# MagicHaskeller on the Web: Automated Programming as a Service

[System Demonstration Proposal]

Susumu Katayama
University of Miyazaki
skata@cs.miyazaki-u.ac.jp

## Abstract

The proposed demonstration will present our Web-based automatic programming tool, named MAGICHASKELLER ON THE WEB, which can help casual programming in Haskell. We will show how simple to use the tool is, and then evaluates its ability.

***Categories and Subject Descriptors*** I.2.2 [*ARTIFICIAL INTELLIGENCE*]: Automatic Programming; D.1.1 [*PROGRAMMING TECHNIQUES*]: Applicative (Functional) Programming; H.3.5 [*INFORMATION STORAGE AND RETRIEVAL*]: Online Information Services

***General Terms*** Algorithms

***Keywords*** Automatic programming, Inductive functional programming, Web application

## 1. Introduction

It is a labor to learn a computer language. The learner has to understand its paradigm, learn its syntax and semantics, and memorize the essential part of its standard library.

It is easy to imagine that for usual people the memorizing part can be a pain. Although the best way to learn a computer language is writing programs, without enough amount of knowledge and experience it is difficult to write programs without stress.

In order to cope with the lack in knowledge and experience, there are various tools and devices. Hoogle[4] is an on-line dictionary for standard Haskell libraries, and it can generate links to documentations of library functions in reply to queries which can be ambiguous to some extent. Some integrated development environments (IDEs) and high-level editors can show functions in available libraries (e.g. the library browser of Leksah[1] and the block editor of App Inventor[2]), and some environments have auto-completion support (e.g. the auto-completion provided by Glasgow Haskell Compiler interactive (GHCi) and the proof search functionality provided by Agda-mode). Such needs are not limited to general-purpose languages. Flash Fill[1] provides a functionality of automatic synthesis of string processing functions based on regular expressions, and is one of the killer functionalities of Microsoft Excel 2013.

The proposed demonstration will present MAGICHASKELLER ON THE WEB, our tool for automated inductive Haskell programming. This tool is as simple to use as usual web search engines. Programs can be synthesized by just entering a one-line specification in a text box , and then, results are printed immediately in most cases. Viewing the return value of applying a generated function to some random arguments is just one click away. If some value is different from the user's expectation, she can make search again by correcting the value (which is in a text box) and pressing the enter key. She can also view the return value of applying the generated function to some argument in mind.

[1] `http://www.leksah.org/`

[2] `http://appinventor.mit.edu/`

## 2. System usage

Programs are searched for by filling in the text box in the Web page[3] with an *incomplete specification*, or the condition that the function must satisfy, and clicking on the "`Synthesize f`" button. The incomplete specification must be a Boolean-valued expression using `f` as a free variable, such as `f "abcde" 2 == "aabbccddee"` which appears in a text box as an example. Then, implementations of `f` which make the expression true are printed, if possible by this tool.

In the case of the example, a resulting expression

```
(\a b -> concat (transpose (replicate b a)))
```

should instantly be synthesized by clicking on the "`Synthesize f`" button. The used library functions, i.e., `concat`, `transpose`, and `replicate` have links to their documentation.

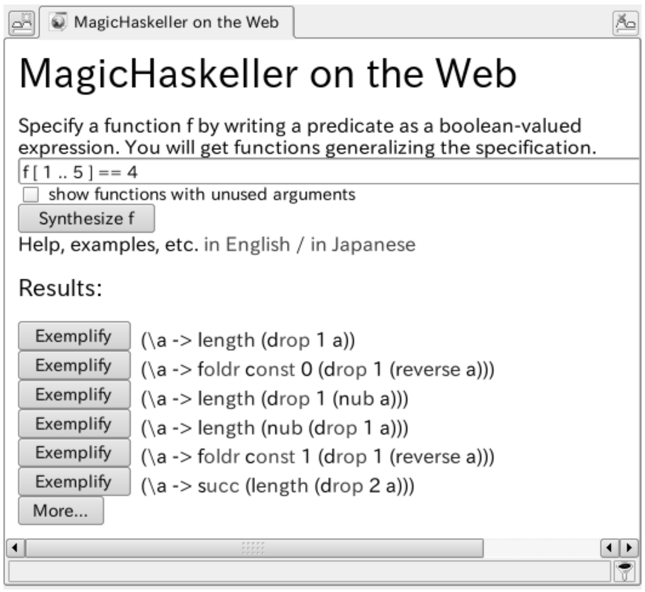
Depending on the specification, more expressions may or may not be obtained by clicking on the "`More`" button appearing at the bottom, though synthesis stops at some point in order to preserve stability of the server. All the resulting expressions are proved to be semantically different from one another. Functions with unused arguments are not shown by default, because it is unlikely that the intended function has an unused argument.

***Narrowing the search condition*** The second example is synthesis of the function returning the second last element of the list-typed argument. Fig. 1 shows the result of using `f [1..5] == 4` as the incomplete specification and clicking the "`More`" button once.

This example suggests that it is not always the case that the intended function is obviously one of the first synthesized functions. Even in this case, the user can casually obtain the intended function, in a similar way to using a search engine.

First thing the user should do is to click one of the "`Exemplify`" buttons to see the page showing the results of applying the function which was shown to the right of the button to some random arguments (Fig. 2). The text box filled with the function can be used to evaluate Haskell expressions. Thus, if the user wants to see the results of applying the function to other arguments in mind, she can add the arguments after the function in the text box and click

[3] `http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html`



MagicHaskeller on the Web

Specify a function f by writing a predicate as a boolean-valued expression. You will get functions generalizing the specification.

```
f [ 1 .. 5 ] == 4
```

show functions with unused arguments

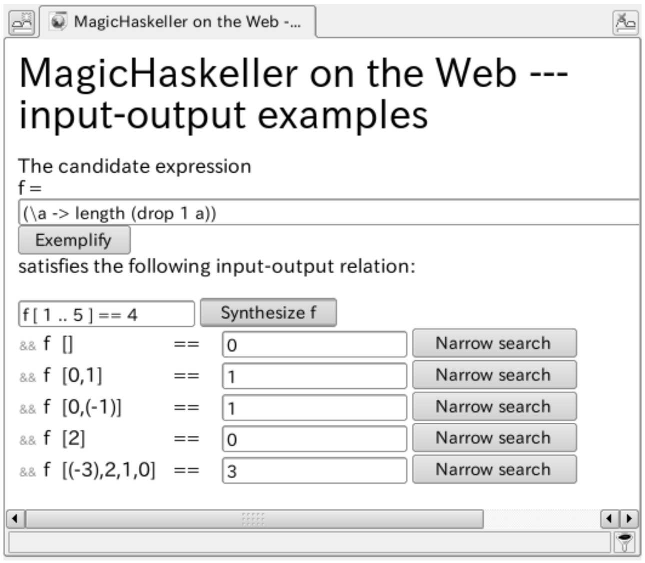
Synthesize f

Help, examples, etc. in English / in Japanese

Results:

```
Exemplify  (\a -> length (drop 1 a))
Exemplify  (\a -> foldr const 0 (drop 1 (reverse a)))
Exemplify  (\a -> length (drop 1 (nub a)))
Exemplify  (\a -> length (nub (drop 1 a)))
Exemplify  (\a -> foldr const 1 (drop 1 (reverse a)))
Exemplify  (\a -> succ (length (drop 2 a)))
More...
```

**Figure 1.** Case where narrowing is required



MagicHaskeller on the Web --- input-output examples

The candidate expression
f =

```
(\a -> length (drop 1 a))
```

Exemplify

satisfies the following input-output relation:

```
f [ 1 .. 5 ] == 4        Synthesize f
&& f  []            ==  0   Narrow search
&& f  [0,1]         ==  1   Narrow search
&& f  [0,(-1)]      ==  1   Narrow search
&& f  [2]           ==  0   Narrow search
&& f  [(-3),2,1,0]  ==  3   Narrow search
```

**Figure 2.** Applying a synthesized expression to random arguments

the "`Exemplify`" button and obtain the result. It is also possible to apply another higher-order function to the function.

If the user finds that the result of applying it to an argument is different from the intended value, she can narrow the search by adding the relation between the argument and the result as a new condition. In the case of Fig. 2, the second line goes `f [0,1] == 1` while it should be `f [0,1] == 0`. The user can easily add the condition by correcting the return value `1` in the text box to `0` and click the "`Narrow search`" button.

This time, functions returning the second last element are obtained. There are two functions, but by clicking their "`Exemplify`" buttons the user can easily notice that they are equivalent except for the special cases where the argument is a list of length at most one.

## 3. Implementation

The MAGICHASKELLER ON THE WEB system consists of a CGI program running on an Apache server and a backend server program, and they communicate via a network socket.

The backend server does the actual program synthesis. The synthesis algorithm takes a generate-and-test approach: it generates an infinite stream of all the expressions having the same type as `f`; then, they are tested against the given predicate. The implementation is based on [2] which discusses how to efficiently generate a stream of expressions with the given type using memoization, and [3] which discusses how to remove semantically equivalent expressions. There are also new devices to reduce heap usage further.

This approach requires a big memoization table. The point of running the algorithm continuously as a service is that users can share a monolithic memoization table (of tens of gigabytes) among accesses. The downside is that they do not have the option to choose which functions are included in the search.

All other tasks are done by the CGI front end. Generating printable functions as arguments of higher-order functions is achieved by running the same algorithm as the backend server, using a small memoization table.

## 4. Evaluation

When applied to the 16 problems not requiring data type definitions from the first 20 of 99 Haskell Problems[4], MAGICHASKELLER ON THE WEB could solve 11 of them, partially solve 1 of them, and could not solve 4 problems. Also, it could solve 3 of the 9 examples used in Section 3 of the Flash Fill paper[1].

Those who think the results are not remarkable should pay attention to the fact that they are the results of just filling the text box shown in MAGICHASKELLER ON THE WEB with the queries, without any domain-specific tuning. Although only easy problems could be solved, all the Flash Fill problems are still problems requested by users, most of which are taken from the Excel Help Forum. The fact that they could be solved without any change in the settings when solving 99 Haskell Problems suggests that this tool can be useful for solving unknown problems.

Even when solving domain specific problems, this tool is still worth trying, because it is easy to try and the results are usually obtained instantly. Three from the solved 99 Haskell Problems were so easy that only pointing single library functions solved them, but even in such cases this tool should be as useful as a reverse dictionary for a beginner Haskell programmer.

## 5. Conclusions

A Web-based automatic programming system named MAGICHASKELLER ON THE WEB is presented. This tool is designed to be helpful especially for beginner Haskell programmers to casually implement pure total functions.

## References

[1] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, pages 317–330, 2011.

[2] S. Katayama. Systematic search for lambda expressions. In *Trends in Functional Programming*, volume 6, pages 111–126. Intellect, 2007.

[3] S. Katayama. Efficient exhaustive generation of functional programs using monte-carlo search with iterative deepening. In T. B. Ho and Z.-H. Zhou, editors, *PRICAI*, volume 5351 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2008. ISBN 978-3-540-89196-3.

[4] N. Mitchell. Hoogle overview. *The Monad.Reader*, 12, 2008.

[4] `http://www.haskell.org/haskellwiki/H-99:_Ninety-Nine_Haskell_Problems`