Fakultät Wirtschaftsinformatik und Angewandte Informatik

Otto-Friedrich-Universität Bamberg

KogSys-Proj-M: Master-Project Cognitive Systems

Alternating Number Series Problems in IGOR 2

Barbora Hrdá (Matr. No. 1761088)
Dea Svoboda (Matr. No. 1757401)

Project Report

WS 2014/15

Supervisor: Prof. Dr. Ute Schmid

Abstract

The inductive programming system IGOR2 was tested in a previous study in solving number series problems in IQ tests. It was shown that it had difficulties with solving alternating number series, i.e. the series consists basically of least two separate intertwined series. Due to IGOR2's inductive approach, at least three examples are necessary for a hypothesis construction for problems where the regularity can be observed from directly succeeding examples. Thus for a successful induction with an alternating series the program would need at least two times three examples if the series consists of just two series. In a previous study number series which contained only five examples were tested, hence they were at least one example too short for alternating ones. The scope of this project is to investigate if IGOR2 can be enabled to solve alternating number series problems successfully by giving it more examples. We tested five series and extended their length from five up to six respectively eight examples. The results showed that IGOR2 does not need more examples if it has to solve an unambiguous alternating series. Ambiguous series are solveable without providing an alternating hypothesis. This is possible for all alternating series using only one operator, or a combination of contrary operators. Unambiguous series are, still, unsolvable. The length of a number series in IGOR2 has not a relevant impact on the solvability of alternating number series.

Contents

Chapter 1

Introduction

The solving of number series is an aspect in IQ (Intelligence Quotient) tests. Those are used to test the abilities of participants in pattern recognition, deriving rules from these patterns and applying those rules to complete the number series. Number series problems as research questions was picked up in 2003 within the research field of artificial intelligence by Sanghi and Dowe (2003). One approach was to use the inductive programming system IGOR2, a program which was initially not developed for solving number series problems. Nonetheless it can be applied to such problems without specializing its induction algorithm or its bias. Milovec (2014) used IGOR2 in combination with the programming language Maude for automatic number series completion. In her master thesis she tested twenty number series. Those comprise different levels of complexity and difficulty e.g. series constructed with one operator (+ or *) or such with more and those combined with using small or large numbers.

These series also include alternating and Fibonacci series. Alternating number series mean here that the values of a series are calculated by two separate equations alternating each other. Simple speaking there are two different number series which are entwined and, therefore, the number series cannot be calculated by just one formula. IGOR2 establishes equations with an initial hypothesis and develops those further in an alternative procedure, which will be discussed more in detail in chapter two. The hypotheses are created via input/output examples and background knowledge. Milovec (2014) tested integer number sequences with the length five, i.e. each series consists of five numbers also called examples. Five examples are being used for inducting the following sixth number. IGOR2 was able to solve 13 out of 20 number series within milliseconds up to 1 hour and 25 minutes. The not solvable set of series included among others all four provided alternating sequences. We think that there are two possible causes for this problem:

- too few examples, i.e. the used sequence is too short,
- insufficient background knowledge.

The scope of this project will focus on the first identified cause. An alternating number series consists basically of at least two separate intertwined series. Therefore, IGOR2 needs two hypotheses for such a series, i.e. one for each alternating series. With only five examples it is hence not possible for IGOR2 to

test the hypothesis for the second series, because it has too few examples. This results in three examples for "the first" and only two for "the second" subsequence, but you need at least three examples each for a successful induction, accordingly the whole sequence is at least one number too short.

In this project we want to answer the question if it possible to empower IGOR2 to solve also alternating series when expanding these to a length of six, eight and a maximum of ten. The used alternating number series consist of two subsequences each. Thus the program would have at least three examples for each of the two subsequence to tests its hypotheses, , the minimum of what it needs to compute a hypothesis. Reaching the length of ten IGOR2 will have five examples for each subsequence, which is the same amount it has for the non alternating series. The result will show whether the background knowledge of IGOR2 is sufficient or not to solve alternating sequences.

In chapter 2 we will describe the state of research in the field of solving number series problems. Chapter 3 will shortly introduce the functionality of IGOR2 and the existing work with it concerning number series, which is the footage of our project. Afterwards, in chapter 4, we will present our approach to solve IGOR2's problem with alternating series. Furthermore, we will explain which number series are tested and why. The results of our implementation are summarized and analysed in chapter 5.

Chapter 2

Approaches in Automatic Solving of Number Series

In the second chapter of our project report we want to provide an insight on the general research field of the automatic solving of number series problems as found in IQ tests for humans. As a first step we will describe what characterizes IQ tests and number series as a subset of IQ tests. Secondly we introduce the research field of artificial intelligence which developed an interest in the question on how computers perform on standard IQ tests (and what conclusions it could bring for the measuring of the intelligence of machines). Furthermore, we will describe shortly which systems are used for solving number series problems.

2.1 IQ Tests and number series

What is intelligence and how to measure it? Intelligence subsumes a large field of different definitions and due to this the range of the available tests varies. In fact, there is no definition of intelligence. One could simply state that "intelligence is what an intelligence test measures". Since the scope of this work is mainly the part of intelligence tests which include examples with number series, we will use the term intelligence for describing the ability to derive rules and patterns from a given series of natural numbers.

So called IQ tests were developed by humans to measure the intelligence of humans in a standardised way. There are many variants of tests respective to language, nation, challenges and also if the age of the participants is used as a value for the final calculation of the IQ or it is already taken account of through the questions asked. The test itself, as designed by Alfred Binet, should show the mental age of a person and with this value the IQ is calculated.

The IQ of a test person expresses in a number how far apart the mental and chronological age of a person is. This is calculated by dividing the chronological age through the mental age of this person and multiply it with 100. Is the result 100 it signifies that the person has the same age mentally and chronologically, a lower value that the mental age is below the chronological age and a higher value that the mental age is above (Sanghi and Dowe, 2003). In most cases according to Sanghi and Dowe (2003) they consist of:

1. Insert missing number,
   (a) At end
   (b) In middle
2. Insert missing letter,
   (a) At end
   (b) In middle
3. Insert suffix/prefix to complete two or more words,
4. Complete matrix of numbers/characters,
5. Questions involving directions,
6. Questions involving comparison,
7. Picture questions,
8. Pick the odd man out (word or picture).

The solving of number series, "insert number at the end", is one aspect in those tests. This part is used to test the abilities of participants in pattern recognition, deriving rules from these patterns and applying those rules to complete the number series.

IQ tests face often a lot of criticism due to the abilities tested and that the test itself is always made for a specific educated stratum, cultural and/or linguistic background. This means that two people with the same intelligence would score different if one of them is and the other is not a native speaker of the language needed to achieve a good score in the test. Some approaches like Strannegård (2013) try to apply human problem solving on machines. In 1973 Kotovsky and Simon (1973) described human reasoning on pattern detection problems. Holzman et al. (1983) refined it twenty years later. These contain:

- Relation detection: A relation between examples is determined and a hypothesis about how the examples are linked is created.
- Discovery of periodicity: The assumed relation between examples is tested and potential breaks are discovered. If breaks appear the length of periods is identified where arithmetic operations recur.
- Completion of Pattern Description: The hypothesis is tested on the whole series. The applied rules should cover all relations between examples else wise new relations are determined and tested.
- Extrapolation: In the last step the hypothesis is applied to generate the following number and complete the series.

These steps of human reasoning are applied on machine and one of the approaches is IGOR2 which we will introduce in the second part of this work.

2.2 Artificial Intelligence and IQ Tests

Number series problems as research question was picked up in 2003 within the research field of artificial intelligence. According to Hofmann (2012) a number series is defined as $f(n) = x$ with domain of definition $\mathbb{N}$ and co-domain $\mathbb{R}$. $n \in \mathbb{N}$ with n ≥ 1 defines the position within the series. The co-domain $\mathbb{R}$ consists in IQ tests of natural numbers, although other like decimal numbers are also possible. Rules between examples can be explicit or recursive. Explicit functions use the position n to calculate the next example. Recursive functions use the predecessor to complete the series. Many series can be defined by either one of the functions or both. The following sequence can be defined through a recursive or a explicit function:

Example

Number Series: 3, 5, 7, 9, 11, ...

- explicit: $f(n) = (2 * n) + 1$
- recursive: $f(n) = f(n-1) + 2$

The next number of the series is 13 no matter which function you use. The goal of the induction is to create a hypothesis about the relations between examples (like in IGOR2) or to continue the series with a concrete number.

Difficulties with number series. In fact, there are infinite valid possibilities to solve a number series. However, in IQ tests only one solution is right. A number series has a unique solution. Sanghi and Dowe (2003) concerned themselves with how IQ Tests can be presented to a computer program and how it can solve those questions. Their initial point is Alan Turings Paper about the Imitation Game, because it also tries to measure the intelligence of an machine to determine if a machine can think. In their paper they describe common questions asked in frequently used IQ Tests and how a machine could be enabled to solve those questions. They use their own program, written in Pearl, to take various IQ Tests and it scored like the average human, who passed these tests.

Concerning the completion of number series they state that their program solves questions of IQ tests from this type successful, in most of the time. It can find the next number of a sequence if this sequence is from a certain type such as arithmetic progression, geometric progression, Fibonacci series, powers of a series etc. As input it needs at least three numbers to find a pattern. The detailed mechanics of how their program solved the number series questions was, however, not part of the paper. A central point from this paper for our work is that if one wants to measure the intelligence of a computer one should not write a specific program to solve specific problems: “The program should ideally not be test specific, as it would be rather abnormal to get a score of over 120 on one I.Q. test and under 50 on another test.” IGOR2, which was not build to solve such questions, is such a program.

2.3 Methods applied to number series problems

Number sequence problems have been faced with many different methods: artificial neuronal networks (ANN), learning models, graphs, inductive programming and others. Out of the many systems dealing with number series problems, we will shortly introduce four of them: MagicHaskeller, E-generalization, Artificial Neuronal Networks, IGOR2. Since IGOR2 is the focus of our work we will discuss this one more in detail in the next chapter.

2.3.1 Generic Symbolic Approaches

We want to shortly introduce three generic symbolic methods. First the enumerative approach MagicHaskeller and afterwards analytical approaches IGOR2 and E-generalization.

MagicHaskeller

MagicHaskeller by Katayama (2005) is a search-based synthesiser which generates Hakell programs, see also Hofmann et al. (2008b). The program follows an analytically-generate-and-test approach and is so able to produce a lot of programs from input/output examples and then takes the ones which satisfy the initial specifications (Katayama, 2011). A study done by Düsel et al. (2012) showed that the MagicHaskeller can solve number series problems, even though it solved only seven out of twenty series.

E-generalization

E-generalization was introduced by Burghardt (2005). It combines theoretical computer science and predicate logic and is also called E-anti-Unification. It combines the usage of background knowledge in form of equational theory with anti-unification. Equational theories represent a set finite of term equations, see also Hofmann et al. (2008b). It is used to solve several IQ test problems.

IGOR2

IGOR2 is focused on learning recursive programs, by partitioning given input examples, which is done systematically and completely instead of randomly or greedy as other systems do. IGOR2 also works parallel when creating subsets while computing. It further unites analytical program synthesis with search (Hofmann et al., 2008b).

2.3.2 Rule-based Symbolic Approaches: Asolver

Some approaches like Strannegård (2013)'s Asolver try to apply human problem solving on machines. According to Strannegård (2013) we have to understand human reasoning in number series problems because IQ test are created by humans and serve the purpose to measure first and foremost human intelligence. Asolver combines artificial intelligence and cognitive psychology. It uses bounded computation and search for pattern matching to solve integer sequences. It's biggest advantage is its possibility to discard patterns based on

human reasoning. Patterns that seem to be to challenging for humans are excluded and enables Asolver to find patterns in number series faster (Strannegård, 2013)

2.3.3 Artificial Neuronal Networks (ANN)

Another method in number series problems whose foundations can be found in the artificial intelligence domain is represented by Ragni and Klein (2011). Their approach uses dynamically learning within Artificial Neural Networks (ANN) to solve number series problems. In their approach they use three-layered networks with error back-propagation and hyperbolic tangent as activation. They varied input nodes, hidden nodes, and learning rate to compare the different artificial neural networks structures. In a further step they even varied the number of training iterations. The focus of the ANNs presented by Ragni and Klein (2011) differs from other approaches. They do not try to reproduce the underlying equations of a series but to complete the series with the correct integer. Unfortunately both approaches, E-generalization and ANNs, are not meant to be cognitive models. But there is another approach that is intended as a cognitive model: IGOR2.

Chapter 3

Number Series Problems with IGOR2

In this chapter we want to focus on the state of the art concerning only IGOR2 and the solving of number series problems. Therefore, we will introduce the papers that build the base of our own work. We will start giving an insight on how IGOR2's algorithm works and of the work already done with it in combination with number series problems. This should form a level which enables to understand the following chapter where we present the problem we tried to solve and how we implement and analysed it.

3.1 IGOR2

To complement the short description of IGOR2 in the previous chapter we try to give now a more detailed look on the system and its functionality. We use in our project the analytical inductive programming system IGOR2. IGOR2 was developed with the focus on function induction over different lists, matrices, natural numbers and artificial intelligence problem solving (Kitzelmann, 2010). It induces programs from input/output examples. By analysing these examples it searches for recursive generalization and uses best first search. Furthermore, it is able to learn functional programs. IGOR2 is implemented in the reflective rewriting based programming and specification language Maude and can, therefore, write programs itself (Kitzelmann and Hofmann, 2008). IGOR2 is data-driven. It analytically constructs recursive first-order equations. Only equation sets entailing the example equations are enumerated (Kitzelmann and Hofmann, 2008).

Inductive Programming (IP) subsumes several approaches concerning the synthesis of logical or functional programs and when one wants to use methods of induction. Different programming languages can be used and because of their special traits the IP can be classified in the sub fields inductive logic programming (ILP), inductive functional programming (IFP) and inductive functional logic programming (IFLP), see also Hofmann et al. (2008a). IGOR2 together with the language Maude is an IFP and combining an analytical approach with a search in the problem space, rather then a generate-and-test. This approach means that after an analysis of Input/Output examples the system, here IGOR2,

is enabled to generate recursive functions (Kitzelmann, 2007). The algorithm of IGOR2 starts with a set of defined recursive equations and constructors. Equations that define a function are constructed as rewriting rules from the left-hand side of an equation to the right-hand side. To generate a correct, meaning consistent (every input has an output) and complete (all I/O examples were used), hypothesis (Kitzelmann, 2007), IGOR2 tries to bind all variables on the right-hand side, i.e. the variables must also occur on the left-hand side. In the initial step a hypothesis and a set of rules, containing least general generalizations of the example equation, is generated. Because of the incompleteness, unbound variables on the right-hand side, of this hypothesis a successor hypothesis is calculated. IGOR2 takes one rule of the unfinished rules and through the applying of operations computes a new set of rules, which then substitute the former ones in the hypothesis. IGOR2 then evaluates the found hypotheses and chooses the best to repeat the described steps. One can also give IGOR2 background knowledge as defined equations to enhance the calculation process. Such knowledge can be as what the result of the addition of two numbers would be or a small multiplication table.

3.2 IGOR2 compared with other AI Systems

Hofmann (2012) compared IGOR2's performance with the systems MagicHaskeller, E-Generalization and Artificial Neuronal Networks (ANN). She showed that IGOR2 can solve number series problems as successfully as the other systems, in many cases it was even faster than the others. Hofmann (2012) tested 100 number series including 11 alternating series. IGOR2 could solve 2 out of 11 but did perform equal to MagicHaskeller and even better than E-Generalization. Only ANN could solve 4 out of 11, which is equal to a success rate of 36,36%. In comparison with human test persons all systems performed worse in the topic of alternating series. Hofmann (2012) shows different types of alternating series:

- alternating arithmetic operations effect the direct predecessor,
- alternating arithmetic operations effect the predecessor of the predecessor and thus create two independent number series within one series.

IGOR2 could solve the two following series correctly:

- Alter1: 7,10,9,12,11 $\rightarrow f(n) = f(n-1) + 3; f(n-1) - 1$
- Alter5: 0,1,2,1,4,1 $\rightarrow f(n) = f(n-2) + 2; 1$

3.3 IGOR2 compared with human performance

In her masters thesis, Milovec (2014) applies inductive programming to the solving of number series problems. She also compares the performance of IGOR2 with those of humans. She tested 20 number series with various complexity levels, which were achieved by varying the grades of operational and numerical complexity and by how those two where combined. The operators in use for all series were addition (+) and multiplication (*) as other operators such as subtraction (-) and division (/) could result in not having an endless series. The

“ + ” operator was declared as a not so complex operator as “ * ”, because it needs less rewrites in IGOR2 and costs less computer resources. Numerical complexity was defined by having numbers in the series which were smaller or bigger than 100 and that smaller numbers are less complex, because it costs IGOR2 less time and resources to work with them. Our work focuses no so much on the difference between IGOR2 and humans, but rather on an interesting aspect of the whole thesis. Milovec (2014) also used alternating series for her research. We will discuss this later on more thoroughly as we will need those as a starting point of our implementation and research question as stated in chapter 1.

In general Milovec (2014) a crucial part of the thesis is her choice of the representation of the number series. Since the number series values must be transferred in input/output equations IGOR2 can induce. Number series can be represented in three ways:

- Representation Type 1: input is a list, output is a successor
- Representation Type 2: input is a position, output is a list
- Representation Type 3: input is a position, output is a value

Milovec (2014) chose Type 1, as it is closest to the form of how the series appear in IQ tests. Representation Type 1 has as a result a function that takes a list of natural numbers and returns the successor of the series. So at first the input/output equations for the natural numbers are defined. This makes it possible for IGOR2 to apply pattern matching to the numbers and so it can access every position in the series. In some cases it is not necessary for IGOR2 to make the recursive calls from the above described workflow, because it found the function for the number series with sheer pattern matching, as it can be seen on a series that is constructed by adding the constant “2” to every successor. The Background knowledge she used for enhancing IGOR2s performance is helping to detect relations between certain numbers. In most cases for each operator that is used in a series there are at least two examples for IGOR2 to learn from. E.g. when receiving an integer with value two and an integer with value four the result is six. This should suggest to the program that an addition of the two inputs takes place.

Out of the 20 number series used for her comparison study between humans and IGOR2 there were four alternating number series. Since the comparison is not in the scope of this work it will not be discussed further and the following refers only on the performance of IGOR2. All number series had the length of five, meaning IGOR2 was given five numbers to compute a function, and a limit of 1 hour 25 minutes to terminate. The four alternating series (NS3, NS8, NS13, NS18, the concrete numbers and functions are given and discussed in chapter 4) terminated in her study, but she states that IGOR2 gave for none of them the correct successor or the function the series was constructed with. NS8 was a special case, because she had to reduce the originally used constants in size in order to enable IGOR2 to terminate with this series.

Over all number series her study also showed that large numbers were a challenge for IGOR2 in whole, because each number is represented by successors starting from zero. Therefore, the number value “three” is known to IGOR2 as the “s s s 0” i.e. the successor of the successor of the successor of zero. This

shows how resource intensive the number value e.g “128” can be for IGOR2 since it always starts to calculate from zero. Milovec (2014) further study confirms that bigger number values influence strongly the performance of IGOR2, i.e. it needs longer/more iterations to terminate. The value of the first number of a series and the constant used in a function have, therefore, a relatively big impact on how fast and if IGOR2 can terminate and compute a result. The complexity of the operator on the other side does not effect IGOR2 so much concerning the iteration number.

She concludes that IGOR2 needed more iterations compared to the time humans needed for the series, but that in contrast to the human participants, whose solving time include correct and incorrect answers, IGOR2 only gave correct solutions. But the alternating series, as well as two of the Fibonacci kind, were excluded from her performance comparison, because IGOR2 gave no correct solutions for them.

In her conclusion she suggests that implementing an “if-construct” and so expanding the background knowledge to distinguish between “odd” and “even” could enable IGOR2 to solve also alternating series.

Chapter 4

Alternating Number Series with IGOR2

As we already explained in chapter two Hofmann (2012) and Milovec (2014) have shown that the IGOR2 algorithm has difficulties with solving alternating number series. Alternating number series are an established part of IQ test and show that test persons have way less difficulties solving them than IGOR2 has.

We suppose that there are two reasons for this. An alternating number series consists basically of at least two separate intertwined series. For each alternating series IGOR2 needs to generate two hypotheses. Due to IGOR2's inductive approach, at least three examples are necessary for a hypothesis construction for problems where the regularity can be observed from directly succeeding examples. Thus for a successful induction with an alternating series the program would need at least two times three examples if the series consists of just two series. With only five examples it is hence not possible for IGOR2 to test the hypothesis for the second series, because it has too few examples. This results in three examples for "the first" and only two for "the second" subsequence, but at least three examples each are needed for a successful induction. Accordingly the tested sequences used by Milovec (2014) are at least one example too short.

To show if IGOR2 has the ability to solve alternating number series, we tested the examples from Milovec (2014) that IGOR2 could not solve, with more examples. The used series are based on results of a former project group Grünwald et al. (2012). They generated 20 number series with different attributes like linear operations with one constant, alternating and Fibonacci series. They also increased the complexity of these attributes in four steps varying the value of the examples and arithmetic operations. Since we only worked with IGOR2 we use IGOR here as synonym.

We will describe our tested series like in Hofmann (2012): A number series is defined as $f(n) = x$ with domain of definition $\mathbb{N}$ and co-domain $\mathbb{R}$. $n \in \mathbb{N}$ with $n \geq 1$ defines the position within the series. Two different equations are separated by a semicolon.

As we are only interested in the alternating series in this project, we will only take the following alternating series tested by Milovec (2014) into account:

Alternating Series

- NS3: 5,7,10,12,15 $\rightarrow f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)+3 & \text{else} \end{cases}$
- NS8: 128,254,381,507,634 $\rightarrow f(n) = \begin{cases} f(n-1)+126 & n \text{ even} \\ f(n-1)+127 & \text{else} \end{cases}$
- NS13: 4,6,12,14,28 $\rightarrow f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)*2 & \text{else} \end{cases}$
- NS18: 16,48,51,153,156 $\rightarrow f(n) = \begin{cases} f(n-1)*3 & n \text{ even} \\ f(n-1)+3 & \text{else} \end{cases}$

Arithmetic operations in each series effect only the direct predecessor and not e.g. the predecessor of the predecessor, so the complexity is reduced. We also increased the time for a time out. In Milovec (2014) the time for the calculation was restricted to 1h and 25 min. Due to the increase of examples in the series, we increased the time limit up to 2 hours. We generated output files for each series and for each tested length. The resulting equations were then evaluated. The Maude modules containing the input and our output files containing the results of IGOR2 for the used number series, can be found in the appendix. To simplify the usage of IGOR2 in Maude for further research, a short manual and other descriptions of our working setup can be found there as well.

Our working environment was the PC-workstation available in the computer laboratory from the Cognitive Systems department (University of Bamberg):

- Operation System: Linux Mint 14 (nadia),
- Processors: Intel(R) Core(TM) i5-3570, CPU @ 3.40GHz: 1600,00MHz,
- Maude 2.6 (Copyright 1997-2010 SRI International).

4.1 Implementation and Approach

As we explained in chapter 2, a possible solution for IGOR2 to solve alternating number series could be more examples. To perform an induction, IGOR2 needs at least three equations of which the first is used as a base case and the other two as recursive functions. Furthermore, the program needs at least two examples to gerneralize the function of the hypotheses it computed (Milovec, 2014). We wanted to show whether more examples in a series would lead to a successful solution, and if so, how many examples would be required. Therefore, we extended the four introduced series by adding more examples in two steps. First we added just one additional example, resulting in a series with a total of six examples. In the second step we added three examples, resulting in a series with a total of eight examples. Since the program had difficulties with terminating series with eight examples or they delivered the same results as with six, we did not test the series with ten examples.

Number series

Length:	5	6	8
NS3	5, 7, 10, 12, 15	17	20, 22
NS8	128, 254, 381, 507, 634	760	887, 1013
NS13	4, 6, 12, 14, 28	30	60, 62
NS18	16, 48, 51, 153, 156	468	471, 1413
NS_AlterNeu	1, 2, 4, 5, 10	11	22, 23

NS_AlterNeu: $f(n) = \begin{cases} f(n-1)+1 & n \text{ even} \\ f(n-1)*2 & \text{else} \end{cases}$

We also tested a new series that was not part of Grünwald et al. (2012) and Milovec (2014) work: NS_AlterNeu: 1, 2, 4, 5, 10, 11, 22, 23.
This series was introduced to verify if IGOR2 can solve a "real" alternating series with small values and slowly growing values of examples. A definition of "real" or unambiguous alternating series is given in the subsection 4.2.1.

4.2 Results for each series

In the following each series is discussed separately, and information about the functions it is constructed of, is given. Furthermore, we provide information on how IGOR2 performed and the results with its interpretation. All series were tested with varying lengths, but at least with the total length of 6 and 8 examples. A few series were also tested with other lengths due to verify previous results, but the reasoning is given in the according paragraph.

IGOR2 generates several hypotheses how to continue the series. Because it does not show a concrete number, the hypotheses must be manually tested. Due to the fact that we have not enough knowledge in Maude, we were not able to interpret the results on our own. Therefore, we were supported by Prof. Schmid.

4.2.1 NS3

The first series we tested was NS3. The estimated difficulty by test participants of this series was defined by Grünwald et al. (2012) as 1 out of 5. In their study Grünwald et al. (2012) assume, the lower the difficulty rating is, the lower is the error percentage of the test participants. We can also define this series as "easy" to solve for humans.

NS3: 5, 7, 10, 12, 15

- added examples: 17, 20, 22
- constructing function: $f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)+3 & \text{else} \end{cases}$

Performance of IGOR2. Generated output files for 6 and 8 examples. By means of NS3 we want to shortly introduce the working method of IGOR2. After

starting IGOR2 in Maude the command for module generalization is needed: "reduce in IGOR : gen('NS3, 'Add3, 'add) ."
The reduction in the IGOR2 module is accomplished by the reduce call, see also Milovec (2014). Thus the defined gen function and the transferred parameters can be called. The parameters in our example are:

- the module which has to be analysed (NS3),
- the contained function (Add3),
- and the used background knowledge (add).

IGOR2 Input

Since datatypes are inapplicable in this program, integers are represented by recursive constructors:

op 0 : → MyNat [ctor] .

op s : MyNat → MyNat [ctor] .

IGOR2 gets its input as follows:

eq Add1((s s s s s 0) nil) = s s s s s s s 0.

eq Add1((s s s s s s s 0)(s s s s s 0) nil) = s s s s s s s s s s 0 .

eq Add1((s s s s s s s s s s 0)(s s s s s 0)(s s s 0) nil) = s s s s s s s s s s s s 0 .

...

The number series is not represented as integers, but as successors (s) of the natural number"0". This means "s s s s s 0" is equal to the integer "5" and so on. The whole series is transferred step by step. Each example has to be added separately. Located on the left side of the example we can find the number series, that was created so far and on the right side of the equal sign we can find the next example that is added. This representation enables the program to use pattern matching and generate equations to continue the series.

Furthermore, IGOR2 receives background knowledge. It has no own knowledge about arithmetic operations, so additional information has to be provided via the section background knowledge which contains similar example equations as shown above for the function: Receiving two "numbers" on the left side and the result on the right side, IGOR2 uses pattern matching to learn how to "count".

After a successful induction IGOR2 shows its results as follows:

```
rewrites:  281634 in 104ms cpu(104ms real)(2707863 rewrites/second).
```

The detailed output for NS3 containing the above information and the found hypotheses can be explored in the Appendix.

As we can see IGOR2 needed 281634 rewrites and 104 ms for the induction. In the second line we can find a list of all hypotheses induced by IGOR2. A

hypothesis (Hypo) is structured in this way: hypo(bool,nat,EQ). The boolean (bool) shows whether a correct hypothesis was induced. The program terminates only if it is able to find a correct hypothesis, so the boolean will always turn to "true". The second parameter shows a natural number of example partitions within a hypothesis. Last but not least, the third parameter (EQ) states the equation of the detected function. In default of a interpreter, we had to check the hypotheses manually.

Results and interpretation. Both tests terminated and provided following results:

Output NS3

- for 6 examples:

  reduce in IGOR : gen('NS3, 'Add3, 'add) .

  rewrites: 281634 in 104ms cpu (104ms real) (2707863 rewrites/second)

- for 8 examples:

  reduce in IGOR : gen('NS3, 'Add3, 'add) .

  rewrites: 461469 in 160ms cpu (160ms real) (2884000 rewrites/second)

The output for 6 total examples is similar to the output for 8 examples and provides the same hypotheses as in Milovec (2014). To validate our results, we tested on a random basis the first and the last hypotheses. Since both hypotheses delivered a correct solution, we assume that the others provide a valid solution as well. The first results led to an astonishment: the program could solve this alternating series correctly without providing an alternating hypothesis. Instead of performing the operation "+ 2" on the first and "+ 3" on the second example, IGOR2 merges these operations and performs "+ 5" on the predecessor of the current element. IGOR2 generated a different, but still correct hypothesis $f(n) = f(n-2)+5$. Hence we can see that IGOR2 did not find the proposed construction function, but a valid, we want it also call correct, function for calculating arbitrary requested successors.

We want to state that the proposed and learned functions are semantically equivalent, since they provide the same result for any possible input. Due to the fact that Milovec (2014) received the same results, we assume that there was a misinterpretation of her results. IGOR2 can already solve alternating series with only five examples by merging operators and constants.

As we will exemplify in the following, we can distinguish between four different kinds of alternating number series:

- series only solvable with an alternating function,
- series with a proposed alternating function but correctly solvable with a non-alternating function, in terms of that an arbitrary number of successors can be predicted (see also NS3),

- series correctly solvable in terms of that only the direct successor can be predicted but nothing further than that (see also NS18 and NS AlterNeu), as well as
- series that were not solvable.

Alternating series that can be solved with an non-alternating function are called in the following "ambiguous". These are all alternating series using only one operator, or a combination of contrary operators like addition and subtraction, or multiplication and division. We also want to call all other alternating series "real" or unambiguous. Series that did not terminate within the given time limit are called "not solvable" or "unsolable".

4.2.2 NS8

The second tested series was NS8. According to Grünwald et al. (2012) the test participants estimated the difficulty of this series as 3 out of 5.

NS8: 128, 254, 381, 507, 634

- added examples: 760, 887, 1013
- constructing function: $f(n) = \begin{cases} f(n-1)+126 & n \text{ even} \\ f(n-1)+127 & \text{else} \end{cases}$

Performance of IGOR2. Generated output files for 6 and 8 examples:

This series with large constants did not terminate several times. Since the used operators are the same as in NS3, we assumed that the big values were to blame for not terminating. So we reduced the values in several steps at about a half of the original value and let IGOR2 try to solve this series again. The output files referred to where we obtained by using smaller constants and just 5 examples.

NS8_1: 128, 191, 255, 318, 382

- added examples: no further examples needed
- constructing function: $f(n) = \begin{cases} f(n-1)+63 & n \text{ even} \\ f(n-1)+64 & \text{else} \end{cases}$

Performance of IGOR2.

Output NS8_1

for 5 examples and smaller values:

reduce in IGOR : gen('NS8_1, 'Add8_1, 'add) .

rewrites: 1134526435 in 2209734ms cpu (2209596ms real) (513422 rewrites/second)

Results and interpretation. In the first run IGOR2 was not able to terminate the given series, neither with six nor eight examples. As stated at the beginning of this chapter, we expected the series not to terminate, because of to few examples, but Milovec (2014) showed in her thesis that the series NS8 had already problems only to terminate. Her speculation was that the values chosen for the constants were too big for the systems capacity hence she reduced them to 63 and 64. This series then terminated, but according to her interpretation with a false result. Therefore, we also tested this series named NS8_1. The interpretation of the results showed that our tests with 6 and 8 examples would not have been necessary, because IGOR2 could find a simpler function to solve this series. NS8 and also NS8_1 are of the same type as NS3, because the two constants are both combined with the addition operator. Thus IGOR merges these values and performs only one operation: $f(n-2)+127$ Like in the case of NS3, we assume that there was a misinterpretation of Milovec (2014) results.

4.2.3 NS13

The third tested series was NS13. According to Grünwald et al. (2012) the test participants estimated the difficulty of this series as 2 out of 5.

NS13: 4, 6, 12, 14, 28

- added examples: 30, 60, 62
- constructing function: $f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)*2 & \text{else} \end{cases}$

Performance of IGOR2. Generated output files for 6, 7 and 8 examples. There was only one output for the series with 6 examples. The tested series with more examples did not terminate.

Output NS13

for 6 examples:

reduce in IGOR : gen('NS13, 'Alter1, 'add * 'mal) .

rewrites: 56454858 in 35082ms cpu (35079ms real)
(1609216 rewrites/second)

Results and interpretation. The resulting hypotheses were not all interpreted. IGOR can reproduce the series with taking the first examples as granted. As we can state this series is a unambiguous alternating series, due to its used operators. Furthermore, the values of the series are increasing moderately. Based on IGORs found hypotheses we examined that it constructed this function: $f(n) = f(n-2)+16$ (for $n \geq 4$)

The program tries to find a relationship between the last examples and is able to find the right operator. But unfortunately it does not find the right equation and so it is not able to continue the series further with valid solutions. The seventh example in this series has to be "60" but the provided solution

does not come to this conclusion. The proposed and learned functions are not semantically equivalent, since they do not provide the same result for any possible input, but only for the next successor. Here the sixth number can be reproduced correctly.

This leads us to the question whether the solution is coincidentally right. We do not think so. Here we have to distinguish between two functions of number series problems. If we expect the test participants to continue the series with two or more numbers the expected equation has to be determined. In this case IGOR2 could not solve NS13 correctly. Then again in IQ tests the task participants have to fulfill is to continue a series with one single number. Finding the expected equation is not a part of the evaluation. So it does not matter how someone continues the series as long as the solution is right. If we define number series problem as completing the series with one additional number, IGOR2 can solve NS13 correctly.

4.2.4 NS18

The fourth tested series was NS18. According to Grünwald et al. (2012) the test participants estimated the difficulty of this series as 2 out of 5.

NS18: 16, 48, 51, 153, 156

- added examples: 468, 471, 1413
- constructing function: $f(n) = \begin{cases} f(n-1) * 3 & n \text{ even} \\ f(n-1) + 3 & \text{else} \end{cases}$

Performance of IGOR2. Generated output files for 6 and 8 examples Did not terminate, therefore, there are no performance details from IGOR as the program only delivers output after terminating.

Results and interpretation. Since the program did not terminate we cannot provide any results. This number series is referred to as a "real" alternating series. Furthermore, the values are fast increasing. Even after a runtime of more than twelve hours there was no termination. We suppose the fast increasing values may cause this long runtime. We assume that IGOR has no difficulties with computing large numbers but with fast increasing differences between them. To test IGOR's abilities to solve unambiguous alternating number series we added another number series: NS_AlterNeu

4.2.5 NS_AlterNeu

The fifth and last tested series was NS_AlterNeu. Since this series was not part of the tests by Grünwald et al. (2012) and Milovec (2014) we cannot provide any difficulty estimation for this series.

NS_AlterNeu: 1, 2, 4, 5, 10

- added examples: 11, 22, 23
- constructing function: $f(n) = f(n-1) + 1; f(n-1) * 2$

Performance of IGOR2.

Output NS_AlterNeu

- for 6 examples:

  reduce in IGOR : gen('NS_AlterNeu, 'Alter1, 'add * 'mal) .

  rewrites: 1922723 in 1072ms cpu (1072ms real) (1793472 rewrites/second)

- for 8 examples:

  reduce in IGOR : gen('NS_AlterNeu, 'Alter1, 'add * 'mal) .

  rewrites: 9581439932 in 6245546ms cpu (6245156ms real) (1534123 rewrites/second)

Results and interpretation. Taking IGORs hypotheses we found that it constructs the function $f(n) = f(n-2)+6$ (for $n \geq 4$) given six examples and $f(n) = (n-2)+12$ (for $n \geq 6$) given eight examples. The examined hypotheses also showed that IGOR just constructed a function for the last number and taking all examples before as a fixed beginning. Therefore, we can only say that it constructs this function for an n greater than the total number of input examples. Due to the amount the output hypotheses we only examined a few and, therefore, it could be that IGOR found a function able to calculate infinite correct successor of the series.

This series is a "real" alternating series, i.e. different operators cannot be merged. It has also small constants and the values are slightly increasing. Difficulties with fast growing values like in NS18 or with to large constants like in NS8 are avoided. The first improvement is that the program terminates. Furthermore, it provides the correct answer for the next number for several hypotheses. Nevertheless, it does not figure out the right equation to continue the series beyond the first missing number. Like we elucidated in the section about NS13, we have to decide which requirements solving number series has. If predicting a single next number in the series is enough, we can state IGOR2 was able to solve the series.

4.3 Interpretation over all results

Over all results it gets clear that we can distinguish between ambiguous and unambiguous alternating number series. An ambiguous series can also be interpreted as a non alternating series e.g. by merging the first constant from the first-inner series with the one from the second. We observed this behaviour with series constructed by only one operator but with different applied values. As we had only additive constants as test examples with IGOR, we had to extrapolate from the observed system abilities for multiplicative operators. Therefore, we constructed some series for ourselves with two constants and the multiplication operator. Division and subtraction was skipped, because on the one hand a division can be expressed as a multiplication and on the other hand both were not part of the number series test in Grünwald et al. (2012) and Milovec (2014).

After our calculations on paper we conclude that same operators like "+x, +y" and "*x, *y", are mergable for IGOR, the multiplication due to the prime factorisation. To collect evidence pro or contra the ability of the system to solve "real" i.e. unambiguous alternating number series we introduced a new number series with two different operators and small start and constant values. The small values should ensure that the system did not get stuck and use all its capacities for the integral part of finding a correct function.

Even though we probed several hypotheses IGOR produced given six and eight examples, we have to conclude that IGOR is just able to predict the next number in the series, but nothing further than that. It provides not a valid function to calculate an arbitrary number of successors. "Real" alternating series contain arithmetic operators like multiplication and division. Alternating series with contrary operators can be easily summarized to one operator.

Ambiguous Series

- Alter1: 7,10,9,12,11 $\rightarrow \begin{cases} f(n-1)+3 & n \text{ even} \\ f(n-1)-1 & \text{else} \end{cases}$

  is equal to: $f(n-2)+2$

- Alter2: 1,2,6,12,36 $\rightarrow \begin{cases} f(n-1)*2 & n \text{ even} \\ f(n-1)*3 & \text{else} \end{cases}$

  is equal to: $f(n-2)*6$

We were able to show that series NS3 and NS8 with smaller constants were already correctly solved by IGOR2 due to the circumstance that IGOR2 used another equation instead of applying alternating ways to calculate the next number. Hence it seems that Milovec (2014) had some interpretation problems with the results of alternating series. This leads to the conclusion that IGOR2 could already solve alternating series with only five examples, as long as different equations can be summarized to one. "Real" alternating series, with multiplication or division, do, still, lead to difficulties like in NS13 or NS18.

But there are also other problems. The interpretation of the results is difficult and leads to mistakes. To improve the work with IGOR2 it would be necessary to write an interpreter which provides a concrete number or a concrete equation for a series, not a Maude file. Furthermore, the submission of a number series is difficult since every number has to be represented as a sequence of successors. This can also lead to mistakes, especially with larger numbers. IGOR2 was able to predict the right number without determining the right equation. In some of this cases, it used the last equation which was used to create the last number given in the series. The predecessors of the last given number were taken for granted and the relationship between the last number and its predecessor was used to continue the series. In some cases the predicted successor was correct. But in some cases, like NS_AlterNeu (NS_{AN}), the equation did not hold for the successor of the successor.

This leads to the question which claim we lay on solving number series. If on one hand the problem definition is to continue a series with just one more example, IGOR2 can already solve some alternating series correctly, without

finding the expected equations. On the other hand if it focuses on finding the right equations there is still room for improvement. In addition we also want to state that our tested series are not applicable for number series in IQ test due to their complexity.

The following table summarizes this chapter. We put together the information for each series: the function it was created by, the function IGOR found, was it possible to find the asked next number of the series with IGORs function, was it possible to find all numbers of the series with IGORs function.

Summary

	Construction Function	IGOR	next/all
NS3	$f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)+3 & \text{else} \end{cases}$	$f(n-2)+5$	yes/yes
NS8	$f(n) = \begin{cases} f(n-1)+126 & n \text{ even} \\ f(n-1)+127 & \text{else} \end{cases}$	—	— / —
NS8_1	$f(n) = \begin{cases} f(n-1)+63 & n \text{ even} \\ f(n-1)+64 & \text{else} \end{cases}$	$f(n-2)+127$	yes/yes
NS13	$f(n) = \begin{cases} f(n-1)+2 & n \text{ even} \\ f(n-1)*2 & \text{else} \end{cases}$	$f(n-2)+16$ (for $n \geq 4$)	yes/no
NS18	$f(n) = \begin{cases} f(n-1)*3 & n \text{ even} \\ f(n-1)+3 & \text{else} \end{cases}$	—	— / —
NS_{AN}	$f(n) = \begin{cases} f(n-1)+1 & n \text{ even} \\ f(n-1)*2 & \text{else} \end{cases}$	$f(n-2)+6$ (for $n \geq 4$) $f(n-2)+12$ (for $n \geq 6$)	yes/no

Chapter 5

Conclusion

The starting point for this project was the work of Milovec (2014) and Hofmann (2012). They both mentioned that IGOR2 did not perform well when given alternating number series and especially Milovec (2014) stated that IGOR2 did not return the correct function and thus also not the desired successor number for all of her four series. Our main expectation was that by providing IGOR2 with more input examples, the system would be able to solve also alternating series. Surprisingly our project revealed that some of the alternating number series used by Milovec (2014) were already solvable by IGOR2 with only five examples. Although they were not interpreted as alternating by IGOR2, the system was able to create another hypothesis that lead to the correct answer. Alternating series with one operator or a combination of contrary operators, can be easily summarized to one operator. In Hofmann (2012) IGOR2 could solve two alternating series. The suggested solution for the series NS3: 5, 7, 10, 12, 15 was $f(n-1)+2; f(n-1)+3$; we stated that IGOR2 did not solve this sequence with an alternating, but with a non-alternating hypothesis: $f(n-2)+5$.

"Real" alternating series contain arithmetic operators like "*" and "/" which cannot be merged by the program. Furthermore, some values were too large for IGOR2 to calculate a result for a series in a time limit of two hours. Apart from this technical issue the high values are fatuous, due to the circumstance that IQ tests use series with small values. Due to this we tested one of our series (NS8) which did not terminate within the time limit, with smaller alternating constants and thus got the right solution using only five examples for this series. In addition to the too high number values the series developed by Grünwald et al. (2012) and tested by Milovec (2014) are not applicable to IQ tests for another reason as they use too complex functions for humans. As a next step the focus of a following project could be to inspect more in detail how to enable IGOR to solve unambiguous number series. Interesting further questions could, therefore, be how the background knowledge could be improved. Particularly with regard to adding knowledge about odd and even length of series to IGOR.

Since the submission of parameters as well as the validation of results is not computable, yet, mistakes can happen. So another useful further task would be to write a parser for the received output. Maude represents the results as recursive equations in a correct Maude-module-syntax. Currently these equations have to be evaluated by hand. A parser could simplify working with IGOR and Maude. This parser could take the equations and deliver a concrete number as

result. This would also diminish the possibility of interpretation errors as they occur when validating results manually. Also not just a few of IGOR's output equations could be considered, but all of them. As for now we had to decide which hypotheses we evaluated, because there were too many for us to interpret them all manually in the scope of this project.

Summing up it can be said that it was an interesting and valuable project for us. We learned a lot about various aspects of number series. We showed that IGOR can handle alternating number series when they can be simplified, but it can not yet solve unambiguous ones. The project exposed a few new questions and aspects for further projects.

Acknowledgement

Finally, we want to take this opportunity to express gratitude to the faculty members of the Cognitive Systems Department of the University of Bamberg for their help and support. Especially, we are thankful to Prof. Dr. Schmid who offered her systematic guidance and encouragement throughout the course of this project with IGOR2 and for the great effort she put into guiding us through the scientific field. Thank you very much!

Bibliography

Burghardt, J. (2005). E-generalization using grammars. *Artificial intelligence*, 165(1):1–35.

Düsel, M., Werner, A., and Zeißner, T. (2012). Solving number series with the magichaskeller. *Course-report (KogSys-KogMod-M), Otto-Friedrich-Universität Bamberg*.

Grünwald, R., Heidel, E., Strätz, A., Sünkel, M., and Terbach, R. (2012). Induction on number series - project report. Technical report, Cognitive Systems Research University of Bamberg. "`http://www.cogsys.wiai.uni-bamberg.de/teaching/ss12/ba_pj/project_report.pdf`".

Hofmann, J. (2012). Automatische Induktion über Zahlenreihen - eine Fallstudie zur Analyse des induktiven Programmiersystems IGOR2. Bachelor thesis, University of Bamberg, Germany.

Hofmann, M., Kitzelmann, E., and Schmid, U. (2008a). Analysis and evaluation of inductive programming systems in a higher-order framework. In *KI 2008: Advances in Artificial Intelligence*, pages 78–86. Springer.

Hofmann, M., Kitzelmann, E., and Schmid, U. (2008b). A unifying framework for analysis and evaluation of inductive programming systems. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 55–60.

Holzman, T. G., Pellegrino, J. W., and Glaser, R. (1983). Cognitive variables in series completion. *Journal of Educational Psychology*, 75(4):603.

Katayama, S. (2005). Systematic search for lambda expressions. *Trends in functional programming*, 6:111–126.

Katayama, S. (2011). Magichaskeller: System demonstration. In *Proceedings of AAIP 2011 4th International Workshop on Approaches and Applications of Inductive Programming*, page 63. Citeseer.

Kitzelmann, E. (2007). Data-driven induction of recursive functions from input/output-examples. In *Proceedings of the ECML/PKDD 2007 Workshop on Approaches and Applications of Inductive Programming (AAIP'07)*, pages 15–26. Citeseer.

Kitzelmann, E. (2010). *A combined Analytical and Search-Based Approach to the Inductive Synthesis of Functional Programs.* PhD thesis, University of Bamberg, Germany.

Kitzelmann, E. and Hofmann, M. (2008). IGOR2–an inductive functional programming prototype. In *Proceedings of the System Demonstrations of the 18th European Conference on Artificial Intelligence*, pages 29–31.

Kotovsky, K. and Simon, H. A. (1973). Empirical tests of a theory of human acquisition of concepts for sequential patterns. *Cognitive Psychology*, 4(3):399–424.

Milovec, M. (2014). Applying inductive programming to solving number series problems – comparing performance of igor with humans. master thesis, University of Bamberg, Germany.

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.

Sanghi, P. and Dowe, D. L. (2003). A computer program capable of passing iq tests.

Strannegård, Claes; Amirghasemi, M. . U. S. (2013). An anthropomorphic method for number sequence problems. *Cognitive Systems Research*, 22–23:27–34.

Appendix A

Appendix

A.1 Generating the output files

Workflow in short.

- Start terminal
- start Maude and start to pipe the output of the program to a file: ‘‘`maude | tee fooBar.txt`’’
- load IGOR: ‘‘`load igor2.2.maude`’’ (loads all modules specified there)
- choose one number series module to work with: ‘‘`reduce in IGOR : gen('moduleName, 'functionName, 'backgroundKnowledge) .`’’
- e.g.: ‘‘`reduce in IGOR : gen('NS1, 'Add1, 'add) .`’’
- close Maude and also output stream: ‘‘`quit`’’
- open output file for further inspection and interpretation

More in detail. We chose to pipe all the output Maude generates into separate flies for each number series. For each series there exist more than one output file, because we made separate ones for each tested length. Since we execute Maude in terminal we used ‘‘`maude | tee fooBar.txt`’’ to generate our different files. After starting Maude with this we executed ‘‘`load igor2.2.maude`’’ to load all IGOR modules. Such a module are the number series. One module for each series. Only modules specified in this file will be available for Maude. In a next step Maude needs to know which modules and hence functions it needs to work with i.e. to reduce with IGOR. The command ‘‘`reduce in IGOR : gen('moduleName, 'functionName, 'backgroundKnowledge) .`’’ The module name is the name of the file in which the series is defined. The function is the name of the function specified in the before mentioned module. The background knowledge is also the name for it as specified in the module. E.g. for the number series 1 the series is defined in ‘‘`NS1.maude`’’, which defines the function ‘‘`Add1`’’ and the background knowledge ‘‘`add`’’: ‘‘`reduce in IGOR : gen('NS1, 'Add1, 'add) .`’’

Note that the `‘‘ .’’` at the end is mandatory. Maude then reduces this series and prints the resulting equations in the terminal. After each reduction we closed Maude with “quit” and with this also the output writing process. All equations are now in the specified text file.

A.2 Naming and integration of new modules

We made working copies from the Maude modules Milovec (2014) produced for her master thesis. The modules were extended by us with the example equations for 6 and 8 inputs. We have not two separate modules for each length because we just had to add some more equations and the focus of our work were the respective outputs. Hence we have produced separate output files for each tested series length. The naming scheme was: `‘‘NS*_proj-SeriesLength_output.txt’’` (e.g.: NS3_proj-6_output.txt) The original modules of Milovec (2014) were kept and renamed to `‘‘* (Kopie).maude’’`. As this was done at the beginning of our project and is not very self-explanatory we mention it here to avoid confusion for readers who might want to use our project data.

A.3 Exemplary shortened output files

Output file of NS3 (6 Examples, shortened to line 1-66)

```
1                       \||||||||||||||||||/
2                     --- Welcome to Maude ---
3                       /||||||||||||||||||\
4              Maude 2.6 built: Nov 18 2011 14:03:45
5              Copyright 1997-2010 SRI International
6                     Tue Mar 17 15:07:53 2015
7  reduce in IGOR : gen('NS3, 'Add3, 'add) .
8  rewrites: 281634 in 104ms cpu (104ms real) (2707863 rewrites
       /second)
9  result HypoList: hypo(true, 2, eq 'Add3['__['s_['s_['s_['s_
       ['s_['0.MyNat]]]]],
10     'nil.MyList]] = 's_['s_['s_['s_['s_['s_['s_['0.MyNat
           ]]]]]]] [none] .
11 eq 'Add3['__['s_['s_['s_['s_['s_['s_['s_['X0:MyNat]]]]]]],'
       __['s_['s_['s_['s_[
12     's_['X1:MyNat]]]]],'X2:MyList]]] = 's_['s_['s_['s_['s_['
           s_['s_['s_['s_['s_[
13     'X1:MyNat]]]]]]]]] [none] .)
14
15 nextHypo
16
17 hypo(true, 2, eq 'Add3['__['s_['s_['s_['s_['s_['X0:MyNat
       ]]]]],'X1:MyList]] =
18     's_['Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:
           MyList]]] [none] .
19 eq 'Sub1['__['s_['s_['s_['s_['s_['0.MyNat]]]]],'nil.MyList]]
       = 's_['s_['s_['s_[
20     's_['s_['0.MyNat]]]]]] [none] .
```

```
21 eq 'Sub1['__['s_['s_['s_['s_['s_['s_['s_['X0:MyNat]]]]]]],'
       __['s_['s_['s_['s_[
22       's_['X1:MyNat]]]]],'X2:MyList]]] = 's_['s_['s_['s_['s_['
           s_['s_['s_['s_[
23       'X1:MyNat]]]]]]]] [none] .)
24
25 nextHypo
26
27 hypo(true, 2, eq 'Add3['__['s_['s_['s_['s_['s_['X0:MyNat
       ]]]]],'X1:MyList]] =
28       's_['Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:
           MyList]]] [none] .
29 eq 'Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
        = 's_['Sub2['__[
30       's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
           .
31 eq 'Sub2['__['s_['s_['s_['s_['s_['0.MyNat]]]]],'nil.MyList]]
        = 's_['s_['s_['s_[
32       's_['0.MyNat]]]]] [none] .
33 eq 'Sub2['__['s_['s_['s_['s_['s_['s_['s_['X0:MyNat]]]]]]],'
       __['s_['s_['s_['s_[
34       's_['X1:MyNat]]]]],'X2:MyList]]] = 's_['s_['s_['s_['s_['
           s_['s_['s_[
35       'X1:MyNat]]]]]]]] [none] .)
36
37 nextHypo
38
39 hypo(true, 2, eq 'Add3['__['s_['s_['s_['s_['s_['X0:MyNat
       ]]]]],'X1:MyList]] =
40       's_['Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:
           MyList]]] [none] .
41 eq 'Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
        = 's_['Sub2['__[
42       's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
           .
43 eq 'Sub2['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
        = 's_['Sub3['__[
44       's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
           .
45 eq 'Sub3['__['s_['s_['s_['s_['s_['0.MyNat]]]]],'nil.MyList]]
        = 's_['s_['s_['s_[
46       '0.MyNat]]]] [none] .
47 eq 'Sub3['__['s_['s_['s_['s_['s_['s_['s_['X0:MyNat]]]]]]],'
       __['s_['s_['s_['s_[
48       's_['X1:MyNat]]]]],'X2:MyList]]] = 's_['s_['s_['s_['s_['
           s_['s_[
49       'X1:MyNat]]]]]]] [none] .)
50
51 nextHypo
52
53 hypo(true, 2, eq 'Add3['__['s_['s_['s_['s_['s_['X0:MyNat
       ]]]]],'X1:MyList]] =
54       's_['Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:
           MyList]]] [none] .
```

```
55 eq 'Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
       = 's_['Sub2['__[
56     's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
         .
57 eq 'Sub2['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
       = 's_['Sub3['__[
58     's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
         .
59 eq 'Sub3['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]
       = 's_['Sub4['__[
60     's_['s_['s_['s_['s_['X0:MyNat]]]]],'X1:MyList]]] [none]
         .
61 eq 'Sub4['__['s_['s_['s_['s_['s_['0.MyNat]]]]],'nil.MyList]]
       = 's_['s_['s_[
62     '0.MyNat]]] [none] .
63 eq 'Sub4['__['s_['s_['s_['s_['s_['s_['s_['X0:MyNat]]]]]]],'
     __['s_['s_['s_['s_[
64     's_['X1:MyNat]]]]],'X2:MyList]]] = 's_['s_['s_['s_['s_['
         s_['X1:MyNat]]]]]]
65     [none] .)
66
67 [More hypotheses till line: 124, shortened for the appendix;
       complete file is on the CD with all materials.]
68
69 Bye.
```

Output file of NS_AlterNeu (6 Examples, shortened to line 1-60)

```
1                      \|||||||||||||||||||/
2                    --- Welcome to Maude ---
3                      /|||||||||||||||||||\
4            Maude 2.6 built: Nov 18 2011 14:03:45
5            Copyright 1997-2010 SRI International
6                   Wed Mar 25 19:43:16 2015
7  reduce in IGOR : gen('NS_AlterNeu, 'Alter1, 'add * 'mal) .
8  rewrites: 1922723 in 1072ms cpu (1072ms real) (1793472
       rewrites/second)
9  result HypoList: hypo(true, 4, eq 'Alter1['__['s_['0.MyNat
       ],'nil.MyList]] =
10     's_['s_['0.MyNat]] [none] .
11 eq 'Alter1['__['s_['s_['0.MyNat]],'__['s_['0.MyNat],'nil.
       MyList]]] = 's_['s_[
12     's_['s_['0.MyNat]]]] [none] .
13 eq 'Alter1['__['s_['s_['s_['s_['0.MyNat]]]],'__['s_['s_['0.
       MyNat]],'__['s_[
14     '0.MyNat],'nil.MyList]]]] = 's_['s_['s_['s_['s_['0.MyNat
         ]]]]] [none] .
15 eq 'Alter1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'__['s_['
       s_['s_['s_[
16     'X1:MyNat]]]],'__['s_['s_['X2:MyNat]],'__['s_['X1:MyNat
         ],'X3:MyList]]]]] =
17     's_['s_['s_['s_['s_['s_['s_['s_['s_['s_['X1:MyNat
         ]]]]]]]]]] [none] .)
18
```

```
19 nextHypo
20
21 hypo(true, 4, eq 'Alter1['__['s_['0.MyNat],'nil.MyList]] = '
       s_['s_['0.MyNat]] [
22     none] .
23 eq 'Alter1['__['s_['s_['0.MyNat]],'__['s_['0.MyNat],'nil.
       MyList]]] = 's_['s_[
24     's_['s_['0.MyNat]]]] [none] .
25 eq 'Alter1['__['s_['s_['s_['s_['X0:MyNat]]]],'__['s_['s_['X1
       :MyNat]],'__['s_[
26     'X2:MyNat],'X3:MyList]]]] = 's_['Sub12['__['s_['s_['s_['
           s_['X0:MyNat]]]],
27     '__['s_['s_['X1:MyNat]],'__['s_['X2:MyNat],'X3:MyList
           ]]]]] [none] .
28 eq 'Sub12['__['s_['s_['s_['s_['0.MyNat]]]],'__['s_['s_['0.
       MyNat]],'__['s_[
29     '0.MyNat],'nil.MyList]]]] = 's_['s_['s_['s_['0.MyNat]]]]
           [none] .
30 eq 'Sub12['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'__['s_['s_
       ['s_['s_['s_[
31     'X1:MyNat]]]],'__['s_['s_['X2:MyNat]],'__['s_['X1:MyNat
           ],'X3:MyList]]]]] =
32     's_['s_['s_['s_['s_['s_['s_['s_['s_['s_['X1:MyNat]]]]]]]]] [
           none] .)
33
34 nextHypo
35
36 hypo(true, 4, eq 'Alter1['__['s_['0.MyNat],'nil.MyList]] = '
       s_['s_['0.MyNat]] [
37     none] .
38 eq 'Alter1['__['s_['s_['X0:MyNat]],'__['s_['X1:MyNat],'X2:
       MyList]]] = 's_[
39     'Sub3['__['s_['s_['X0:MyNat]],'__['s_['X1:MyNat],'X2:
           MyList]]]] [none] .
40 eq 'Sub3['__['s_['s_['0.MyNat]],'__['s_['0.MyNat],'nil.
       MyList]]] = 's_['s_['s_[
41     '0.MyNat]]] [none] .
42 eq 'Sub3['__['s_['s_['s_['s_['0.MyNat]]]],'__['s_['s_['0.
       MyNat]],'__['s_[
43     '0.MyNat],'nil.MyList]]]] = 's_['s_['s_['s_['0.MyNat]]]]
           [none] .
44 eq 'Sub3['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'__['s_['s_
       ['s_['s_['s_[
45     'X1:MyNat]]]],'__['s_['s_['X2:MyNat]],'__['s_['X1:MyNat
           ],'X3:MyList]]]]] =
46     's_['s_['s_['s_['s_['s_['s_['s_['s_['X1:MyNat]]]]]]]]] [
           none] .)
47
48 nextHypo
49
50 hypo(true, 4, eq 'Alter1['__['s_['X0:MyNat],'X1:MyList]] = '
       s_['Sub1['__['s_[
51     'X0:MyNat],'X1:MyList]]] [none] .
```

```
52 eq 'Sub1['__['s_['0.MyNat],'nil.MyList]] = 's_['0.MyNat] [
       none] .
53 eq 'Sub1['__['s_['s_['0.MyNat]],'__['s_['0.MyNat],'nil.
       MyList]]] = 's_['s_['s_[
54      '0.MyNat]]] [none] .
55 eq 'Sub1['__['s_['s_['s_['s_['0.MyNat]]]],'__['s_['s_['0.
       MyNat]],'__['s_[
56      '0.MyNat],'nil.MyList]]]] = 's_['s_['s_['s_['0.MyNat]]]]
            [none] .
57 eq 'Sub1['__['s_['s_['s_['s_['s_['X0:MyNat]]]]],'__['s_['s_
       ['s_['s_[
58      'X1:MyNat]]]],'__['s_['s_['X2:MyNat]],'__['s_['X1:MyNat
           ],'X3:MyList]]]]] =
59      's_['s_['s_['s_['s_['s_['s_['s_['s_['X1:MyNat]]]]]]]]] [
           none] .)
60
61
62 [More hypotheses till line: 864, shortened for the appendix;
       complete file is on the CD with all materials.]
63
64
65 Bye.
```