Fakultät Wirtschaftsinformatik und Angewandte Informatik

Otto-Friedrich-Universität Bamberg

Induction on Number Series - A case study with MagicHaskeller on the Web

Henrik Marquardt (Matr. No. 1666105)

Project Report

WS 2014/2015

April 14, 2015

Supervisor: Prof. Dr. Ute Schmid

Abstract

Induction of number series is part of most intelligence tests nowadays. Also cognitive science research has a focus to that subject. With induction of number series you can measure inductive reasoning ability. Systems like *IGOR II* and *MagicHaskeller* deal with induction of number series. This paper has a focus on *MagicHaskeller*, especially *MagicHaskeller on the Web*. The aim of this paper is to give a short introduction to *MagicHaskeller on the Web* - a description of the system, the way how to use the system and the capabilities of the system. After the description of the system the used data and ideas were presented. Then the way how to use the system depending on three possibilities was detected. This way was taken for evaluation to describe the capabilities of the system. And at least the conclusion gives an overview of the results and some problems that could not be fixed yet.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As a part of many classical intelligence tests in psychology, solving number series pose an interesting and also challenging task for both humans and machines. One function of an intelligent test is to measure inductive reasoning ability. Solving number series depends on difficulty of operators, amount of numbers, complexity and other criteria. Cognitive science research has a focus on the ability of inductive reasoning (Hofmann et al., 2014). An approach of artificial intelligence is to solve number series by program synthesis. Inductive functional programming algorithms do so. The algorithms automatically generate functional programs from a set of input-output example pairs (Katayama, 2012).
Currently there are two approaches to inductive functional programming under active development - the *analytical approach* and the *generate-and-test approach*(Katayama, 2012). Both approaches can be used for generating functional programs, e.g. IGOR II uses the *analytical approach*, MAGICHASKELLER uses the *generate-and-test approach*.
This article deals with MAGICHASKELLER especially the MAGICHASKELLER ON THE WEB. On the one hand it is about how to use the system: which functionalities are provided by the system and which form of input-output examples is required to work with the system. On the other hand this article tests which problems in form of number series MAGICHASKELLER ON THE WEB is able to solve and which ones not.
In 2 there is a description of the used number series and an introduction to the used system MAGICHASKELLER ON THE WEB. 3 describes the form of input-output possibilities and how to deal with the number series in conjunction with the system. In 4 there is an evaluation of Chapter 3 - which possibilities work fine and which number series MAGICHASKELLER ON THE WEB is able to solve. And in 5 there is a short conclusion and the things that could not be done so far.

Chapter 2

Theoretical Background

2.1 Number Series

2.1.1 Number Series and Intelligence

Number series problems tasks are part of intelligence tests. Intelligence tests were made for measuring the IQ-Value - the intelligence of a subject. For solving these number series problems you need logical capabilities or calculating skills. These skills are important factors for intelligence measurement (Ragni and Klein, 2011).

2.1.2 A collection of Number Series

To illustrate the relation between numbers you can define number series mathematically as a function $f_n|n \in N$ with the domain N and the co-domain W (Erbrecht et al., 2003). *MagicHaskeller* and *MagicHaskeller on the Web* are able to do this if the systems have a number series as an input. How to give *MagicHaskeller on the Web* a correct input will be described later.
There are two ways of illustrating the relation of numbers inside of number series - explicitly and recursively.

	Number series	Function
explicitly	1, 4, 9, 16, 25	$a_n = n^2$
recursively	3, 5, 7, 9, |11	$a_n = a_{n-1} + 2$

Figure 2.1: Explicitly and Recursively Illustrated Functions from (Martina, 2012) [p.5]

Explicit means that the successor depends on the position inside of a number series.
Implicit means that the successor depends on the pre-successor. (Martina, 2012)

Finding a representation of a number series depends on many factors. Some number series e.g. $3, 5, 7, 9, 11$ represented as $f_n = f_{n-1} + 2$ seems to be very

easy while e.g. $7, 7, 14, 42, 168$ represented as $f_n = f_{n-1} * n$ seems to be hard to find quickly a representation. 2012 a group of students - Grünwald et al. (2012) - had to think about number series that can be used to test *MagicHaskeller* on its functional complexity. They classified number series into three classes: *Operational Complexity*, *Numerical Complexity* and *Structural Complexity*.

Operational Complexity

Operational Complexity means the complexity of operators e.g. addition, subtraction, multiplication or division. Also possible operators could be square-root and others. It is obvious that the complexity respectively the difficulty of usage of operators are different. So for humans it is easier to operate with an addition than to operate with a division.

Numerical Complexity

Another factor is the *Numerical Complexity*. Opined is the value of numbers. For humans it is easier to operate with low values than with high values, e.g. working with values inside the times table is easier than working with values of the The Great Multiplication. The reason for that is that (the most) people learned the times table in the course of their life and another reason is the more often usage of the times table than The Great Multiplication. There is no telling up to which value the *Numerical Complexity* increases strongly or what are the domains of low values or high values, but it is fact that the *Numerical Complexity* raises by increasing values.

Structural Complexity

The third dimension of Grünwald et al. (2012) classification is the *Structural Complexity*. *Structural Complexity* means the combination of operators and values, e.g. pre-successor, fix operator and fix values. They divided the *Structural Complexity* into five classes.

- Class 1: pre-successor - fix operator - fix values
- Class 2: pre-successor - fix operator - series from class 1
- Class 3: pre-successor - varying operator - varying values
- Class 4: pre-successor - pre-successor - fix operator
- Class 5: pre-successor - fix operator - series from class 4

Number Series

To get number series including *Operational Complexity*, *Numerical Complexity* and *Structural Complexity*, Grünwald et al. (2012) linked all three classes. They combined *Operational Complexity* and *Numerical Complexity* (simple - simple; complex - simple; simple - complex; complex - complex) and linked the five classes to each combination. The result are 20 number series with different specifications of combinations between *Operational Complexity*, *Numerical Complexity* and *Structural Complexity* 2.1.2 that will be used as input to demonstrate the functional complexity of *MagicHaskeller on the Web*.

numerical simple - operational simple

- Class 1 - NS1: 3, 5, 7, 9, 11, (13)
- Class 2 - NS2: 2, 7, 13, 20, 28, (37)
- Class 3 - NS3: 5, 7, 10, 12, 15, (17)
- Class 4 - NS4: 2, 2, 4, 6, 10, (16)
- Class 5 - NS5: 1, 2, 4, 7, 12, (20)

numerical complex - operational simple

- Class 1 - NS6: 107, 291, 475, 659, 843, (1027)
- Class 2 - NS7: 237, 311, 386, 462, 539, (617)
- Class 3 - NS8: 128, 254, 381, 507, 634, (760)
- Class 4 - NS9: 103, 103, 206, 309, 515, (824)
- Class 5 - NS10: 1, 24, 47, 93, 162, (277)

numerical simple - operational complex

- Class 1 - NS11: 1, 4, 9, 16, 25, (36)
- Class 2 - NS12: 1, 1, 2, 6, 24, (120)
- Class 3 - NS13: 4, 6, 12, 14, 28, (30)
- Class 4 - NS14: 2, 2, 4, 8, 32, (256)
- Class 5 - NS15: 1, 1, 3, 12, 84, (924)

numerical complex - operational complex

- Class 1 - NS16: 121, 144, 169, 196, 225, (256)
- Class 2 - NS17: 7, 7, 14, 42, 168, (840)
- Class 3 - NS18: 16, 48, 51, 153, 156, (468)
- Class 4 - NS19: 2, 3, 6, 18, 108, (1944)
- Class 5 - NS20: 3, 3, 9, 36, 252, (2772)

2.2 Inductive Programming with MagicHaskeller

2.2.1 Inductive Programming

Induction of number series means the thinking about how to represent a number series as a valid function. (Martina, 2012). The way of induction with a system can be done differently. So on the one hand Inductive Programming is automatic programming (Flener and Schmid, 2008). This is the way of synthesizing small sets of input/output examples to generate programs. Afterwards these automatic generated programs will be checked to validation. This principle is often described as *generate-and-test approach*. Inductive programming systems like *MagicHaskeller on the Web* include this way.
On the other hand Inductive Programming is machine learning. The system synthesizes the small sets of input/output by investigating the inference of an algorithm or program (Flener and Schmid, 2008). This way is also known as the *analytical approach*. *IGOR2* or the latest versions of *MagicHaskeller* include this way of synthesizing input/output examples.

2.2.2 MagicHaskeller

MagicHaskeller is an inductive functional programming system based on systematic exhaustive search (Katayama, 2011). The system synthesizes given input-output specifications to find functions that represent the input-output specifications, e.g. $f[1..5] == 6 \Rightarrow f = (\backslash f \rightarrow 1 + length\ a)$. $f[1..5] == 6$ is the given input-output specification and $f = (\backslash f \rightarrow 1 + length\ a)$ is the solution or rather the represented function of the program. *Note:* the input-output specification is the syntax of *MagicHaskeller on the Web*! The solution of *MagicHaskeller* and *MagicHaskeller on the Web* would be the same here.
Since version 0.8.6 *MagicHaskeller* was extended by a module for analytical synthesis - the *analytical approach* (Katayama, 2011). The first versions included only the module for exhaustive search - the *generate-and-test approach*. The *generate-and-test approach* generates all possible programs dependend on the given set of functions and lambda abstractions. The complexity of combinations increases step by step, so *MagicHaskeller* starts with the easy tasks (Grünwald et al., 2012). Afterwards the system checks all these combinations against the input-output specification to satisfiability. Each satisfiable function is a potential solution and will be presented by the system.
The *analytical approach* is an extension to the *IGOR-algorithm* (Hofmann, 2012). Programs were synthesized by looking into the input-output specifications and conducting inductive inference (Katayama, 2011). The advantage of the extension by the *analytical approach* is that more functions can be synthesized. Fibonacci functions cannot be synthesized by the *analytical approach*, but by the *generate-and-test approach* and the *analytical approach* can synthesize some functions in a realistic time what the *generate-and-test approach* cannot do. The combination of both approaches is called *analytically-generate-and-test approach* (Katayama, 2011) which is maybe implemented in the latest versions of the *MagicHaskeller*.

2.2.3 MagicHaskeller on the Web

MagicHaskeller on the Web is as *MagicHaskeller* an inductive functional programming system. Instead of *MagicHaskeller*, *MagicHaskeller on the Web* is no stand-alone version - it's a web application. More differences are described in 2.2.4.

MagicHaskeller on the Web is available on `http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html`. If you visit the link you can directly use *MagicHaskeller on the Web*.
The first textbox below the caption „Use it now!“2.2 is for the input of the number serie. How to put in a valid expression for a number series will be explained in chapter 3.

By clicking the „Synthesize-Button“the system synthesizes the inserted expression and gives possible solutions if solutions were found. Below the textbox there is a caption „Examples“2.3. This caption shows possible valid expressions that can be used to use *MagicHaskeller on the Web*. As can be seen the system is also able to deal with Strings, Doubles and combinations of different data types.

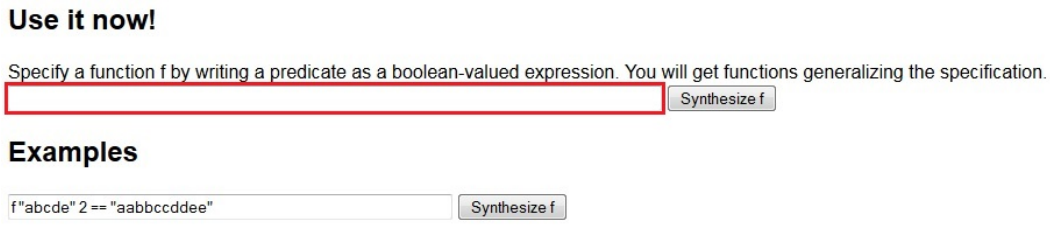

Figure 2.2: MagicHaskeller on the Web - Textfield for Input

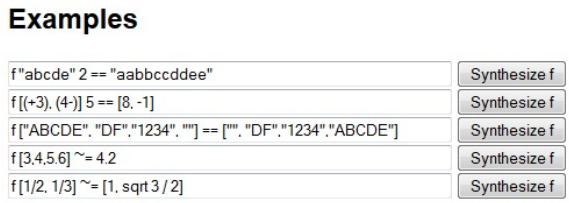

Figure 2.3: MagicHaskeller on the Web - Examples

If you synthesize the example f"*abcde*"2 == "*aabbccddee*" the system gives as the first solution f = (\a b → *concat*(*transpose*(*replicate* b a))). The solution is presented as implemented library functions, e.g. concat, transpose, replicate and more. A documentation will be opened if you click onto a function (Katayama, 2013). If the solution does not accord with the expected one, there is a button „More...“. By clicking the MORE-BUTTON the system will research for other solutions that present the input as a function and will give more solutions. If no MORE-BUTTON is available, the system does not find other solutions than presented already 2.4.

When all solutions are presented and there is no expected solution contained, the user has the possibility to click one of the EXEMPLIFY-BUTTON. If you do so you get a list of fragments of the exemplified function. If a value does not correspondent to the expected value, you can easily correct the value by changing the value in the suitable textbox and clicking the NARROW SEARCH-BUTTON. The function gets modified that the changed value is correct within the new function.

If completely no function was found by the system, there is a textbox above the SUGGEST-BUTTON. You can submit a correct expression in there and press the SUGGEST-BUTTON. This solution will be regarded as your suggestion and may be able to be synthesized by future versions.

2.2.4 MagicHaskeller vs. MagicHaskeller on the Web

This article deals with the *MagicHaskeller on the Web*, a partly translated version of the *MagicHaskeller*. *MagicHaskeller* is a stand-alone version which means that it has to be installed e.g. on a desktop PC. The problem of the stand-alone version is the special combination of the versions of the *operat-*

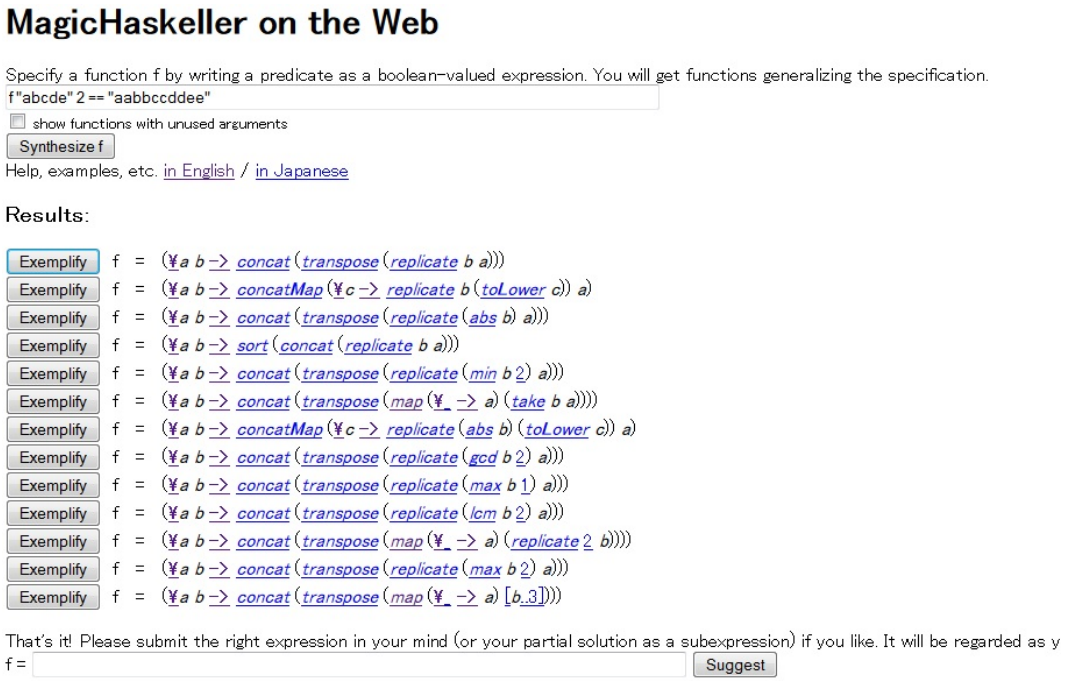

Figure 2.4: MagicHaskeller on the Web - All Solutions presented

ing system, the *Glasgow Haskell Compiler* and the *MagicHaskeller*, cause of the compatibility among each other. One possible combination for a running system would be a server of *Ubuntu 10.04*, *GHC 6.12.1* and *MagicHaskeller 0.8.6.3* A.1.

Another problem is that the *MagicHaskeller* is not updated very often (Katayama, 2015).

An advantage over the *MagicHaskeller on the Web* is the functional complexity: the latest version of *MagicHaskeller* includes both approaches - the *generate-and-test* and the analytical approach. *MagicHaskeller on the Web* has no module for *analytical synthesis*. So in theory the stand-alone version will be able to solve more number series problems than the web application.

The most important advantage of the *MagicHaskeller on the Web* over the stand-alone version of the *MagicHaskeller* is not the functionality, but the availability. It is easy to deal with the *MagicHaskeller on the Web* because it is a web application that can be executed from each PC with a connection to the internet. With the *MagicHaskeller on the Web* there is a tool that can be used for automated inductive programming without any problems with the compatibility of versions. In spite of the smaller complexity of this system this article deals with the *MagicHaskeller on the Web* because of the better availability and the more current maintenance.

MagicHaskeller on the Web is written in *Haskell* so the usage of this tool is as simple as a web search engine if syntax and semantics were recognized (Katayama, 2013). This article describes how to use the *MagicHaskeller on the Web* for solving number series problems. Furthermore this article response to components, limits and functionality of the *MagicHaskeller on the Web*.

Chapter 3

Induction on Number Series with MagicHaskeller on the Web

The main subject of this article is how to use *MagicHaskeller on the Web* to solve number series problems. The number series presented in 2.1.2 are 20 different problems depending on the combination of *Operational Complexity*, *Numerical Complexity* and *Structural Complexity*. To put it straight - this article shows which of these problems *MagicHaskeller on the Web* is able to fix and which ones *MagicHaskeller on the Web* is not able to fix. This article also shows how to create a valid input expression for the system and which expressions work better and which ones work worse. The solutions were reviewed to correctness in chapter 4.
MagicHaskeller on the Web is a system written in Haskell. So the system needs a Haskell expression returning a Boolean value without using a *let expressions* or *where clauses* (Katayama, 2015). If you do so you will get an error message „Error: let expressions and where clauses are prohibited here. You can still use case expressions without where clauses for non-recursive bindings“.
Another point is the count of numbers that will be used as an expression fed to *MagicHaskeller on the Web*. Five values should be enough for number series problems depending on one pre-successor. For a consistent work the number series with two pre-successors will be extended to ten values, so the favored solution should be more clearly for the system. The extended version of the number series including a possible solution can be seen in the appendix A.2. As can be seen not all number series depending on two pre-successor were extended to 10 values. The reason of that is the dimension of the values, e.g. the eighth value of NS19 is $408 * 10^6$. To work with such a high value makes less sense, so I decided to use 9999 as a maximum value. If values of a number series overrun 9999 they will not be included. The highest included values are flagged with brackets. *Note:* $fib(1, 1, 3)$ means the Fibonacci-Series $1, 1, 3, 4, 7, 11, 18, ...$!
The idea how to formulate different expressions to test *MagicHaskeller on the Web* on several ways consists of Hofmann et al. (2014). This cap includes induction of number series and *Igor2*, that is also an inductive functional program-

ming system like *MagicHaskeller (on the Web)*. Three possibilities to represent number series were defined:

- Input List - Output Successor Value
- Input Position - Output List
- Input Position - Output Value

This article uses all three ways two times - forwards and backwards. So there are six different ways to test input-output expression. In each of the following sections there is a description of the way how to form a valid Haskell expression for the system by the first number series as an example. All other number series were also tested. The solutions of the system of each number series combined with each way is written as an overview in the appendix A.3.1.

3.1 Input List - Output Successor Value

One of the presented ways is Input List - Output Successor Value. The input has to be presented as a list while the output has to be presented as the next value of the list. To work with *MagicHaskeller on the Web* you have to transform this statement into a valid Haskell expression. The system needs a function as an input. As an output the system expects a value like a number or a list. Translated to this way the required input is a function list and the output is the next successor value of the list. The first number series shall serve as an example for a correct feed. Given is the first number series $3, 5, 7, 9, 11$ and the next value should be 13, so there are five values that can be taken for a Haskell expression. The two possible expression analyzed in this article are

1. f [3, 5, 7, 9] == 11
2. f [9, 7, 5, 3] == 11

A possible solution given from Grünwald et al. (2012) is $f_n = f_{n-1} + 2$. The first solutions of *MagicHaskeller on the Web* are

1. $f = (\backslash a \rightarrow 2 + last(0 : a))$ for f [3, 5, 7, 9] == 11
2. $f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$ for f [9, 7, 5, 3] == 11

In A.3.1 there is an overview of all 20 number series and the first three solutions of *MagicHaskeller on the Web*. The evaluation of these solutions will be described in Chapter 4.

3.2 Input Position - Output List

Another way for a representation of an input is Input Position - Output List. It is the same principle as before: the system needs a function as an input and a value as an output. For this way the input function has to be presented as a position and the output as a list is. Translated into a correct Haskell expression it looks like

1. f 4 == [3, 5, 7, 9, 11]
2. f 4 == [11, 9, 7, 5, 3]

For this way of representing a number series problem, *MagicHaskeller on the Web* is not able to find any solutions (cf. A.3.2).

3.3 Input Position - Output Value

The third way to present a number series problem is Input Position - Output Value. The basic principle of creating the Haskell expression is the same like before, but now there are single values instead of the list. To generate a valid Haskell expression you have to link the values to their position. To demonstrate the whole number series you have to string together all links with the && - operator. Demonstrated as a valid Haskell expression it looks like

1. f 0 == 3 && f 1 == 5 && f 2 == 7 && f 3 == 9 && f 4 == 11
2. f 0 == 11 && f 1 == 9 && f 2 == 7 && f 3 == 5 && f 4 == 3

For presenting number series problems by this way, the system is able to find solutions.

1. $f = (\backslash a \rightarrow 1 + (2 * (1 + a)))$
2. $f = (\backslash a \rightarrow length(drop\ a\ [a..10]))$

The overview of all number series problems and the first three solutions by this way is illustrated in A.3.3.

Chapter 4

Realization and Evaluation

To analyze the system the extended number series that can be seen in A.2 were used as input-output sets. **Three forms** of an input-output set were tested to find out which form of input-output examples *MagicHaskeller on the Web* can deal with. As can be seen in A.3, *MagicHaskeller on the Web* is good in dealing with the form Input List - Output Successor Value. The system is completely not able to deal with Input Position - Output List and of limited suitability with Input Position - Output Value. Because of that only Input List - Output Successor Value is evaluated in this article.

Due to lack of time only the **first three solutions** (if three or more solutions were found) were evaluated, but both directions - forwards and backwards.

The differences between **forward** and **backward** input in reference to the first three solutions are minimal. If a correct solution for the number series was found, the system found the solution with both ways. Often the same solutions were found (cf. A.4.1 NS2) or the input in form of a list is reversed (cf. A.4.1 NS1). Another difference is the usage of function pairs like *drop* and *take*, but logical it is the same when a list is reversed. So I guess that there is no difference in finding solutions between the forward input and the backward input. To prove that you have to analyze and compare all solutions (not only three) of the system. Due to lack of time I did not.

To **proof the solutions** for correctness towards the input and towards the number series, as can be seen in A.4.1, the solutions were recognized or tested with the GHCi version 7.8.3.

While *MagicHaskeller on the Web* **finds a solution**, it is always a correct solution for the used input (cf. A.4.1). So with each solution you can continue the number series. Sometimes the solution was correct for the regarded number series, sometimes the solutions was only correct for another number series that have the same values at the beginning as the regarded one. This means that all solutions of the system were correct, but not forceful a correct solution to continue exactly the regarded number series.

The solutions for number series NS2, NS3', NS6, NS7, NS8', NS10', NS11, NS13', NS15', NS16, NS18' and NS20' were **no correct solutions** to continue the number series. It is possible to say that the system finds a correct solution, if a correct solution is shown in the first three solutions. But while only three solutions were regarded, it is not possible to say that the system is not able to find a correct solution if no correct solution was shown in the first three solutions. In favor you have to analyze all solutions prevented by the system. It is only possible to say that the first three solutions do not include a correct solution and this is not significant because it is possible that correct solutions can be found later (cf. A.4.1 NS12).

In the case of *MagicHaskeller* **finds no solution**, there might be two reasons for that. On the one hand the system is not able to solve the problem, on the other hand the systems stops automatically at some point to preserve stability of the server (Katayama, 2013). It can be observed that the system needs a lot of more time to find solutions for number series with large numbers. It is notable that all solutions that were not found are numerical complex (NS6, NS7, NS8', NS10', NS16). However, since the system stops automatically at some point to preserve stability of the server, you will not be able answer the question, if the system is not able to find possible solutions or if it depends on the automatic stop. *Note:* the time varies until the time-out occurs. One reason for that is the connection to the server.

Chapter 5

Conclusion and further Works

This article gives an introduction to *MagicHaskeller on the Web*. 2.2.3 describes how to use the system and 2.2.4 describes theoretically the differences between the stand-alone and the web version *MagicHaskeller*. A practical comparison can be done in further works. Also a comparison with *IGOR II* can be made, that would be more interesting.

2.1.2 shows the data that were used to analyze the system. All these series number were taken to find out which input expression is the best for the system. The best expression is Input List - Output Successor Value. Maybe there are more than the three presented possibilities of expressions that *MagicHaskeller on the Web* can deal with, but this has to be done in further works.

One can note that (depending on the used number series) the system is not good in dealing with large numbers. The system needs more time and after some time a time-out will occur. So if you want to find out whether the reason of No Solution depends on the system or the time-out, the programmer has to deactivate the time-out. In favor you have to contact Mr. Katayama.

Another open question is which of the number series *MagicHaskeller on the Web* can solve. As can be seen in A.4.1 the system is able to solve some problems with the first three solutions. But is the system able to solve more number series if you analyze more than three respectively all solutions? To answer this question and to prove my statement that there are no differences in finding solutions between the forward input and the backward input at Input List - Output Successor Value, you have to analyze all solutions.

During work I noticed that the used number series are not as good as I thought, so if this work will be continued I recommend to use number series like the Ninety-Nine Haskell Problems or others.

Bibliography

Erbrecht, R., Felsch, M., König, H., Kricke, W., Martin, K., Pfeil, W., Winter, R., and Wörstenfeld, W. (2003). *Das große Tafelwerk interaktiv - Formelsammlung für die Sekundarstufen I und II*. Cornelsen.

Flener, P. and Schmid, U. (2008). An introduction to inductive programming. *Artif. Intell. Rev.*, 29(1):45–62.

Grünwald, R., Heidel, E., Strätz, A., Sünkel, M., and Terbach, R. (2012). Induction on Number Series. Project Report.

Hofmann, J. (2012). Automatische Induktion über Zahlenreihen. Master's thesis, Uni Bamberg. Bachelor Thesis.

Hofmann, J., Kitzelmann, E., and Schmid, U. (2014). Applying Inductive Program Synthesis to Induction of Number Series - A Case Study with IGOR2. In *KI 2014: Advances in Artificial Intelligence*, pages 25–36. Springer.

Katayama, S. (2011). MagicHaskeller: System demonstration. Online; `http://www.cogsys.wiai.uni-bamberg.de/aaip11/accepted/katayama_short.pdf`; accessed 20.03.2015.

Katayama, S. (2012). An Analytical Inductive Functional Programming System that Avoids Unintended Programs. In *PEPM '12 Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation*, pages 43–52. ACM.

Katayama, S. (2013). MagicHaskeller on the Web: Automated Programming as a Service. Online; `https://www.haskell.org/haskell-symposium/2013/magichaskeller.pdf`; accessed 20.03.2015.

Katayama, S. (2015). MagicHaskeller: An Inductive Functional Programming System for Casual/Beginner Haskell Programmers. Online; `http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html`; accessed 20.03.2015.

Martina, M. (2012). Applying Inductive Programming to Solving Number Series Problems - Comparing Performance of IGOR with Humans. Master's Thesis.

Ragni, M. and Klein, A. (2011). Predicting Numbers: An AI Approach to Solving Number Series. In *KI 2011: Advances in Artificial Intelligence*, pages 255–259. Springer.

Appendix A

First Appendix Chapter

A.1 E-Mail about a possible Combination for a running System

Guten Morgen,

MagicHaskeller ist nun auf allen Laborrechnern eingerichtet. [...] Dann bootet ein Ubuntu 10.04 Server. [...] Es ist GHC6.12.1 mit dem MagicHaskeller0.8.6.3 installiert.

Mit besten Grüßen,
Michael Sünkel

A.2 Extended 20 Number Series

# Number Series	Original Number Series	Extended Number Series'	possible Solution
NS1	3,5,7,9,11,(13)	3,5,7,9,11,(13)	$f_n = f_{n-1} + 2$
NS2	2,7,13,20,28,(37)	2,7,13,20,28,(37)	$f_n = f_{n-1} + (n + 4)$
NS3	5,7,10,12,15,(17)	5,7,10,12,15,17, 20,22,25,27,(30)	$f_{n=2k-1} = f_{n-1} + 2$ || $f_{n=2k} = f_{n-1} + 3$
NS4	2,2,4,6,10,(16)	2,2,4,6,10,16, 26,42,68,110,(178)	$f_n = f_{n-1} + f_{n-2}$
NS5	1,2,4,7,12,(20)	1,2,4,7,12,20, 33,54,88,143,(232)	$f_n = f_{n-1} + f_{n-2} + 1$
NS6	107,291,475,659,843,(1027)	107,291,475,659,843,(1027)	$f_n = f_{n-1} + 184$
NS7	237,311,386,462,539,(617)	237,311,386,462,539,(617)	$f_n = f_{n-1} + (n + 73)$
NS8	128,254,381,507,634,(760)	128,254,381,507,634,760, 887,1013,1140,1266,(1393)	$f_{n=2k-1} = f_{n-1} + 126$ || $f_{n=2k} = f_{n-1} + 127$
NS9	103,103,206,309,515,(824)	103,103,206,309,515,824, 1339,2163,3502,5665,(9167)	$f_n = f_{n-1} + f_{n-2}$
NS10	1,24,47,93,162,(277)	1,24,47,93,162,277, 461,760,1243,2025,(3290)	$f_n = f_{n-1} + f_{n-2} + 22$
NS11	1,4,9,16,25,(36)	1,4,9,16,25,(36)	$f_n = n^2$
NS12	1,1,2,6,24,(120)	1,1,2,6,24,(120)	$f_n = f_{n-1} * n$
NS13	4,6,12,14,28,(30)	4,6,12,14,28,30, 60,62,124,126,(252)	$f_{n=2k-1} = f_{n-1} + 2$ || $f_{n=2k} = f_{n-1} * 2$
NS14	2,2,4,8,32,(256)	2,2,4,8,32,256, (8192),2097152,$17 * 10^9$,...	$f_n = f_{n-1} * f_{n-2}$
NS15	1,1,3,12,84,(924)	1,1,3,12,84,(924), 16632,$482{*}10^3$,...	$f_n = f_{n-1} * fib(1, 1, 3)$
NS16	121,144,169,196,225,(256)	121,144,169,196,225,(256)	$f_n = (n + 10)^2$
NS17	7,7,14,42,168,(840)	7,7,14,42,168,(840)	$f_n = f_{n-1} * n$
NS18	16,48,51,153,156,(468)	16,48,51,153,156,468, 471,1413,1416,(4248)	$f_{n=2k-1} = f_{n-1} * 3$ || $f_{n=2k} = f_{n-1} + 3$
NS19	2,3,6,18,108,(1944)	2,3,6,18,108,(1944), 209952,$408 * 10^6$,...	$f_n = f_{n-1} * f_{n-2}$
NS20	3,3,9,36,252,(2772)	3,3,9,36,252,(2772), 49896,1,$4{*}10^6$,...	$f_n = f_{n-1} * fib(1, 1, 3) * 3$

A.3 Solutions of MagicHaskeller on the Web

A.3.1 Solutions for Input List - Output Successor Value

Table A.1: forward Input NS1-NS10, e.g. f [3,5,7,9] == 11

# Number Series	Number Series	Solutions
NS1	3,5,7,9,11,(13)	$f = (\backslash a \rightarrow 2 + last(0 : a))$ $f = (\backslash a \rightarrow sum(drop\ 1\ a) - 10)$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse\ a)) - 4)$
NS2	2,7,13,20,28,(37)	$f = (\backslash a \rightarrow ord'' - length\ a)$ $f = (\backslash a \rightarrow ceiling(sinh(fromIntegral(length\ a))))$ $f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow exponent\ 100)\ a))$
NS3'	5,7,10,12,15,17, 20,22,25,27,(30)	$f = (\backslash a \rightarrow 2 + last(0 : a))$ $f = (\backslash a \rightarrow 3 * (1 + length\ a))$ $f = (\backslash a \rightarrow sum(take\ 2(drop\ 3\ a)))$
NS4'	2,2,4,6,10,16, 26,42,68,110,(178)	$f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse(2 : a))))$ $f = (\backslash a \rightarrow 2 + sum(drop\ 1(reverse\ a)))$
NS5'	1,2,4,7,12,20, 33,54,88,143,(232)	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(10 : a))))$ $f = (\backslash a \rightarrow 1 + sum(take\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse((1 + length\ a) : a))))$
NS6	107,291,475,659,843,(1027)	no solutions
NS7	237,311,386,462,539,(617)	$f = (\backslash a \rightarrow 1001 - last(0 : a))$
NS8'	128,254,381,507,634,760, 887,1013,1140,1266,(1393)	no solutions
NS9'	103,103,206,309,515,824, 1339,2163,3502,5665,(9167)	$f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow sum(drop(exponent\ 100)\ a))$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse(take\ 3(reverse\ a)))))$
NS10'	1,24,47,93,162,277, 461,760,1243,2025,(3290)	no solutions

Table A.2: forward Input NS11-NS20

# Number Series	Number Series	Solutions
NS11	1,4,9,16,25,(36)	$f = (\backslash a \rightarrow sum(drop\ 2\ a))$ $f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow sum(scanr1(\backslash\_\ \_ \rightarrow 3)\ a))$
NS12	1,1,2,6,24,(120)	$f = (\backslash a \rightarrow 2 * product\ a)$ $f = (\backslash a \rightarrow sum(scanr1(\backslash\_\ c \rightarrow c)\ a))$ $f = (\backslash a \rightarrow product(drop\ 1\ (reverse(a{+}{+}a))))$
NS13'	4,6,12,14,28,30, 60,62,124,126,(252)	$f = (\backslash a \rightarrow 2 + last(0 : a))$ $f = (\backslash a \rightarrow 1 + abs(1 + last(0 : a)))$ $f = (\backslash a \rightarrow 2 + abs(last(0 : a)))$
NS14'	2,2,4,8,32,256, (8192)	$f = (\backslash a \rightarrow product(drop\ 3\ a))$ $f = (\backslash a \rightarrow product(take\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow product(drop\ 1(reverse(2 : a))))$
NS15'	1,1,3,12,84,(924)	$f = (\backslash a \rightarrow 100 - sum(drop\ 1\ a))$ $f = (\backslash a \rightarrow 101 - sum\ a))$ $f = (\backslash a \rightarrow ceiling(1000/fromIntegral(last(0 : a))))$
NS16	121,144,169,196,225, (256)	no solutions
NS17	7,7,14,42,168,(840)	$f = (\backslash a \rightarrow sum(scanr1(\backslash\_\ c \rightarrow c)\ a))$ $f = (\backslash a \rightarrow 4 * last(0 : a))$ $f = (\backslash a \rightarrow (sum(drop\ 1(reverse(concat(replicate\ 3\ a))))))$
NS18'	16,48,51,153,156,468, 471,1413,1416,(4248)	$f = (\backslash a \rightarrow 3 + last(0 : a))$ $f = (\backslash a \rightarrow sum(nub(scanr1(\backslash\_\ \_ \rightarrow 3)\ a)))$ $f = (foldl(\backslash\_\ c \rightarrow 3 + c)\ 0)$
NS19'	2,3,6,18,108,(1944)	$f = (\backslash a \rightarrow product(drop\ 2\ a))$ $f = (\backslash a \rightarrow sum(concatMap(\backslash\_ \rightarrow drop\ 1\ a)\ a))$ $f = (\backslash a \rightarrow product(take\ 2(reverse\ a)))$
NS20'	3,3,9,36,252,(2772)	$f = (\backslash a \rightarrow exponent\ 100\ *\ last(0 : a))$ $f = (\backslash a \rightarrow sum(drop\ 1(a{+}{+}concatMap(\backslash\_ \rightarrow a)\ a)))$ $f = (\backslash a \rightarrow lcm(exponent\ 100)(last(0 : a)))$

Table A.3: backward Input NS1-NS10, e.g. f [9,7,5,3] == 11

# Number Series	Number Series	Solutions
NS1	3,5,7,9,11,(13)	$f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse\ a)) - 10)$ $f = (\backslash a \rightarrow sum(drop\ 1\ a) - 4)$
NS2	2,7,13,20,28,(37)	$f = (\backslash a \rightarrow ord'' - length\ a)$ $f = (\backslash a \rightarrow ceiling(sinh(fromIntegral(length\ a))))$ $f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow exponent\ 100)\ a))$
NS3'	5,7,10,12,15,17, 20,22,25,27,(30)	$f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow 3)\ a))$ $f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$ $f = (\backslash a \rightarrow sum(map(\backslash b \rightarrow min\ b\ 3)\ a))$
NS4'	2,2,4,6,10,16, 26,42,68,110,(178)	$f = (\backslash a \rightarrow sum(take\ 2\ a))$ $f = (\backslash a \rightarrow abs(sum(take\ 2\ a)))$ $f = (\backslash a \rightarrow 2 + sum(drop\ 1\ a))$
NS5'	1,2,4,7,12,20, 33,54,88,143,(232)	$f = (\backslash a \rightarrow 1 + sum(take\ 2\ a))$ $f = (\backslash a \rightarrow abs(1 + sum(take\ 2\ a)))$ $f = (\backslash a \rightarrow 1 + abs(sum(take\ 2\ a)))$
NS6	107,291,475,659,843,(1027)	no solutions
NS7	237,311,386,462,539,(617)	no solutions
NS8'	128,254,381,507,634,760, 887,1013,1140,1266,(1393)	no solutions
NS9'	103,103,206,309,515,824, 1339,2163,3502,5665,(9167)	$f = (\backslash a \rightarrow sum(take\ 2\ a))$ $f = (\backslash a \rightarrow abs(sum(take\ 2\ a)))$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse(take\ 3\ a))))$
NS10'	1,24,47,93,162,277, 461,760,1243,2025,(3290)	no solutions

Table A.4: backward Input NS11-NS20

# Number Series	Number Series	Solutions
NS11	1,4,9,16,25,(36)	$f = (\backslash a \rightarrow sum(take\ 2\ a))$ $f = (\backslash a \rightarrow sum(drop\ 2(reverse\ a)))$ $f = (\backslash a \rightarrow sum(scanl1(\backslash \_\ \_ \rightarrow 3)\ a))$
NS12	1,1,2,6,24,(120)	$f = (\backslash a \rightarrow 2 * product\ a)$ $f = (\backslash a \rightarrow sum(scanl1\ const\ a))$ $f = (\backslash a \rightarrow product(drop\ 1\ (a{+}{+}a)))$
NS13'	4,6,12,14,28,30, 60,62,124,126,(252)	$f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$ $f = (\backslash a \rightarrow abs(2 + foldr\ const\ 0\ a))$ $f = (\backslash a \rightarrow 1 + abs(1 + foldr\ const\ 0\ a))$
NS14'	2,2,4,8,32,256, (8192)	$f = (\backslash a \rightarrow product(take\ 2\ a))$ $f = (\backslash a \rightarrow product(drop\ 1(reverse(take\ 3\ a))))$ $f = (\backslash a \rightarrow abs(product(take\ 2\ a)))$
NS15'	1,1,3,12,84,(924)	$f = (\backslash a \rightarrow 100 - sum(drop\ 1(reverse\ a)))$ $f = (\backslash a \rightarrow 101 - sum\ a))$ $f = (\backslash a \rightarrow ceiling(1000/fromIntegral(foldr\ const\ 0\ a)))$
NS16	121,144,169,196,225, (256)	no solutions
NS17	7,7,14,42,168,(840)	$f = (\backslash a \rightarrow sum(scanl1\ const\ a))$ $f = (\backslash a \rightarrow 4 * foldr\ const\ 0\ a)$ $f = (\backslash a \rightarrow 3 * sum(take\ 2\ a))$
NS18'	16,48,51,153,156,468, 471,1413,1416,(4248)	$f = (\backslash a \rightarrow 3 + foldr\ const0\ a)$ $f = (\backslash a \rightarrow sum(nub(scanl1(\backslash \_\ \_ \rightarrow 3)\ a)))$ $f = (foldr(\backslash b \rightarrow 3 + b)\ 0)$
NS19'	2,3,6,18,108,(1944)	$f = (\backslash a \rightarrow product(take\ 2\ a))$ $f = (\backslash a \rightarrow product(drop\ 1(reverse(take\ 3\ a))))$ $f = (\backslash a \rightarrow abs(product(take\ 2\ a)))$
NS20'	3,3,9,36,252,(2772)	$f = (\backslash a \rightarrow exponent\ 100\ *\ foldr\ const\ 0\ a)$ $f = (\backslash a \rightarrow sum(drop\ 1(reverse(a{+}{+}concatMap(\backslash \_ \rightarrow a)\ a))))$ $f = (\backslash a \rightarrow lcm(exponent\ 100)(foldr\ const\ 0\ a))$

A.3.2 Solutions for Input Position - Output List

Table A.5: forward and backward Input, e.g. f 4 == [3,5,7,9,11] or f 4 == [11,9,7,5,3]

# Number Series	Number Series	Solutions
NS1	3,5,7,9,11,(13)	no solutions
NS2	2,7,13,20,28,(37)	no solutions
NS3'	5,7,10,12,15,17,20,22,25,27,(30)	no solutions
NS4'	2,2,4,6,10,16,26,42,68,110,(178)	no solutions
NS5'	1,2,4,7,12,20,33,54,88,143,(232)	no solutions
NS6	107,291,475,659,843,(1027)	no solutions
NS7	237,311,386,462,539,(617)	no solutions
NS8'	128,254,381,507,634,760,887,1013,1140,1266,(1393)	no solutions
NS9'	103,103,206,309,515,824,1339,2163,3502,5665,(9167)	no solutions
NS10'	1,24,47,93,162,277,461,760,1243,2025,(3290)	no solutions
NS11	1,4,9,16,25,(36)	no solutions
NS12	1,1,2,6,24,(120)	no solutions
NS13'	4,6,12,14,28,30,60,62,124,126,(252)	no solutions
NS14'	2,2,4,8,32,256,(8192)	no solutions
NS15'	1,1,3,12,84,(924)	no solutions
NS16	121,144,169,196,225,(256)	no solutions
NS17	7,7,14,42,168,(840)	no solutions
NS18'	16,48,51,153,156,468,471,1413,1416,(4248)	no solutions
NS19'	2,3,6,18,108,(1944)	no solutions
NS20'	3,3,9,36,252,(2772)	no solutions

A.3.3 Solutions for Input Position - Output Value

Table A.6: forward Input, e.g. f 0 == 3 && f 1 == 5 && f 2 == 7 && f 3 == 9 && f 4 == 11

# Number Series	Number Series	Solutions
NS1	3,5,7,9,11,(13)	$f = (\backslash a \rightarrow 1 + (2 * (1 + a)))$ $f = (\backslash a \rightarrow 1 + sum(replicate * (1 + a)\ 2))$ $f = (\backslash a \rightarrow abs(1 + (2 * (1 + a))))$
NS2	2,7,13,20,28,(37)	no solutions
NS3'	5,7,10,12,15,17,...	no solutions
NS4'	2,2,4,6,10,16,26,...	no solutions
NS5'	1,2,4,7,12,20,...	no solutions
NS6	107,291,475,659,...	no solutions
NS7	237,311,386,462,...	no solutions
NS8'	128,254,381,507,...	no solutions
NS9'	103,103,206,309,...	no solutions
NS10'	1,24,47,93,162,...	no solutions
NS11	1,4,9,16,25,(36)	$f = (\backslash a \rightarrow product(replicate\ 2(1 + a)))$ $f = (\backslash a \rightarrow (1 + a) * (1 + abs\ a))$ $f = (\backslash a \rightarrow (1 + a) * abs(1 + a))$
NS12	1,1,2,6,24,(120)	$f = (\backslash a \rightarrow product[1..a])$ $f = (\backslash a \rightarrow product[1..abs\ a])$ $f = (\backslash a \rightarrow product(take\ 3[2..a]))$
NS13'	4,6,12,14,28,30,...	no solutions
NS14'	2,2,4,8,32,256,(8192)	no solutions
NS15'	1,1,3,12,84,924,...	no solutions
NS16	121,144,169,196,...	no solutions
NS17	7,7,14,42,168,(840)	no solutions
NS18'	16,48,51,153,...	no solutions
NS19'	2,3,6,18,108,...	no solutions
NS20'	3,3,9,36,252...	no solutions

Table A.7: backward Input, e.g. f 0 == 11 && f 1 == 9 && f 2 == 7 && f 3 == 5 && f 4 == 3

# Number Series	Number Series	Solutions
NS1	3,5,7,9,11,(13)	f = (\a→ *length*(*drop* a [a..10])) f = (\a→ 11-(a+a)) f = (\a→ *length*[a..10-a])
NS2	2,7,13,20,28,(37)	no solutions
NS3'	5,7,10,12,15,17, 20,22,25,27,(30)	no solutions
NS4'	2,2,4,6,10,16, 26,42,68,110,(178)	no solutions
NS5'	1,2,4,7,12,20, 33,54,88,143,(232)	no solutions
NS6	107,291,475,659,843,(1027)	no solutions
NS7	237,311,386,462,539,(617)	no solutions
NS8'	128,254,381,507,634,760, 887,1013,1140,1266,(1393)	no solutions
NS9'	103,103,206,309,515,824, 1339,2163,3502,5665,(9167)	no solutions
NS10'	1,24,47,93,162,277, 461,760,1243,2025,(3290)	no solutions
NS11	1,4,9,16,25,(36)	no solutions
NS12	1,1,2,6,24,(120)	f = (\a→ *product*[1..4 - a])
NS13'	4,6,12,14,28,30, 60,62,124,126,(252)	no solutions
NS14'	2,2,4,8,32,256, (8192)	no solutions
NS15'	1,1,3,12,84,(924)	no solutions
NS16	121,144,169,196,225,(256)	no solutions
NS17	7,7,14,42,168,(840)	no solutions
NS18'	16,48,51,153,156,468, 471,1413,1416,(4248)	no solutions
NS19'	2,3,6,18,108,(1944)	no solutions
NS20'	3,3,9,36,252,(2772)	no solutions

A.4 Evaluation of Solutions

A.4.1 Evaluation for Input List - Output Successor Value

Table A.8: Evaluation of forward Input NS1-NS10, e.g. f [3,5,7,9] == 11

# NS	Solutions	correct for Input	correct for Number Series
NS1	$f = (\backslash a \rightarrow 2 + last(0 : a))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop\ 1\ a) - 10)$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse\ a)) - 4)$	Yes	No
NS2	$f = (\backslash a \rightarrow ord'' - length\ a)$	Yes	No
	$f = (\backslash a \rightarrow ceiling(sinh(fromIntegral(length\ a))))$	Yes	No
	$f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow exponent\ 100)\ a))$	Yes	No
NS3'	$f = (\backslash a \rightarrow 2 + last(0 : a))$	Yes	No
	$f = (\backslash a \rightarrow 3 * (1 + length\ a))$	Yes	No
	$f = (\backslash a \rightarrow sum(take\ 2(drop\ 3\ a)))$	Yes	No
NS4'	$f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(2 : a))))$	Yes	Yes
	$f = (\backslash a \rightarrow 2 + sum(drop\ 1(reverse\ a)))$	Yes	Yes
NS5'	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(10 : a))))$	Yes	No
	$f = (\backslash a \rightarrow 1 + sum(take\ 2(reverse\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse((1 + length\ a) : a))))$	Yes	Yes
NS6	no solutions	-	-
NS7	$f = (\backslash a \rightarrow 1001 - last(0 : a))$	Yes	No
NS8'	no solutions	-	-
NS9'	$f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop(exponent\ 100)\ a))$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(take\ 3(reverse\ a)))))$	Yes	Yes
NS10'	no solutions	-	-

Table A.9: Evaluation of forward Input NS11-NS20

# NS	Solutions	correct for Input	correct for Number Series
NS11	$f = (\backslash a \rightarrow sum(drop\ 2\ a))$	Yes	No
	$f = (\backslash a \rightarrow sum(take\ 2(reverse\ a)))$	Yes	No
	$f = (\backslash a \rightarrow sum(scanr1(\backslash \_\ \_ \rightarrow 3)\ a))$	Yes	No
NS12	$f = (\backslash a \rightarrow 2 * product\ a)$	Yes	No
	$f = (\backslash a \rightarrow sum(scanr1(\backslash \_\ c \rightarrow c)\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow product(drop\ 1\ (reverse(a{+}{+}a))))$	Yes	No
NS13'	$f = (\backslash a \rightarrow 2 + last(0 : a))$	Yes	No
	$f = (\backslash a \rightarrow 1 + abs(1 + last(0 : a)))$	Yes	No
	$f = (\backslash a \rightarrow 2 + abs(last(0 : a)))$	Yes	No
NS14'	$f = (\backslash a \rightarrow product(drop\ 3\ a))$	Yes	No
	$f = (\backslash a \rightarrow product(take\ 2(reverse\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow product(drop\ 1(reverse(2 : a))))$	Yes	Yes
NS15'	$f = (\backslash a \rightarrow 100 - sum(drop\ 1\ a))$	Yes	No
	$f = (\backslash a \rightarrow 101 - sum\ a))$	Yes	No
	$f = (\backslash a \rightarrow ceiling(1000/fromIntegral(last(0 : a))))$	Yes	No
NS16	no solutions	-	-
NS17	$f = (\backslash a \rightarrow sum(scanr1(\backslash \_\ c \rightarrow c)\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow 4 * last(0 : a))$	Yes	No
	$f = (\backslash a \rightarrow (sum(drop\ 1(reverse(concat(replicate\ 3\ a))))))$	Yes	No
NS18'	$f = (\backslash a \rightarrow 3 + last(0 : a))$	Yes	No
	$f = (\backslash a \rightarrow sum(nub(scanr1(\backslash \_\ \_ \rightarrow 3)\ a)))$	Yes	No
	$f = (foldl(\backslash \_\ c \rightarrow 3 + c)\ 0)$	Yes	No
NS19'	$f = (\backslash a \rightarrow product(drop\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(concatMap(\backslash \_ \rightarrow drop\ 1\ a)\ a))$	Yes	No
	$f = (\backslash a \rightarrow product(take\ 2(reverse\ a)))$	Yes	Yes
NS20'	$f = (\backslash a \rightarrow exponent\ 100\ *\ last(0 : a))$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 1(a{+}{+}concatMap(\backslash \rightarrow a)\ a)))$	Yes	No
	$f = (\backslash a \rightarrow lcm(exponent\ 100)(last(0 : a)))$	Yes	No

Table A.10: Evaluation of backward Input NS1-NS10, e.g. f [9,7,5,3] == 11

# NS	Solutions	correct for Input	correct for Number Series
NS1	$f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse\ a)) - 10)$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 1\ a) - 4)$	Yes	No
NS2	$f = (\backslash a \rightarrow ord'' - length\ a)$	Yes	No
	$f = (\backslash a \rightarrow ceiling(sinh(fromIntegral(length\ a))))$	Yes	No
	$f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow exponent\ 100)\ a))$	Yes	No
NS3'	$f = (\backslash a \rightarrow sum(map(\backslash\_ \rightarrow 3)\ a))$	Yes	No
	$f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$	Yes	No
	$f = (\backslash a \rightarrow sum(map(\backslash b \rightarrow min\ b\ 3)\ a))$	Yes	No
NS4'	$f = (\backslash a \rightarrow sum(take\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow abs(sum(take\ 2\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow 2 + sum(drop\ 1\ a))$	Yes	Yes
NS5'	$f = (\backslash a \rightarrow 1 + sum(take\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow abs(1 + sum(take\ 2\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow 1 + abs(sum(take\ 2\ a)))$	Yes	Yes
NS6	no solutions	-	-
NS7	no solutions	-	-
NS8'	no solutions	-	-
NS9'	$f = (\backslash a \rightarrow sum(take\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow abs(sum(take\ 2\ a)))$	Yes	Yes
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(take\ 3\ a))))$	Yes	Yes
NS10'	no solutions	-	-

Table A.11: Evaluation of backward Input NS11-NS20

# NS	Solutions	correct for Input	correct for Number Series
NS11	$f = (\backslash a \rightarrow sum(take\ 2\ a))$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 2(reverse\ a)))$	Yes	No
	$f = (\backslash a \rightarrow sum(scanl1(\backslash \_\ \_ \rightarrow 3)\ a))$	Yes	No
NS12	$f = (\backslash a \rightarrow 2 * product\ a)$	Yes	No
	$f = (\backslash a \rightarrow sum(scanl1\ const\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow product(drop\ 1\ (a{+}{+}a)))$	Yes	No
NS13'	$f = (\backslash a \rightarrow 2 + foldr\ const\ 0\ a)$	Yes	No
	$f = (\backslash a \rightarrow abs(2 + foldr\ const\ 0\ a))$	Yes	No
	$f = (\backslash a \rightarrow 1 + abs(1 + foldr\ const\ 0\ a))$	Yes	No
NS14'	$f = (\backslash a \rightarrow product(take\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow product(drop\ 1(reverse(take\ 3\ a))))$	Yes	Yes
	$f = (\backslash a \rightarrow abs(product(take\ 2\ a)))$	Yes	Yes
NS15'	$f = (\backslash a \rightarrow 100 - sum(drop\ 1(reverse\ a)))$	Yes	No
	$f = (\backslash a \rightarrow 101 - sum\ a))$	Yes	No
	$f = (\backslash a \rightarrow ceiling(1000/fromIntegral(foldr\ const\ 0\ a)))$	Yes	No
NS16	no solutions	-	-
NS17	$f = (\backslash a \rightarrow sum(scanl1\ const\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow 4 * foldr\ const\ 0\ a)$	Yes	No
	$f = (\backslash a \rightarrow 3 * sum(take\ 2\ a))$	Yes	No
NS18'	$f = (\backslash a \rightarrow 3 + foldr\ const 0\ a)$	Yes	No
	$f = (\backslash a \rightarrow sum(nub(scanl1(\backslash \_\ \_ \rightarrow 3)\ a)))$	Yes	No
	$f = (foldr(\backslash b \rightarrow 3 + b)\ 0)$	Yes	No
NS19'	$f = (\backslash a \rightarrow product(take\ 2\ a))$	Yes	Yes
	$f = (\backslash a \rightarrow product(drop\ 1(reverse(take\ 3\ a))))$	Yes	Yes
	$f = (\backslash a \rightarrow abs(product(take\ 2\ a)))$	Yes	Yes
NS20'	$f = (\backslash a \rightarrow exponent\ 100\ *\ foldr\ const\ 0\ a)$	Yes	No
	$f = (\backslash a \rightarrow sum(drop\ 1(reverse(a{+}{+}concatMap(\backslash \_ \rightarrow a)\ a))))$	Yes	No
	$f = (\backslash a \rightarrow lcm(exponent\ 100)(foldr\ const\ 0\ a))$	Yes	No