by property-based testing. Firstly, to eliminate results that always crash, it finds combinations of arguments such that the synthesized result runs correctly, and thus classifies result to be always crashing or not. Furthermore, it tries to derive a distinguishing input for synthesized results, and eliminate duplicates based on the existence of such a distinguishing input. For example, Check+ generates the following examples for Eq. 2.

$$\texttt{(\\x → if x > 0 then Left x else Right x) [4,-1]} \Rightarrow \texttt{([4], [-1])} \tag{3}$$
$$\texttt{Left [0]} \Rightarrow \texttt{([0], [])} \tag{4}$$
$$\texttt{Left [4]} \Rightarrow \texttt{([4], [])} \tag{5}$$
$$\texttt{Right [7]} \Rightarrow \texttt{([], [7])} \tag{6}$$
$$\texttt{Right [8]} \Rightarrow \texttt{([], [8])} \tag{7}$$

3 PROPOSED WORK: CHECK+ FOR RESULT COMPREHENSION

Humans are good at learning by example: for example, the user could easily guess what Eq. 1 does by looking into the input-output pair Eq. 3. However, these examples are not necessarily self-explanatory enough for comprehension: there are many similar or duplicate kinds of examples that would not provide any extra information. For instance, all of Eq. 4, Eq. 5, Eq. 6, and Eq. 7 can be classified into a class where the result is always classified as `Left` or `Right`.

Moreover, another challenge on generating good examples arises when we are trying to derive function instances for both testing higher-order functions and comprehension. It is tricky to formalize

Author's address: Ziteng Wang, UC San Diego, USA, ziw329@ucsd.edu.

a strategy to derive an instance of various functions, given the variety in the real-world applications of higher-order functions. We want to have a good balance between the readiness, effectiveness of examples, and the performance of the example-generation algorithm.

Proposal. Our goal is to generate examples that are helpful for comprehension. The ideal usage of Hoogle+ is that each solution could be easily construed with the examples given so that the user would locate the desirable solution quickly.

(1) Analyze human-designed vs. auto-generated examples and extract patterns of what humans do;
(2) Implement more human-like generation and address the difficulty on HOF mentioned above;
(3) Run user study to evaluate comprehension with auto-generated examples vs. human-designed examples.

REFERENCES

Zheng Guo, Michael James, David Justo, Jiaxiao Zhou, Ziteng Wang, Ranjit Jhala, and Nadia Polikarpova. 2020. Program Synthesis by Type-Guided Abstraction Refinement. *PACMPL* 4, POPL (Jan. 2020), 28. https://doi.org/10.1145/3371080

Michael James, Zheng Guo, Ziteng Wang, Shivani Doshi, Hila Peleg, Ranjit Jhala, and Nadia Polikarpova. 2020. Digging for Fold: Synthesis-Aided API Discovery for Haskell. *PACMPL* 4, OOPSLA (Nov. 2020), 28. https://doi.org/10.1145/3428273