Otto-Friedrich-Universität Bamberg

Professur für Kognitive Systeme

Applying Inductive Programming to Solving Number Series Problems

- Comparing Performance of IGOR with Humans

Masterarbeit

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Verfasser: Milovec Martina

Gutachter: Prof. Dr. Ute Schmid

Abstract

Number series problems in IQ tests are challenging, especially for humans and computer programs. This thesis presents a comparison study between the IGOR algorithm and human performance on number series problems. IGOR's task was to find a correct function for number series problems with different complexity factors, in order to generate the number series from this function. IGOR's performance results bring closer the capabilities of a computer program to the human intelligence. The introduction of this thesis describes induction and the psychological background. Furthermore, different computer models for solving number series problems, such as MagicHaskeller, ANN, Semi-analytical model, Spaun, Asolver and Seqsolver, are given. A short introduction to the IGOR algorithm in Maude and representation of number series problems in IGOR is also given. The results first focus on the human performance and the preliminary study on number series problems with IGOR. Secondly, the comparison of both performances of IGOR and humans are represented and summarized. The conclusion gives an overview of the results, problems and some corresponding solution suggestions, as well as ideas for further research on number series problems with IGOR.

Contents

1 Introduction **1**

2 Solving Number Series Problems **3**

2.1 Induction and Number Series 3

2.1.1 Induction, Deduction and Abduction 3

2.1.2 Induction and number series 4

2.2 Psychological Background 5

2.2.1 Cognitive aspects 5

2.2.2 Intelligence and IQ Tests 10

2.3 Computer Models 14

2.3.1 Inductive Programming 14

2.3.2 Artificial Neural Networks 17

2.3.3 Semi-analytical model 21

2.3.4 Spaun Brain Model 24

2.3.5 Asolver and Seqsolver 27

3 Applying Inductive Programming to Number Series Problems **33**

3.1 An Introduction to IGOR 33

3.1.1 The IGOR algorithm 33

3.1.2 IGOR in Maude 37

3.2 Representing Number Series in IGOR 40

3.2.1 Data representation 40

4 Results **44**

4.1 A study with humans 44

4.1.1 Generating Number Series 44

4.1.2 Method and Results of the Human Study 47

4.2 Scope and solution times for IGOR 49

4.2.1 Performance and Material 49

4.2.2 Preliminary Study and Performance results 50

4.3 Comparison of Human and Computational Results 55

4.3.1 Performance of Human and IGOR 56

4.3.2 Performance of Human and IGOR by Groups 59

5 Conclusion **65**

A Appendix **71**

A.1 NS1 Module in Representation type input: list - output: successor 71

1 Introduction

Program synthesis has been a subject of research since 1960, especially automatic induction of recursive functional or logic programs from input/output-examples [Kit07]. In general, there are two approaches dealing with inductive program synthesis, generate-and-test and the analytical approach (see [Kit07]). In this Master thesis the focus is on the analytical approach in which programs generalise recursive functions from recurrences of examples, where hypotheses are not searched but computed [Kit07]. This kind of approach has many performance advantages, especially that it narrows a search field by making the approach faster [Kit07]. A system that implements the inductive functional programming and automatic induction is called IGOR[1]. Based on input/output-examples, IGOR generalises a function which proved to be very successful in solving number series problems.

This kind of approach can be applied not only in computer science but also in other research fields, e.g. in psychology. With the help of induction, psychology can show us how people are able to deal with number series problems. Solving number series problems is an exercise in many IQ tests, which measure the level of human intelligence. The task was to correctly predict the next number in a given number series at various complexity levels. Here, complexity levels were defined: (i) by various operational complexity, such as addition, subtraction, multiplication or division, (ii) by numerical complexity, such as numbers with low and high values, and (iii) by structural complexity, where number series were classified by the same generational structure [GHSST12]. Some of these tasks have been chosen in an empirical study by a group of students of University of Bamberg (see [GHSST12]) with a purpose of giving an insight into cognitive processes and reasoning.

Results of number series tests have been used in a performance comparison between the humans and the analytical induction system IGOR, and have been analysed in this Master thesis. The performance of IGOR was tested: (i) at various complexity levels, such as operational, numerical and structural complexity, and (ii) in a number of categories, such as time, correctness, number of iterations and its limits.

This Master thesis provides a performance comparison between IGOR and humans. If human level of intelligence can be measured by IQ tests, then in a certain way the "intelligence" of IGOR can be reflected in the score of

[1] IGOR-***I****nductive* ***G****eneralization* ***o****f* ***r****ecursive Functions*

number series tasks. This kind of approach gives a new way of investigating advantages and disadvantages of IGOR in order to improve its performance.

In the second chapter of this Master thesis the basics principles for solving number series problems are explained. The first section lays out the relevant terms and concepts for induction that are important for the further work in this Master thesis. The second section gives the psychological background for solving number series problems, the cognitive aspects and the importance of IQ tests. The third section explains different computer models for number series solving that have been developed and tested so far, such as MagicHaskeller, ANNs, Spaun Asolver and Seqsolver.

In the third chapter IGOR's working principles and its structure is given. Foundations presented in this chapter are of major importance for representing number series problems. The first section gives the algorithm of IGOR and in addition how is IGOR implemented in the Maude language. Data representation of number series used in this Master study on concrete examples are given in the second section.

The results of the comparison study of IGOR and humans are given in the fourth chapter. The first section shows a detailed overview of human study on number series with its results. The human study represents the foundations for the IGOR study. In addition, the second section gives the results of the IGOR performance on number series problems including the preliminary study. In the third section the results of both studies, human and IGOR, will be compared and discussed in details.
The last chapter summarizes the results of the study in order to identify possible problems and appropriate solutions.

2 Solving Number Series Problems

Induction of number series can be found in many research fields, e.g. in psychology where it is shown how humans deal with number series problems. Therefore, the most common field of appliance for induction of number series are many IQ tests. While humans are used to such number series tasks in IQ tests, these can be very challenging for various computer systems, e.g. IGOR.

In the first part of this chapter a small introduction into induction and number series will be given. The induction, in comparison of deduction and abduction, will be explained and how number series and induction are related. In the second part of this chapter the psychological background of human cognitive principles and their relationships to the number series problems will be detailed explained. Afterwards the importance of intelligence and IQ tests for human number series solving will be investigated. In the last part of this chapter an overview of different computer models and methods for solving number series problems, such as MagicHaskeller, Artificial Neural Networks, Spaun, Asolver and Seqsolver, is given.

2.1 Induction and Number Series

Induction as a method can be applied on a various problem specifications in psychology or artificial intelligence. Induction includes many processes that can expand knowledge [HHNT86]. Humans observe certain behaviour and make plausible choices or decisions based on their observations or experience. Such processes of thinking and decision making are central to humans reasoning and computer science. In the following section the psychological background of induction and induction on number series will be briefly explained.

2.1.1 Induction, Deduction and Abduction

Any intelligent system has various features, such as learning, adaptability, efficiency improving etc., but one of the most important features is inductive reasoning [HHNT86]. Reasoning involves judgement about a situation by considering the facts and to thinking in a logical way. Therefore, we can say that inductive reasoning is a sort of logical inference which enables humans to make "logical" decisions. In order to accurately explain the term *induction*

one should first understand its relationship to the deduction and abduction.

- **Induction** - is a method where general rules and principles are derived from facts and examples. [HHNT86] gave an simple example: Socrates is a man. Socrates is mortal. Therefore, all men are mortal. However, there is a possibility that derived conclusions are not always true, because of uncertainty of facts or examples, which are based mostly on experience.

- **Deduction** - does the opposite: logical consequence is derived from assumptions (e.g. from a general rule or statement and a single example): All men are mortal. Socrates is a man. Therefore, Socrates is mortal. [HHNT86] mentioned that these assumptions must be true for deduction to be valid.

- **Abduction** - is a method where a possible condition or conclusion is derived from a single general statement and a consequence: All men are mortal. Socrates is mortal. Therefore, Socrates is a man. The general statement "All men are mortal" is explained by the consequence "Socrates is mortal", which allows abduction to interefere.

According to the previous examples, inductive reasoning is based on initial observations which can lead to the discovery of certain patterns and theories. Such uncertain predictions about different things (based on observations and logic) is a method for decision making and generating general theories about certain things. In the following, induction will be observed in the context of number series.

2.1.2 Induction and number series

The central point of this Master thesis are number series, which are represented through functions for better understanding of the relations between the numbers. Number series can be defined mathematically as a function $f(n)$ containing the set of natural numbers $\mathbb{N}$ and the field of real numbers $\mathbb{R}$ [EKF+03]. The input values for functions are actually the positions of number series which are be represented as: $n \in \mathbb{N}$, where $n \geq 1$. In this Master thesis all number series are defined from the set of natural numbers

$\mathbb{N}$.
In mathematics, there are two ways how the relationships between numbers in a number series can be expressed:

- *explicitly* - the successor is calculated based on the position n or the function input and
- *recursively* - the successor is calculated based on its pre-successors.

Table 1 shows two representational types on examples on examples from [GHSST12].

Table 1: Function examples for explicit and recursive relationship representations between numbers in number series

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

Number series complexity can vary due to the operator complexity or other factors. In some cases number series can relate to other number series or functions, which is not the case in this Master thesis.

2.2 Psychological Background

Psychology is closely related to computer science, even more than people are actually aware. After all, computers and computer applications have been created by humans and constantly used for different purposes. Psychological concepts and principles are very important for computer science, especially for artificial intelligence. Therefore, psychology has been a significant part of computer science for many years and due to its interdisciplinarity, it is still being researched. This section explains the human cognitive principles and how they are important for solving number series problems.

2.2.1 Cognitive aspects

Different models for solving number series problems

The problem how people approach the number series problems and their way of reasoning during such tasks has been discussed since the 1970. Kotovsky and Simon [KS73] proposed a model for solving letter series completion test, which was a starting point for a theoretical number series model from Holzman et al. [HPG83]. [KS73] remodelled their information processing theory of human acquisition into a computer program which performs the task and runs tests with subjects. The task consisted of two parts [KS73]:

1. *a pattern generator* - takes for an input a patterned sequence of letters, summarized into a pattern description of the sequence, e.g. "concept" or "rule" ;

2. *a sequence generator* - uses as an input the pattern descriptions from the pattern generator and based on this generalizes the letter sequence.

In their study, Kotovsky and Simon [KS73] presented the results of subjects behaviour during the problem solving process, their thinking processes, the hypotheses about possible solutions, their methods for recognizing the "patterns" in sequences, etc, which were very important for the further work of Holzman et al. [HPG83]. They used Kotovsky and Simon's [KS73] proposition framework of letters series completion and applied it to the number series completion. Number series have much more relations that can be investigated than letter series, therefore Holzman et al. [HPG83] upgraded the framework and covered different types of inductive reasoning considering the number series. It takes four steps to solve number series problems as shown in the figure 1 [HPG83].

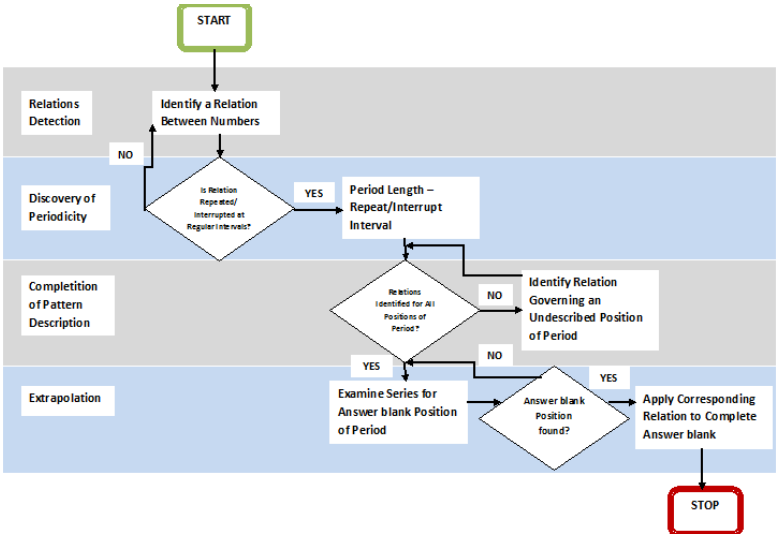

Figure 1: Flow diagram for number series completion

Holzman et al. [HPG83] described the process of solving number series as followed:

1. **Relation Detection** - in this phase subjects are required to examine the number series and build a hypothesis on how the numbers are interrelated. Compared to letters, number series have a wider variety of relations like arithmetic operations (e.g. addition, subtraction, division, multiplication, etc.) and the magnitude of the arithmetic operation (e.g. in the number series: "3, 5, 7, 9, 11" the magnitude of the addition operator is +2).

2. **Discovery of Periodicity** - here, subjects need to determine the period length of the number series. Period length is the number of elements which makes a pattern cycle complete, e.g number series (a) "5, 7, 10, 12, 15" has a period length of one, meaning the same relation is applied to every element of the series, but number series (b) "1, 1, 2, 2, 3, 3" has a period length of two, because a new relation cycle begins after two elements in the number series. There are two methods how to discover periodicity in number series: (i) "*adjacent*" method where the regular breaks in relations between adjacent element are discovered (e.g. in number series b the relation regularity breaks after exactly two elements), and (ii) "*nonadjacent*" method discovers regular intervals for

repeating relations (e.g. number series "45, 36, 44, 36, 43, 36" the relation is repeating for every two elements intervals which means that the period length is two).

3. **Completition of Pattern Description** - in third phase, subjects need to examine all other elements in the number series and see if previously defined relations can be applied to each element in the number series in order to define a rule for the whole number series. If a defined rule can not be applied to all relations within the series, then new relations need to be identified and rule must be completed. A correct defined rule is used to extrapolate the number series.

4. **Extrapolation** - the last phase of solving number series is to continue the series or to generate a next element (number) for the series. First the position of the next element should be identified, and then the relation of a previously defined rule needs to be applied so that the next element of number series could be generated.

Holzman et al. [HPG83] have extended the initial framework for letter series from Kotovsky and Simon [KS73] and shown that its use for number series has been very successful. This model describes in the first three phases how subjects build relations between elements in the number series and also their approach to detect certain patterns, which is important for rule definition. In the last phase, subjects need to apply the defined rule to the number series. This model describes the importance of cognitive processes for number series problem solving and provide good foundations for further work and use in computer science.

Cognitive Variables in Number Series Induction

Induction (e.g. over number series) and analogy[2] problems can be found in IQ tests representing a very important class of problem defining. Also, they can help answer question as to which criteria are crucial for solving complex or simple number sequence. In [HPG82] study a solution model in mentioned that can be applied to a numerical analogical problem in a form of "A:B :: C:D :: E:_ _" (see [HPG82] p. 361). This model happens to be crucial for defining cognitive variables and their impact on induction. Three phases of processing are described in the model as following [HPG82]:

[2]An Analogy is a comparison between two objects, or their systems, that tries to find in which aspects they are similar.

- **First Phase** - here the relation between the "A:B" pair must be identified and the result is an operation and a value. With this value it is possible from the A element of the series to produce the B element of the series.
- **Second Phase** - conclusion from the first phase is transferred into the second processing phase where it is applied to the encoded second pair "C:D". Here the relation from the first pair is compared with the relation of the second pair and if they are consistent, the processing is continued into the third phase.
- **Third Phase** - the "E" is encoded and the relation and value from the previous phase is simply applied to the E element in order to produce an "F" element.

Such simple processing model is not always the case, therefore [HPG82] emphasize how processes and content knowledge are very important for solving complex numerical analogical problems. The first significant variable for solving numerical problems is number of arithmetic operations (e.g. +,-,*,/). If within number series various types of operations occur, then the search for a consistent relation can take longer due to its complexity. The second variable is relational ambiguity, which is the "consequence" of various types operations occurrence in number series. Relational ambiguity occurs when two or more hypotheses for a possible solution exist, while subjects are obligated to investigate these hypotheses. Subject's content knowledge and its approach to the problem are important for dealing with ambiguity and finding a solution. Above mentioned factors from the [HPG83] study and its process model (e.g. period length and magnitude of the arithmetic operation) can also be categorised as cognitive variables for solving numerical problems. The impact on such variables to the number series problems performance was investigated in both studies: in [HPG82] and in [HPG83]. There were 54 subjects included in the study and divided into three groups: the first group consisted of intelligent school children (with IQ score over 130), in the second group were regular school children (with IQ score about 100) and in the third group were students.
The first study [HPG82] showed that the subjects solved first the simplest number series and that the relations with only one operator were successful applied to the number series. With this practice the subjects were capable to deal with complex number series and relations with different operators. The complexity of that kind of relations influenced the performance of all three groups. The regular school children were not that successful in choosing

the correct operator for complex number series as the students or intelligent school children. [HPG82] explained that these two groups are more familiar with complex operators (e.g. multiplication or division) because of their common use, and have, therefore, achieved better results.
The second study [HPG83] showed similar results. The performance of the subjects was influenced by the operator number and the period length in complex number series. The experienced subjects (second and third group) did not have much difficulties with content based knowledge and showed better results in solving complex number series.
In general, both studies showed that subjects that have more experience with mathematics had better results in solving complex number series. These two frameworks provided a clear picture of the cognitive approach to number series problems. Also, those studies showed that there are many other factors that influence the number series induction like subjects particular skills and their intelligence.

2.2.2 Intelligence and IQ Tests

Intelligence is the most common subject within psychology and psychologists have studied it hundred years. On the other hand, there is still no standard definition as to what intelligence actually defines. During the past some psychologists have defined intelligence as a general ability, while other defined it as set of skills, ability to apply knowledge or set of talents, etc. Therefore, psychologists have developed different IQ tests to measure intelligence and to cover various fields of knowledge.

The History of IQ Tests and Intelligence Theories

The first IQ test was developed by a French psychologist Alfred Binet, who was asked by his government to develop a test with a purpose of identifying students with learning problems in school. Together with his colleague Theodore Simon, they developed various tests for testing students mental skills. The starting point for their research was a number of questions with the focus on attention, memory and problem solving-skills of students [BS16]. Results showed that some younger students were able to answer the "difficult" questions that in general only older and experienced students were able to answer, while other older students could answer only the questions that were primarily intended for the younger students [BS16]. These questions and answers helped [BS16] to make a conclusion about the mental age of

students and build a model which is known as the Binet-Simon scale. This scale is used nowadays in many IQ tests.
British psychologist Charles Spearman introduced the term *general intelligence* known as *g factor*. Spearman ran many tests to measure human mental ability using the factor analysis technique in order to compare the performances. The test results showed that the scores were very similar, so Spearman concluded that the intelligence can be generalised over other tests due to the achieved test score. " The above and other analogous observed facts indicate that all branches of intellectual activity have in common one fundamental function (or group of functions), whereas the remaining or specific elements of the activity seem in every case to be wholly different from that in all the others [...]" [Spe04]. The score factor that is always the same in all tests is called *g factor*, while factor that varies across tests is called specific factor [Spe04]. Spearman's observations are fundamental for intelligence measurement.
When we think about IQ tests, then our first idea is the variety of problems and tasks that must be solved within a certain time. The task variety was very important for [Thu38] intelligence measurement, because his idea was to cover different problem areas in tests. Thurstone's theory is focused on seven different "primary mental abilities" [Thu38]:

1. Verbal comprehension
2. Reasoning
3. Perceptual speed
4. Numerical ability
5. Word fluency
6. Associative memory
7. Spatial visualization

With his new model of human intelligence, Thurstone challenged Spearman's view on intelligence as unitary conception and gave good foundations for future researches.
In 1983 in his book *Frames of Mind: The Theory of Multiple Intelligences*, Gardner has outlined a new theory of human intellectual skills called "*multiple intelligence*". He was of the opinion that previous theories are in a certain way limited and tried to expand them with an assumption that all people,

besides the intellectual capacity, have many different types of "intelligences" [Gar83]. He proposed eight "intelligences" based on skills and abilities of people and investigated them within different cultures [Gar83]:

1. Visual-spatial Intelligence
2. Verbal-linguistic Intelligence
3. Bodily-kinesthetic Intelligence
4. Logical-mathematical Intelligence
5. Interpersonal Intelligence
6. Musical Intelligence
7. Intra personal Intelligence
8. Naturalistic Intelligence

Sternberg presented two kinds of intelligence theories: the *explicit* theory based on collected data from IQ tests and the *implicit* theory which is mostly discovered by people and not created (such theories already are "present" in people's heads, people just need to recall them) [Ste85]. Despite the fact that Sternberg actually supported Gardner's " multiple intelligences", he considered that following factors are necessary for a "*successful intelligence*" [Ste85]:

- *Analytical Intelligence* - consists of three components: meta-components, performance components and knowledge acquisition components, which are crucial for problem-solving.
- *Creative Intelligence* - involves experience: are humans able to handle new situations and use their own experiences, but also current skills.
- *Practical Intelligence* - refers to the context of the situation: are humans able to understand and deal with a changing situation.

Sternberg's triarchic theory of intelligence represents a more cognitive approach to intelligence then any other theory before.
Many of above mentioned theories can be found in present IQ tests. The Intelligence Structure Test 2000 R (I-S-T 2000 R) [LBBA07] is based on the Thurstone's seven "primary mental abilities". Therefore, this tests provides

many verbal or numerical abilities tasks.
In psychometric IQ tests, many number series problems tasks can be found. These tasks cover a wide range of human capabilities for number series problems, like their logical capabilities or calculating skills, which happens to be an important factor for intelligence measurement [RK11]. Some studies [KKW12] compared the complexity of number series problems with subjects performance by simply classifying the number series as "simple" if many subjects have solved them correctly.

Number Series as a Task in IQ Tests

Dealing with number series problems and induction is not easy, either for psychologists or subjects. Number series completion tasks occur often in psychometric IQ tests where the subjects need to complete the number sequence by one or two successor of the sequence. Subjects try to create a rule that can be applied over the whole number sequence in order to find a correct solution. This way of solving number sequence problems and thinking is called induction.
In the Korossy's study [Kor98] two problems of number series completion tasks in IQ tests are discussed: *existence* and *uniqueness* of solution. The first problem is considered with the solvability of a number series, because for some number sequences is difficult to find a logical solution or a general rule that can be applied to the sequence. The second problem Korossy explained as the existence of many possible uniqueness for only one number sequence solution. Despite the fact that subjects apply only one general rule to generate the successor of number sequence, solution or successor can be interpreted as followed [Kor98].
For example, the arithmetic number sequence "1, 3, 5, 7, 9, 11,..." can be defined through three linear-recursive equations [Kor98]:

1. $a_i = a_1 + (i-1)d$ for $i = 2, 3, 4...$ where $a_1 = 1$ is the starting number and $d = 2$
2. $a_i = a_{i-1} + d$ for $i = 1, 2, 3, 4...$ where $d = 2$
3. $a_i = 2 * a_{i-1} - a_{i-1}$ for $i = 3, 4, 5...$

From above mentioned solutions, each of them have different complexity levels, e.g. third solution requires more cognitive effort and time from a subject then the second one. At the end it depends on a subject which solution suggestion it will he use for solving the number sequence in IQ test.

For a successful task completion in IQ tests, only the correctness of a number sequence is important, meaning is the successor correct or not, regardless which one of three above mentioned equations was used.
Here, Korossy [Kor98] mentioned the importance of time for solving number series completion tasks in IQ tests. The subject's performance success in IQ test depends on how fast a subject can solve number sequence. Therefore, if the subject uses more complex equations to solve a number sequence it can be time consuming, which affects the test score. Korossy suggested that in IQ tests, along with number series completion tasks, instructions should be given, as to which rules should be followed for solving these tasks [Kor98].

2.3 Computer Models

Automatic data induction is closely related to computer science, especially to machine learning and artificial intelligence. Number series induction is researched within these disciplines, what is the focus of this Master thesis.
In this chapter different computer systems and models for induction over number series, as well their scope of appliance will be discussed. In the first part inductive programming will be explained together with the Magic-Haskeller system, while in the second part of the chapter different models will be investigated, especially Artificial Neural Networks (ANN) and Semantic Pointer Architecture Unified Network (Spaun).

2.3.1 Inductive Programming

Inductive Programming (IP) integrates different approaches which are related with program synthesis or algorithms from incomplete specifications, such as input/output (I/0) examples. IP focuses on synthesis of declarative, meaning logical or functional programs [HKS08] and it also helps to apply the methods of induction to programming. IP systems can be classified into more subfields: inductive logic programming (ILP), inductive functional programming (IFP) or inductive functional logic programming (IFLP). IP classification depends on the target language [HKS08] and in this chapter the focus is on inductive functional programming (IFP).
IFP is a research field that has found its usage in artificial intelligence discipline which deals with incomplete program specifications, usually represented as I/O examples [Kat12]. Programs based on inductive functional programming are constructed to synthesize the generalized data. The main differences

between current inductive programming systems are [HKS08]:

- target language,
- synthesis strategies,
- information that needs to be presented,
- scope of the program.

Due to various approaches to IFP it is necessary to categorize them into two groups:

1. Analytical approach - is based on the structural analysis of the input/output examples and systems with that kind of approach derive programs from examples. The principle is simple: after the analysis of the I/O examples, system is able to generate a recursive function due to recursive structures. Such systems have many advantages, such as efficiency, faster termination and execution. On the other hand, systems are restricted in terms of specific program classes and there is no efficient way to deal with background knowledge [Kit07].
2. Generate-and-test approach - systems based on this approach construct the first program or more hypothetical programs, which are evaluated against input/output examples and continue to work on them with the most promising hypotheses. Such approach allows systems to be powerful, because can induce programs without bigger restrictions. However, such approach can be time consuming [HKS09].

MagicHaskeller

In [Kat11] MagicHaskeller is described as a system for inducing functions based on systematic exhausting search. MagicHaskeller was developed in 2005 and since then was based on the generate-and-test approach. In the Version 0.8.6 the system was upgraded by analytical search algorithm, called *analytically-generate-and-test approach* [Kat11]. Here, the analytical synthesis enables to generate many programs from input/output examples and chooses those which satisfied the early defined specification [Kat11].
[Kat11] in his paper introduced two kind of modules:

- *Modules for exhaustive search* - are based on the originally generate-and-test approach and define functions that can generate all the programs, constructed as an infinite stream of function applications and lambda abstractions (called primitive set). The "test" approach in this module is function definition for testing previous generated programs which extracts the programs that satisfies a given specification. The advantage of these modules is that they are able to synthesize programs within a simple plan: first select the primitive set and then write a specification (no need for input/output pairs here) in a form of predicate [Kat11].

- *Modules for analytical synthesis* - since the Version 0.8.6 MagicHaskeller system implements the analytical synthesis using the algorithm that extends IGOR2 algorithm (see [Kit07], [HKS09]). Here, the programs are generated as a stream of lists so that afterwards programs with few case splittings could be high prioritized and appear at the top of the list. That kind of program prioritisation and sorting happens during the search without stopping it. The advantage of these modules is that the analytical approach is not that time consuming as the generate-and-test approach. Nevertheless, some problems can occur considering complex function, such as Fibonacci function, which cannot be synthesized by analytical algorithm, but by exhaustive search [Kat11].

Number Series Problems with MagicHaskeller. In year 2012 a group of students at University of Bamberg carried out an empirical study with MagicHaskeller to test number series completion problems [DWZ12]. For their implementation the project group used two approaches: analytical synthesis and generate-and-test approach.
In the first approach they used the quickStart function from the module Run.Analytical to test it over the number series. For example, for the number series "1,2,3,4,5" a correct synthesized program should be delivered, meaning a function that for the input "1" produces the successor "2", for the input "2" produces the successor "3", etc. The function call was defined as followed:

```
> quickStart [ d | f 1 = 2 ; f 2 = 3 ; f 3 = 4 ; f 4 = 5 ; | ]
noBKQ (\f -> f 5 == 6)
```

The quickStart function has three arguments here: a list of input/output pairs (f 1 = 2 ; f 2 = 3 ; f 3 = 4 ; f 4 = 5 ;), a list of input/output pairs for the background knowledge (noBKQ, in this case no background knowledge) and a function that needs to be tested ((\f -> f 5 == 6)). This approach did

not bring any remarkable results, out of the 20 number series only one was solved, which was a successor function [DWZ12].
In the second approach the number series was tested based on the generate-and-test approach from Susumu Katayama and his advice of using ProgramGenerator. Within the ProgramGenerator was integrated the component library, containing basic operators (+,-,/,*), arguments that the project group needed for their number series and a basic function series. The series function is able to generate a stream of a number series based on the starting number and operators. For example, for the number series " 1,3,6,8,16,18,36" the function call was defined as followed:

```
filterSeries (\f -> take 7 (f ( 2::Int)) == [2,5,8,11,14,17,20::Int])
>>= MagicHaskeller.pprs
```

With the call of the above filterSeries function all programs will be filtered, according to the operators and defined arguments. The principle is following: first a number series with starting element 2 (f (2::Int)) needs to be constructed. It already has the first seven elements ([2,5,8,11,14,17,20::Int]) defined. Together with the defined mathematical operators and numbers, the series function generates several number series, which are elements compared with the test function ([2,5,8,11,14,17,20::Int]) [DWZ12].
With this approach MagicHaskeller was able, out of twenty possible number series, to solve seven number series correctly. This study showed MagicHaskeller is able to solve number series problems. [DWZ12] emphasized the potential of improving the performance efficiency and also gave excellent foundations for further work, which has been explored in [GHSST12]. Details of the empirical study with all the results can be found in [DWZ12].

2.3.2 Artificial Neural Networks

Another approach for dealing with number series problems has foundations in the artificial intelligence domain and it is represented by [RK11]. They proposed a new method based on Artificial Neural Networks (ANNs) with a task to solve number series completion problems by a dynamically learning approach.
Characteristics of the artificial neural networks have foundations in the features of the biological neural networks. Still, for a scientist, it is almost impossible to simulate the number of neurons, their interconnections or operations. In many aspects biological neural networks are better than artificial

neural networks, like in following features [Yeg09]:

- Robustness and fault tolerance - performance is not significantly affected by the nerve cells decay
- Flexibility - automatic adjustment of the network to the new environment
- Ability to deal with a variety of data situations - network can deal with incomplete information
- Collective computation - network can perform many operations in parallel.

These biological neural networks features are better in pattern recognition tasks then any other artificial intelligent computer system. The reason for that is, that we cannot fully understand how these processes actually happen [Yeg09]. Still, artificial neural networks are simulating some basics features of the biological networks, such as various kind of information processing. The following figure 2 is representing a model of an ANN.

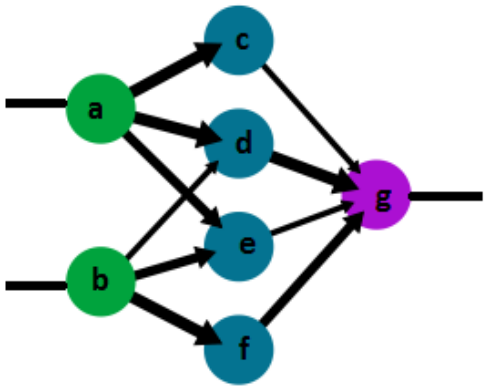

Figure 2: Graphical representation of a simple artificial neural network

According to [Kri07] this model of ANN would be defined as (N, V, w) where:

- $N = a, b, c, d, e, f, g$ is the set of neurons,
- $V = (a, c), (a, d), (a, e), (b, d), (b, e), (b, f), (c, g), (d, g), (e, g), (f, g)$ is the set of relations between neurons and
- $w : V \to \mathbb{R}$ is the level of influence of that relations or their weights (see figure 2).

Such ANN has three different levels: the *input level* with its input nodes a and b, the *output level* with its output node g and the *hidden level* with all the relations between input and output nodes (see figure 2).

Solving number series problems with ANNs. The purpose of an artificial neural network is to learn from the given input data while manipulating the relations between the input and output layer. According to these basic principles [RK11] developed a new method with a dynamic learning approach in order to solve number series problems. Ragni and Klein [RK11] compared their approach with human performance and used an Online Encyclopedia of Integer Sequences (OEIS) database for benchmarking.
For number series solving with ANNs [RK11] used three-layered networks with back-propagation for errors. For the activation function, the hyperbolic tangent was chosen. The input function for the ANN was defined as $f_i = \frac{n}{10^x}$, where n is the number series length and x the higher value of a number series. With that function [RK11] solved the problem of representing the number series values, which were originally in integers, in network interval values [-1,1]. The output function was the inverted f_i function: $f_o = n * 10^x$ where results were converted back into integers. The weights between neurons were randomly assigned between 0 and 1 and the momentum factor was defined as 0.1. For the data analysis, the learning rate, the number of input nodes, the number of hidden nodes and the number of training iterations were systematically varied. For all configurations, only one output node was used.
In order to test the network and generate training patterns, [RK11] build the input pattern from a set of m training values and one target value, which was the outcome. For a network configuration with m input nodes, the n-m patterns were generated.

Pattern	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
p_1	v_1	v_2	v_3	t				
p_2		v_2	v_3	v_4	t			
p_3			v_3	v_4	v_5	t		
p_4				v_4	v_5	v_6	t	
p_5					v_5	v_6	v_7	?

Figure 3: Pattern generation example for a number series with seven given numbers [RK11]

The example from the figure 3 shows how patterns can be generated for an artificial neural network testing using the dynamic approach. In the example the seven elements of the number series are given and the n_8 element needs to be predicted. p_1, p_2, p_3, p_4 are the training patterns, where each of them has three training values v_i, v_{i+1}, v_{i+2} and one target value $t = n_{i+3}$. The pattern p_5 is here to test the ANN without a target value.

Results of the empirical study. There were tested 17 subjects for 20 number series and the performance of 840 network configurations with four variants of training iterations ($i_1 = 500, i_2 = 1000, i_3 = 5000, i_4 = 10000$). The results are documented in the table 2 where C indicates the number of correct answers, I the number of incorrect and U the number of subjects who were unable to solve the number series. The learning rate was varied between 0.125 and 0.875, then the number of input nodes (iterated from 1-6) and the number of hidden nodes as well (iterated from 1-20). Three number series (NS6, NS9, NS10) could not be solved by any ANN configuration. The comparison results of different configurations showed varying results for the iterations number and between number series. Therefore, [RK11] concluded that configurations had a lot of problems with the concept learning of the number series.

[RK11] wanted to show if humans are better at solving number series problems than their ANN with the dynamic approach. Unfortunately, they did not go into further discussion of interpreting the results of humans performance, but from the table 2, we can see that all number series could be solved by subjects. On the other hand, the results also showed that the structure of

Table 2: Results of the empirical analysis

#NS	**Number Series**	**C. I. U.**	$i_1 = 500$	$i_2 = 1000$	$i_3 = 5000$	$i_4 = 10000$
1	12,15,8,11,4	15 0 2	306	385	475	530
2	148,84,52,36,28	12 2 3	555	637	670	689
3	2,12,21,29,36	14 1 2	405	440	502	539
4	2,3,5,9,17	13 1 3	3	7	61	192
5	2,5,8,11,14	9 3 5	581	618	659	667
6	2,5,9,19,37	6 4 7	0	0	0	0
7	25,22,19,16,13	16 0 1	562	615	648	667
8	28,33,31,36,34	17 0 0	121	183	315	332
9	3,6,12,24,48	13 1 3	0	0	0	0
10	3,7,15,31,63	12 3 2	0	0	0	0
11	4,11,15,26,41	8 1 8	6	14	32	22
12	5,6,7,8,10	10 1 6	83	91	65	114
13	54,48,42,36,30	16 1 0	274	299	338	376
14	6,8,5,7,4	16 0 1	134	169	198	219
15	6,9,18,21,42	14 1 2	48	24	94	101
16	7,10,9,12,11	14 0 3	111	202	380	404
17	8,10,14,18,26	13 1 3	57	46	30	29
18	8,12,10,16,12	17 0 0	37	75	41	51
19	8,12,16,20,24	15 0 2	507	546	594	613
20	9,20,6,17,3	16 0 1	255	305	397	406

an ANN has remarkable influence on the performance. [RK11] provided with this study a completely new approach for number series problem solving, to predict the correct successor of a number series, and good foundations for further research.

2.3.3 Semi-analytical model

[SS12] presented the semi-analytical system that solves the number series induction problems, which is similar to the human strategy of dealing with number series problems in IQ tests. The general idea of this approach is split into two steps: (i) define term structure of a number series and (ii) apply the semi-instantiated formula.

Definition induction for number series. Each number series with its elements can be defined algorithmically as formula in order to find solutions for number series problems. For example, the number series "2,3,4" can be defined as follows [SS12]: $b_0 = 2; b_n = b_{n-1} + 1$. In their paper, [SS12] focus in on finding formulas for natural numbers which consist of m elements $c_0, ..., c_{m-1}$ and r terms $t_0, ..., t_{r-1}$ applied for calculating elements of number series such as:

$$a(n) = \begin{cases} c_n & \text{for } 0 \leq n < m \\ t_{(n-m) \bmod r} & \text{for } n \geq m \end{cases}$$

This formula represents the *a term*, which can be: (i) a *constant number*, the value of predecessor a_{n-1}, the position number n of the current element, or (ii) some value of *auxiliary number series* $b(n + j)$, where $0 < i \leq m$ and $j \geq -m$. Such term structure represents only the term types, such as (<predecessor> + <constant>) instead of terms [SS12].
To find a formula for a number series represents a pretty demanding search problem $x_0, ..., x_{s-1}$, where needs to be find $m, c_0, ..., c_{m-1}, r$ and $t_0, ...t_{r-1}$ such that $a_n = x_n$. Nevertheless, [SS12] proposed a combination of heuristic search and analytical simplification with following steps in order to solve such search problem:

1. enumerate heuristically the number series tupels of $m, c_0, ..., c_{m-1}, r$ and $t_0, ...t_{r-1}$,
2. each tupel instantiation is semi-analytically searched, which means that number series is correctly predicted,
3. at the end return the first found instantiation.

The tupels are heuristically enumerated like this: (i) assume that initial constants are not needed (*m=0*) and if it is necessary increase m, (ii) start with easy term structures and if it is necessary increase complexity and (iii) allow auxiliary series just in a case of a process failure.
Term structure complexity is defined as an order: constants < predecessor < position value < auxiliary series. Also, the structure trees with lower height within bounded terms are preferred and within heights, trees are ordered according to the operator [SS12]. Several restrictions and simplifications were given as well [SS12]: (i) induction process was restricted to only one term

(r=*1*), (ii) terms with height > 3 were not used, (iii) division and subtraction operator were not induced, (iv) terms that were not expressed naturally were left out and (v) the induced regularity pattern must occur at least twice in a series (to build and test a hypothesis).
Semi-analytical search is based on the calculating every number of the number series that are not covered by the initial constants under following conditions [SS12]: (i) only if all numbers are equal a <constant> can be instantiated, (ii) for <predecessor> are tested all allowed predecessors from a_{n-1} to a_{n-m}, (iii) a <position> is succeeded only if $x_n = n$ and (iv) for <auxiliary series> the main induction is applied on number that need to be predicted. The following figure 4 represents the algorithm of semi-analytical number series induction.

Input: A number series $x_1, \ldots, x_{s-1}$
for $m \in \{0, \ldots, \lfloor \frac{s}{3} \rfloor\}$ **do**
  $\forall i \in \{0, \ldots, m-1\}\ c_i = x_i$;
  foreach *term structure ts in increasing complexity excluding aux. series* **do**
    $t \leftarrow$ induce term for ts, x, m;
    if $t \neq$ error **then**
      return *Definition with* $c_0, \ldots, c_{m-1}$, t;
    end
  end
end
repeat above loop allowing auxiliary series;

Figure 4: Algorithm for semi-analytical number series induction [SS12]

[SS12] tested the semi-analytic search on 25 000 randomly created number series, which had following operators: ^ + - / *. Also, for induction 4 initial constants and 4 interleaving terms were used.
The results showed surprisingly good performance of the semi-analytical system with 93,2% correct induced definitions. However, the system had problems with inducing definitions with interleaving terms and it should be extended.
This system uses term structure complexity as difficulty measure in order to compare number series problem solving with humans and their performance in IQ tests. Such cognitive analysis of number series is of great importance for this thesis, because it provides good foundations for the humans and IGOR performance comparing.

2.3.4 Spaun Brain Model

Eliasmith et al. [ESC+12], a group of neuroscientists and software engineers, designed a "simple neuron" model of the brain that can deal with different human behaviour called Semantic pointer architecture unified network (Spaun). Even though Spaun is a computer model, it is the representation of a very realistic artificial human brain. In neuroscience brain is considered too complex to be fully understood, and therefore Eliasmith et al. [ESC+12] model is based on the 2.5-million-simulated-neurons, which makes Spaun the most complex, large-scale model simulation of the human brain. Spaun is quite admirable and provides excellent foundations for the understanding of the big brain behaviour gap.
Spaun runs on a supercomputer and it has a digital eye which it uses for visual input. That is the reason why the model is presented only with visual image sequences. Also, Spaun has a physically modelled (robotic) arm that is used for drawing all of its responses as an output.
Eliasmith et al. [ESC+12] presented the results of the study on different tasks that are performed by Spaun. All inputs that were given to the Spaun are 28 by 28 images of handwritten or typed characters, while all outputs were movements of a robotic arm. Eight different tasks performed by Spaun are [ESC+12];

A0 Copy drawing - randomly chosen handwritten digits are given to Spaun, the task of Spaun is to produce the same digit in the same handwritten style.

A1 Image recognition - a randomly chosen handwritten digit is given to Spaun, the task is to produce the same digit written in its *default* writing.

A2 Reinforcement Learning (RM) - here, Spaun should perform a *three-armed bandit task*. For this task Spaun must determinate which of three possible choices generates the greatest stochastically generated reward, while reward contingencies can change from trial to trial.

A3 Serial Working Memory (WM) - Spaun receives a list of any length and it should reproduce it.

A4 Counting - Spaun receives a starting value and a count value and it should write the final value: the sum of the two values.

A5 Question answering - Spaun receives a list of numbers and it should answer one of the two possible question: (i) *"the P question"* - What is in a given position in the list? or (ii) *"the K question"* - At what position is the given number in a list?

A6 Rapid variable creation - Spaun is given syntactic input/output examples (e.g. 0011 -> 11, 0024 -> 24), afterwards it receives a new pattern, which should be completed (e.g. 0014 -> ?).

A7 Fluid reasoning - Spaun should be able to perform a syntactic or semantic reasoning task that is similar to the induction problems from the Raven's Progressive Matrices (RPM) test for fluid intelligence. An example of a task would be completing patterns of the form: 1 2 3; 5 6 7; 3 4 ?

At the beginning of the task, Spaun is first shown "A" and then the number of a task (from 0 to 7). After that, Spaun receives the input, while each image is shown for 150 ms separated by a 150 ms blank. With the invalid inputs the model deals powerful and can perform tasks without any help or intervention.

The Spaun architecture. [ESC$^+$12] developed two architecture types for Spaun: (i) the *anatomical architecture* where models major brain structures and their connections are similar to the ones of a human brain, and (ii) the *functional architecture* representing the computational components and connections based on the anatomical architecture.

Compression is very important for understanding neural information processing. From an aspect of a neuroscientist, information can be compressed from a higher-dimensional (image-based) space into a lower-dimensional (feature) space [ESC$^+$12]. Such information compression is important for Spaun and its architecture. Spaun has certain compression hierarchies: (i) a visual hierarchy where input image is compressed into lower-dimensional firing patterns, (ii) a motor hierarchy decompresses the firing patterns in a low-dimensional space to control the arm and (iii) working memory (WM) which stores serial position information by constructing the compressed firing patterns and it has five subsystems shown in the figure 5. *Information encoding subsystem* maps the visual hierarchy firing pattern to a conceptual firing pattern, *transformation calculation subsystem* extracts elements between input elements, *reward evaluation system* evaluates the input associated reward, *information decoding subsystem* decompresses firing patterns from memory to the conceptual patterns and *motor processing subsystem* maps conceptual patterns to the motor patterns and controls the motor timing [ESC$^+$12].

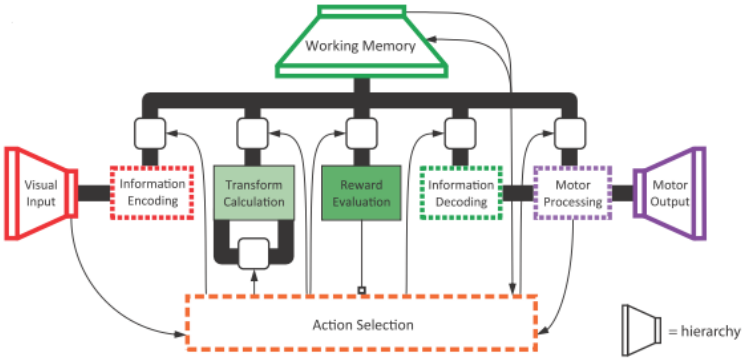

Figure 5: The functional architecture of Spaun [ESC+12]

The results showed why Spaun is so interesting to the scientists and humans: it can solve the basic tasks of an IQ test, like number series problems. Spaun is able to recognize the pattern of a number series and its architecture is very similar to a real brain because it simulates almost the same behaviour. Visual input is processed by the thalamus, neurons storage the data and the basal ganglia controls the task order and sends them to a part of the cortex designed for task handling (see [ESC+12]). This model is designed so precisely that for some tasks it imitates the behaviour of a human, e.g. task A3 consists of a memorizing a list of numbers and then repeating the list back. If the number list is short, Spaun has no problem with memorising the list, but if the number list is longer Spaun cannot complete the list, like humans. Also, the model is very flexible and is able to switch between a wide variety tasks quickly and without problems [ESC+12].
Despite all these advantages, Spaun is faced with several limitations, e.g. non-adaptability (it can not learn a completely new task like human brain), it can not control its own input (it has fixed eyes and attention position), perceptual and conceptual representation limitations. These limitations Eliasmith et al. [ESC+12] see as starting point for new ideas and further development of Spaun.

2.3.5 Asolver and Seqsolver

Strannegård, et al. [SAU13] presented an anthropomorphic[3] method for the Asolver computer program, that is able to solve number sequence problems appearing in the PJP[4] IQ test. The main focus of the Asolver program is search for pattern matching in number sequences based on the anthropomorphic method. In the recent study Strannegård, et al. [SNSE13] presented a similar method for pattern discovery based on the restricted Kolmogorov complexity within the Seqsolver program. These two programs will be explained here.

Asolver with its anthropomorphic method. Asolver recognize the patterns in number sequences as follows: it searches for pattern matching of a given number sequence and discards those patterns that demand too much decoding. Strannegård, et al. [SAU13] emphasized that in the context of IQ tests the only important thing which ensures points on the test is the right answer. This makes the right answer *unique*, which was a starting point for [SAU13] in developing a computer model that can find a "number fitting the pattern". First they observe human test maker and test taker problems with solving and making number sequences and named reasons in order to develop a similar computer model. The reasons were as follows [SAU13]: (i) a number sequence can be reproduced by infinitely many functions, despite that (ii) test maker take only one function (and only one number) as the right answer, therefore (iii) the human cognitive model of a test maker could help the program to choose the right function. Also, they discovered certain limitations of human cognition [SAU13]:

- **person's repertoire** - e.g. for the number sequence 0,2,4,6,8 subjects had no difficulties with recognizing the generator $2 * n$, but with the number sequence 3,6,17,66 none of the subjects could recognize the multicolor Ramsey function.
- **person's generator preference** - e.g. for the number sequence 1,2 some subjects preferred the $n + 1$ generator, other 2^n generator.
- **person's computational abilities** - e.g. for the number sequence 1,1,2,6,24,120 subjects were able to calculate the 6!, but to calculate 9! would be much more difficult.

[3]Here, anthropomorphic means that a computer model emulates the behaviour of a human, e.g. human reasoning

[4] PJP - Predicting Job Performance IQ test developed by Sjöberg, A. ; Sjöberg, S. ; Forssén, K. (2006)

These limitations were seen from [SAU13] as a strategy that benefits the computational complexity of the algorithm and the IQ scores. Therefore, the developed model had: (i) a *limited set of patterns* used for describing number sequences and (ii) a *limited set of computations with bounded cognitive resources* used for decoding the patterns into number sequences.
The pattern notion definition is represented in the figure 6, where b is the number of case base (here, 0, 1 or 2), a_i are constants, t is a term, f a unary function symbol and the variable n is also a term. The pattern for the number sequence 2,5,8,11,14 is shown as an example in the figure 7.

$$f(n) = \begin{cases} a_0 & \text{if } n = 0 \\ \cdots & \\ a_{b-1} & \text{if } n = b-1 \\ t & \text{if } n \geqslant b. \end{cases}$$

Figure 6: A syntactic expression of a pattern [SAU13]

$$f(n) = \begin{cases} 2 & \text{if } n = 0 \\ f(n-1)+3 & \text{if } n \geqslant 1 \end{cases}$$

Figure 7: Example of a pattern definition [SAU13]

For a computational model [SAU13] needed to define notion for bounded computations according to the basic cognitive model C_p, where P is the pattern appearing in the figure 6. The model C_p consists of a two basic sets [SAU13]:

1. the **DM** (declarative memory) set - consists of a four true equations in the form $t'' = t'$ where *t", t'* are terms (e.g. addition table. $a + b = c : a, b \in [1, 10]$ and $a + b = c$).
2. the **PM** (procedural memory) set - consists of the transition rules between terms (e.g. expansion, distribution, power multiplication, recall, etc.)

The general definition of a bounded computation in C_P is a sequence of terms $(t_0, ...t_m)$ such that [SAU13]:

- For all i such that $0 \leq i \leq m-1 (t_i, t_{i+1}) \in PM$.
- For all i such that $0 \leq i \leq m, l(t_i) \leq 8$.

Here, the working memory limit is set on 8, because if the limit would be set on 7 the program wouldn't be able to solve some simple number sequences, and if the limit would be set on 9 it would significantly increase the runtime [SAU13]. Not only these patterns were defined, but also a *simplification* function that allows the program to search for patterns in the subsequences of s if any pattern could not be found in the number sequence s.
Strannegård, et al. [SAU13] defined the pattern preferences notion based on the adjusted Kolmogorov complexity using following criteria for pattern:

- **Comprehensibility** - preferring anthropomorphic patterns
- **Generality** - preferring total patterns
- **Conciseness** - preferring small patterns.

The results of the Asolver program. The Asolver was implemented in Haskell and compared to the software product Mathematica, another software product Maple, the online computational knowledge engine Wolfram Alpha and the Online Encyclopedia of Integer Sequences (OEIS). All these programs were tested on the 11 number sequences of the IQ test PJP and results are represented in table 3, where C stands for correct, I for incorrect and N for none predictions.

While the other programs scored on this test below IQ 100, Asolver scored at least 143.0, meaning Asolver solved all the 11 number sequences correct. Strannegård, et al. [SAU13] concluded that the notion of the anthropomorphic pattern and using it as a filter for removing too demanding patterns was successful. Still, Asolver was not successful in solving more complex number sequences, such as multicolor Ramsey numbers, the binary numbers and the decimal of π what other programs were able to solve. [SAU13] are aware that the number of tested number sequences is very limited, what makes it hard to give another further conclusions about the performance. Still, the test showed that, apart from Asolver, other standard programs scored below an average human. The goal was to describe the anthropomorphic method for

Table 3: Performance results

#PJP	Asolver	Mathematica	Maple	OEIS	WA
19	C	C	C	C	C
20	C	N	C	C	N
21	C	C	C	N	C
22	C	C	I	N	C
23	C	N	I	I	N
24	C	C	C	C	C
25	C	N	I	C	N
26	C	C	I	N	C
27	C	C	C	N	C
28	C	N	C	N	N
29	C	N	I	N	N

the Asolver which performance was at the same level to the human models and introduce new ideas for number sequence problem solving in AI.

Seqsolver and the bounded Kolmogorov complexity. Despite the fact that the Kolmogorov complexity is non-computable, its versions are computable, and can be applied on number sequences problems. The Kolmogorov complexity has several variations, defined as computational resources (e.g. time and space) and used by Turing machines. For example, one Kolmogorov complexity version $K(x)$, where x is an object, is defined as the length of the shortest program p which outputs x and then stops [SNSE13]. This makes Kolmogorov complexity non-computable. For the analysis of the patterns in number sequences Strannegård, et al. [SNSE13] used the Kolmogorov complexity variation of the Levin and Solomonoff that is actually computable. This variation defines Kt as a combination of the program length that the number of execution steps of a program before stopping. Such approach is used for generating number sequences.
The same as in previous study (see [SAU13]) Strannegård, et al. [SNSE13] first defined the cognitive model for pattern discovery and then the computational model for the Seqsolver program. The *cognitive model* for pattern discovery is explained as follows [SNSE13]:

- **Language modelling** - language that describes the number sequence, e.g. 2,2,4,8,32 as term $f(n-2) * f(n-1)$.

- **Term rewriting system (TRS)** - compute terms with numerical values of the language, e.g. $8 * 32$ and $32 * 256$ may be computable.
- **Bounded version of this TRS** - reflect the cognitive resources of subjects, e.g. $8 * 32$ may be computable, but $32 * 256$ not.
- **Find the smallest term** in the bounded TRS.

The *computational model* defines terms and their descriptions, bounded computation (e.g. the term length, which is no longer then 8) which is very similar to the bounded computation in [SAU13] described above. Also also defined subsequences of a number sequence (e.g. odd-positioned elements, even-positioned elements or base case), starting value of the position in a number sequence, which can vary from -9 to 9, and the order of term preferences to avoid the term ambiguities. Strannegård, et al. [SNSE13] defined the term preference order as follows:

- shorter terms are preferred,
- description preference, e.g. description modulo 2 is preferred over description modulo 3 and so on,
- terms not including n are preferred over terms containing n,
- terms predicting smaller values are preferred.

All of these criteria in both, cognitive and computational model, ensure that every program prediction about a number sequence is unique.
The results of the Seqsolver are represented in the table 4. Seqsolver was tested against the same programs as Asolver with difference that it was tested on three IQ tests: the PJP test (with 11 number sequences), the PA test[5] (with 29 number sequences) and IST test[6] (with 38 number sequences).
The results showed that the IQ score of the Seqsolver was at least 130, while for WolframAlpha it was 99. The score of solved number sequence problems is significantly higher than from the other programs (see table4). Here Kolmogorov complexity version is defined by constructing a cognitive model of a human problem solver with its limited cognitive resources, therefore the complexity measure is also limited. This complexity measure differs from other Kolmogorov complexity versions in the way that refers explicitly to the

[5]PA - Personaladministrativa Rådet: PA Number series (1969)
[6]see [LBBA07]

human cognition model. Also, the set of computations developed within this model is large enough for human level performance. In general, these results assure that the resource-bounded Komogorov complexity is very useful in pattern recognition [SNSE13].

Table 4: Performance results of the number sequence problems

IQ test	**Seq-solver**	**Mathe-matica**	**Maple**	**OEIS**	**WA**
PJP $\sum$ 11	11	6	6	4	6
PA $\sum$ 29	29	9	6	11	9
IST $\sum$ 38	28	9	9	11	12

3 Applying Inductive Programming to Number Series Problems

The main subject of this Master thesis is IGOR, system that implements the inductive functional programming. IGOR was developed for a purpose of a doctor dissertation [Kit10] at University of Bamberg and can be classified into the systems mentioned in chapter 2.3.1. The working strategy of IGOR is a combination of the analytical approach and a search in the problem space. This combination is successful for the limitation restrictions of the analytical approach.
In this chapter the general structure and working principle of IGOR will be explained. In the second part we will take a closer look at the IGOR's working principle using a concrete example. In addition, the data representation types in IGOR will be explained.

3.1 An Introduction to IGOR

IGOR2 was developed from the IGOR1 [KS06] version, which again is based on the approach of Summers. Summers [Sum77] developed the inductive functional programming system THESYS for constructing recursive LISP programs from input/output examples. This system was one of the first with a huge influence on the research of inductive programming and the analytical approach up to the present.
IGOR is a functional program that is able to write programs, therefore the programming language in which IGOR was implemented is Maude. Maude is a declarative programming language which can be very well applied for inductive programming. This Master thesis is based on the Maude version of IGOR which is explained in detail in the first part of this chapter.

3.1.1 The IGOR algorithm

IGOR starts from the incomplete specification of a function and combines the analytical approach with the best-first search to construct a correct system that describes the goal function. How exactly IGOR works will be explained on the framework for term rewriting system and constructor term rewriting system (see [Kit07], [HKS08]).
Framework for inductive programming. Here the term rewriting con-

structor system will be described. Induced functional programs are defined as sets of the recursive equations over a signature: $\sum = F \cup C$, where F is the set of the defined functions and C the set of defined constructors [HKS08]. A no defined function symbol p_i is called a *constructor* term. Equations that define a function F are constructed as rewriting rules from left to right forming a term rewriting system (TRS). Therefore, the term rewriting system has a form of:

- *lefthand side (lhs)* - equations have the form $F(p_1, ..., p_n)$, where constructor terms sequences p_i are called patterns. This equation format is also known as pattern matching in functional programming languages. TRS within this form is called *constructor (term rewriting) system (CS)*. The lefthand side is also called the *function head*.
- *righthand side (rhs)* - every variable in the *rhs* must be bound, i.e. it must also occur in the *lhs*.

A general approach. In the first step of induction, IGOR generates the initial hypothesis which consists of one initial rule per target function. Initial rules are least general generalisations (*lgg*) of the example equations. The initial hypothesis is usually incomplete, because the rhs of the equation contains unbounded variables. In addition, a successor hypothesis is computed. Here, one unfinished rule is chosen and several operations are applied to this rule. The result is one or more new sets of rules. One new set of rules replaces the original rule in a hypothesis, which can produce a one successor hypothesis. IGOR chooses the best actual hypothesis and repeats the previous steps. The search is finished when the best actual hypothesis is complete, i.e. until all variables in the rhs equations are bounded. The hypotheses are generated from the input/output examples, therefore they are correct induced hypotheses. A hypothesis is *correct* if for each example equation is: (i) *consistent*, meaning for every input there is an output, and (ii) *complete*, meaning all input/output examples are subsumed [Kit07]. Besides the equations, the background knowledge can be also formulated. Background knowledge is in form of explicitly defined equations and contributes in solving of a problem, which will be explained in the following part.

Computing Sets of Rules

For unfinished rules, IGOR must apply three different rules to compute a set of successor rules which are explained as follows [HKS08].

Pattern refining rule computes a set of more specific patterns in order to generalise case distinction. The principle is following: if a rule has at least two input examples on the left side of the equation it will be replaced by at least two new rules with corresponding patterns for case distinction. Also, at least two input examples must have different constructors at this position, because the variable represents the lgg. The examples that at this position have the same constructor will be subsumed in a new rule. For example, there are some equation examples for the function *reverse* given [HKS08]:

$$reverse([]) = []$$

$$reverse([A]) = [A]$$

$$reverse([A, B]) = [B, A]$$

The initial hypothesis has one rule representing all examples of the *reverse* function containing the unbounded variable in the right side $reverse(Q) = Q2$. The variable Q contains the input of the first example, the empty list [] and inputs of the following examples, the constructor list *cons*. Due to the different constructors in examples, the rule can be partitioned:

$$reverse([]) = []$$

$$reverse([Q, Qs]) = [Q2, Q2s]$$

Help functions rule replaces the rhs by the recursive call of defined function, where the function call argument is considered as a new induction problem, in other words, a help function is created. In example of $reverse([Q, Qs]) = [Q2, Q2s]$ the rhs is the constructor list containing the two unbound variables *Q2* and *Q2s* for that new help functions (*sub1, sub2*) can be introduced [HKS08]:

$$sub1([X]) = [X]$$

$$sub1([X, Y]) = [Y]$$

$$sub2([X]) = []$$

$$sub2([X, Y]) = [X]$$

These equation examples are considered as new induction problem and will be added to the initial rule of the hypothesis:

$$reverse([]) = []$$

$$reverse([Q, Qs]) = [sub1([Q, Qs]), sub2([Q, Qs])]$$

$$sub1([Q, Qs]) = Q2$$

$$sub2([Q, Qs]) = Q3$$

Recursive function calls rule replaces the *subterms* in the rhs, where unbound variables occur, by recursive calls to the new subprograms. In the previous example, the outputs of the help function *sub2* can be matched to the first two equation examples of the *reverse* function. Therefore, the rhs of the *sub2* can be replaced by the recursive call of the *reverse* function. For this purpose the new help function *sub3* is introduced:

$$sub3([X]) = []$$

$$sub3([X, Y]) = [X]$$

These equation examples are considered as new induction problem and will be added to the initial rule of the hypothesis:

$$reverse([]) = []$$

$$reverse([Q, Qs]) = [sub1([Q, Qs]), sub2([Q, Qs])]$$

$$sub1([Q, Qs]) = Q2$$

$$sub2([Q, Qs]) = reverse(sub3([Q, Qs]))$$

$$sub3([Q, Qs]) = Q2$$

This approach allows IGOR a search by keeping only those programs that are correct. The approach makes the search more *efficient* (testing of the found programs is unnecessary due to the program correctness) and the search space can be expanded by introducing the new help functions and background knowledge.

3.1.2 IGOR in Maude

Maude is a functional programming language that follows the principle of homoiconic languages by allowing Maude programs to be data for the same Maude programs. This characteristic of a programming language, is ability to write its own programs.
A Maude module is the functional subpart of Maude, which has defined signature $\sum$, a set of variables X and a term rewriting system over $\sum$ and X. As described in 3.1.1, IGOR defines the input/output examples, background knowledge and induced programs. These components can be represented as valid and evaluateable modules in Maude [HKS09].

Creating IGOR Modules. IGOR's task is to induce a function from a set of given equations, which can be represented in Maude modules. These modules are represented as functional modules fmod. In the figure 8 an induction module for the *last* function is represented. The *last* function has a list of elements as an input and it should give the last element of the list out.

```
1  fmod LAST is
2
3          sorts Item List InVec .
4
5          op [] : -> List [ctor] .
6          op cons : Item List -> MyList [ctor] .
7          op last : List -> Item [metadata "induce"] .
8          op in : List -> InVec [ctor] .
9
10         vars X Y Z V : Item .
11
12         eq last (cons(X, [])) = X .
13         eq last (cons(X, cons(Y, []))) = Y .
14         eq last (cons(X, cons(Y, cons(Z, [])))) = Z .
15         eq last (cons(X, cons(Y, cons(Z, cons(V, []))))) = V .
16
17 endfm
```

Figure 8: Module for induction of *last* function

The sorts are defined in the line 3, Item, List, InVec. The constructors for the data type List must be explicitly specified in the module as ctor (see line 5 and 6). The type definition of a function is explicitly given in the line 7, where [metadata "induce"] specification identifies the function that needs to be induced and input/output sorts as Item. The background knowledge is defined by [metadata ""] specification. For every defined function type of a

dummy constructor must be given, which the input of the List represents in the InVec sort (see line 8). The variables that are used in equations must be explicitly given together with associated sorts (see line 10). The actual equations for the function induction are given in the lines 12-15.

Result of a module call. First, IGOR must be loaded in Maude, then the command for module generalization is given:

```
Maude> load igor2.2.maude

Maude> reduce in IGOR : gen('LAST, 'last, noName) .
```

The reduce call allows the reduction in the IGOR module. Therefore, IGOR can call the defined gen function and the LAST module containing the last function and the background knowledge (here, the background knowledge is not defined noName).
After the induction, the results are shown in the representation of the module LAST (syntactical adapted):

```
rewrites: 6020 in -944565ms cpu (33ms real) (  rewrites/second)

result Hypo: hypo(true, 2, eq last[cons[X0:Item, []]] = X0:Item .

eq last[cons[X0:Item, cons[X1:Item, X2:List]]] = last[cons[X1:Item, X2:List]] .)
```

In the first line the rewrites number is given (6020), following by time needed for induction (-944565ms) and the list with all hypothesises that IGOR has induced. A hypothesis (Hypo) is in the form *hypo(bool,nat,EQ)*, where the first value *bool* indicates if a complete and correct hypothesis is induced, the second value *nat* indicates the partitions of the examples in hypothesises and the third value *EQ* indicates the equations that define the goal function if induction was successful. The second value is always true because IGOR terminates only if a correct and complete hypothesis is induced. For each variable that is induced, the sort is also given (X0:Item). In this example, the Hypo consists of two equations, first equation represents the Base Case with only one-element list as input (cons[X0:Item, []]) and the second equation represents the recursive call.

Performance of IGOR in Maude and Haskell. Originally, IGOR was initially implemented in the termrewriting language Maude and then ported

to Haskell. [HKS09] could not model all Maude's reflexive capabilities in Haskell, still they managed to implement higher-order context into induction and extract information from types and their classes, which were used as background knowledge. The results showed that the performance of IGOR in Haskell is faster then in Maude, due to the fact that Haskell is a compiled language and Maude an interpreted language. This study showed that the key features of IGOR can be implemented in both languages, such as [Hof10]:

- termination by construction,
- arbitrary user-defined data types can be handled,
- using arbitrary background knowledge,
- auxiliary functions as subprograms can be automatically created,
- complex relationships can be learned, such as tree- or nested recursion,
- variables are allowed in the example equations,
- simultaneous induction of mutually recursive target functions,
- using higher-order functions as program schemes.

In the context of Hofmann's [Hof12] Bachelor thesis, IGOR was dealing with solving number series problems and was compared to other systems, such as MagicHaskeller, E-Generalization and ANNs. From the 41 number series IGOR was able to solve 25 number series correctly. The results showed that IGOR is capable of solving number series problems as good as other systems and, in some cases, even better. For example, compared to the E-Generalization IGOR solved the exact same number of series correctly, compared to the MagicHaskeller IGOR showed better results. On the other hand, IGOR showed poor performance compared to the ANNs. Another important aspect for solving number series correctly as [Hof12] indicates, is the representation of the number series and defining background knowledge. The above mentioned studies show the advantages of IGOR in comparison to other inductive programming systems and their possible use in further research.

3.2 Representing Number Series in IGOR

IGOR was developed with the purpose of function induction over different lists, matrices, natural numbers and artificial intelligence problem solving [Kit10]. For the number series induction and representation in IGOR are important arithmetical functions with their operators, such as $+, -, *, /$. In this Master thesis, various arithmetic operators are combined within number series that are used in intelligence test, and IGOR, as inductive system for recursive functions, needs to correctly predict the successor of a number series. Due to this problem, number series needed to be represented in suitable Maude modules.
In the following part different data representation types of number series will be discussed and IGOR's approach toward number series problems will be explained on concrete examples. Here, the outputs will be syntactically adapted for better understanding.

3.2.1 Data representation

The most number series problems in IQ tests are represented in a form such as: 3,5,7,9,11. Unfortunately, such form cannot be used for automatic induction, and it, therefore, must be adapted in a set of input/output equations so that IGOR could induce a function. IGOR needs at least two examples to generalize the function.

Data type and syntax

Since IGOR does not know any data type the natural number can be represented only by recursive constructors:

```
op 0 : -> MyNat [ctor] .
op s_ : MyNat -> MyNat [ctor] .
```

Here, a natural number is defined as 0 followed by many successors s. For example, the number 3 will be represented in IGOR as *s s s 0*. This representation allows IGOR to execute the pattern matching and generalization of equations. Not only the natural number but also the lists can be represented in IGOR. Lists also require constructors:

```
op nil : -> MyList [ctor] .

op _ _ : MyNat MyList -> MyList [ctor] .
```

According to the figure 8, the representation of the module LAST has defined the empty list *nil* and the recursive list constructor. The recursive list constructor defines a list followed by a number and a list. This notation allows better understanding and notion of the equations and induced functions.

Input/Output Equations

The representation of the number series is important for IGOR to induce the function. In order to execute induction over input/output equations, and to induce a generalization, examples must be selected. For a successful generalisation IGOR needs at least three equations for each induced function: Base Case and two recursive equations. This specification can be seen in the figure 8 for the *last* example, where the Base Case equation is simple (see line 12), while following defined equations are getting more complex (see lines 13-15).

Number series can be represented in several representation types. [SS12] introduced three representation types for number series in Maude modules which were tested in IGOR: (i) **representation type 1**: input: list - output: successor, (ii) **representation type 2**: input: position - output: list and (iii) **representation type 3**: input: position - output value.
For the purpose of this Master thesis all number series are represented and tested in the representation type 1 which is the most suitable for the number series problems in IQ tests.
Input: list - output: successor. The result of this representation should be a function that receives a list of natural numbers and as output gives the successor of the number series. For that reason, input/output equations must be defined, so that the equation list becomes gradually longer. For example, the number series 3,5,7,9,11 has four equations (compare to the appendix A.1):

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

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

```
eq Add1((s s s s s s s s s 0)(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 0 .
```

With this representation IGOR has all the previous numbers of the series available and via pattern matching can access every position of the number series. Therefore, in this example the function generalization is based on the given input values, and the recursive call of the Add1 is not necessary. The induced hypothesis is shown as follows:

```
result Hypo: hypo(true, 1,

eq Add1[_ _[s[s[s[X0:MyNat]]],X1:MyList]] = s[s[s[s[s[X0:MyNat]]]]] [none] .)
```

The two underlines _ _ indicate that a list begins from this point on. IGOR has identified that for the function generalization only the first element is important. The rest of the list is summarized and not considered. For that reason, the goal function is formulated as [first_element, rest_of_list]. Pattern matching for the first element of the series indicates that the value of the next element in series is increased by 2 (s s s X0 - s s s s s X0).

Background knowledge

Dealing with number series problems considers the relation detection between the elements, i.e. understanding how a number series is created. Creating number series is mostly related to the use of various arithmetical operators, such as + or ∗. IGOR does not know any operators, therefore the background knowledge for an operator must be explicitly defined.
The background knowledge is obligatory for complex operators (∗ or /), otherwise IGOR would not be able to induce a goal function. In the previous example of the 3,5,7,9,11 number series, where addition occurs (as simple operator), the background knowledge is defined, in order to avoid any possible false induced functions or non-functions. The background knowledge consists of a constructor definition and corresponding equations:

```
op add : MyNat MyNat -> MyNat [metadata ""] .

eq add(s s s 0, s s 0) = s s s s s 0 .

eq add(s s s s s 0, s s 0) = s s s s s s s 0 .

eq add(s s s s s s s 0, s s 0) = s s s s s s s s s 0 .
```

eq add(s s s s s s s s s 0, s s 0) = s s s s s s s s s s s 0 .

To apply the background knowledge such as add, IGOR needs to be given all relevant equations for the given number series. Taking this into account, IGOR generalize over this equations in order to induce a function.
The number series and their arithmetic operations are represented in IGOR in a way very to the humans and their thinking approach. Therefore, in the next chapter the performance of humans and IGOR will be compared and discussed.

4 Results

The main goal of this Master thesis is, firstly, to provide an answer to the question of whether IGOR is able to solve the number series that were represented in the [GHSST12] study, and secondly, to compare the performances of both studies in order to see how we can bring a computer system closer to humans. The main IGOR task is very similar to the task of MagicHaskeller (see 2.3.1). IGOR should be able to induce a function, that is a result of a given number series equations, and to predict a successor of a number series correctly.
The first section of this chapter informs the reader on [GHSST12] study with humans. The scope and the results of IGOR study, which was based on the human study, is presented in the second section. Conclusively, both studies are compared and discussed.

4.1 A study with humans

The number series, that were tested on humans, were developed by a group of students at University of Bamberg [GHSST12]. They wanted to see how humans can solve number series completion tasks, the one that can be frequently be found in IQ test.
The first part explains how number series were generated and which techniques were used. In the second part the results are discussed.

4.1.1 Generating Number Series

The complexity level of number series tasks in IQ tests has a huge impact on the score and human performance. Starting point in generating number series is to define complexity levels for each number series. Each of the generated number series in [GHSST12] study has a different complexity class, which are defined as follows:

- **Operational complexity** - operators can have different types of complexity, therefore addition or subtraction were classified as ***simple*** operators and multiplication, division, or square as more ***complex*** operators. Due to the several problems with number series (see [GHSST12]), *subtraction and division* were left out of the study, while *addition, multiplication* and *square* were used for creating number series.

- **Numerical complexity** - another way of classifying the number series, is to divide them in two groups: number series with ***low*** values and number series with ***high*** values. It is hard to explicitly define where the boundaries between those two values are, so [GHSST12] decided to resolve the numerical complexity of number series instinctively. The need for numerical complexity comes from the assumption that humans have difficulties to work with big numbers like thousands or millions, while for a computer system this should not be the case.
- **Structural complexity** - number series can be grouped, meaning that, different number series can have same general structure. A number series structure is defined as a combination of different operators within one number series. [GHSST12] realised, that only a combination of maximum two operators per number series is possible. For example, multiplication and addition operator can be used together within each number series. The result of the structural complexity are number series groups shown in the table 5.

Table 5: Structural Groups of Number Series

#Group	Number Series Structure	Function Example
Group 1	fix operator & fix constant	$a_n = a_{n-1} + 2$
Group 2	fix operator & series from group 1	$a_n = a_{n-1} + (n + 4)$
Group 3	varying operators & varying constants	$a_{n=2k-1} = a_{n-1} + 2$; $a_{n=2k} = a_{n-1} + 3$
Group 4	two predecessors & fix operator	$a_n = a_{n-1} + a_{n-2}$
Group 5	fix operators & series from group 4	$a_n = a_{n-1} + fib(n, 1, 2)$

In the first structural group of number series always one identical operator is always used, and it needs to be applied to the whole number series. The structural group 2 shows how the position of the number in the number series influences the operator. Third structural group has alternating operators and constants, while structural group 4 builds the successor of a number series based on two predecessors. In the structural group 5 the operator is defined for the whole number series with addition of the series from group 4 as constant (for further explanation see [GHSST12]).
As a result [GHSST12] created 20 number series of different complexity classes and structural groups represented in the table 6.

Table 6: Number Series Complexity Classes

#Number Series	Number Series	Numerical Complexity	Operational Complexity	Structural Complexity
NS1	3,5,7,9,11	simple	simple	sc_1
NS2	2,7,13,20,28	simple	simple	sc_2
NS3	5,7,10,12,15	simple	simple	sc_3
NS4	2,2,4,6,10	simple	simple	sc_4
NS5	1,2,4,7,12	simple	simple	sc_5
NS6	107,291,475,659,843	complex	simple	sc_1
NS7	237,311,386,462,539	complex	simple	sc_2
NS8	128,254,381,507,634	complex	simple	sc_3
NS9	103,103,206,309,515	complex	simple	sc_4
NS10	1,24,47,93,162	complex	simple	sc_5
NS11	1,4,9,16,25	simple	complex	sc_1
NS12	1,1,2,6,24	simple	complex	sc_2
NS13	4,6,12,14,28	simple	complex	sc_3
NS14	2,2,4,8,32	simple	complex	sc_4
NS15	1,1,3,12,84	simple	complex	sc_5
NS16	121,144,169,196,225	complex	complex	sc_1
NS17	7,7,14,42,168	complex	complex	sc_2
NS18	16,48,51,153,156	complex	complex	sc_3
NS19	2,3,6,18,108	complex	complex	sc_4
NS20	3,3,9,36,252	complex	complex	sc_5

We can see from the table 6, that the number series are a combination of different complexities. In the table 7 we can see the function representations for number series. According to the functions, the number series successors can be predicted, that is why the functions were used for creating the Maude modules.

Table 7: Function Representation for Number Series

#Number Series	Number Series	Function how to solve
NS1	3,5,7,9,11	$a_n = a_{n-1} + 2$
NS2	2,7,13,20,28	$a_n = a_{n-1} + (n + 4)$
NS3	5,7,10,12,15	$a_{n=2k-1} = a_{n-1} + 2$; $a_{n=2k} = a_{n-1} + 3$
NS4	2,2,4,6,10	$a_n = a_{n-1} + a_{n-2}$
NS5	1,2,4,7,12	$a_n = a_{n-1} + fib(n, 1, 2)$
NS6	107,291,475,659,843	$a_n = a_{n-1} + 184$
NS7	237,311,386,462,539	$a_n = a_{n-1} + (n + 73)$
NS8	128,254,381,507,634	$a_{n=2k-1} = a_{n-1} + 126$; $a_{n=2k} = a_{n-1} + 127$
NS9	103,103,206,309,515	$a_n = a_{n-1} + a_{n-2}$
NS10	1,24,47,93,162	$a_n = a_{n-1} + fib(n, 23, 23)$
NS11	1,4,9,16,25	$a_n = n^2$
NS12	1,1,2,6,24	$a_n = a_{n-1} * n$
NS13	4,6,12,14,28	$a_{n=2k-1} = a_{n-1} + 2$; $a_{n=2k} = a_{n-1} * 2$
NS14	2,2,4,8,32	$a_n = a_{n-1} * a_{n-2}$
NS15	1,1,3,12,84	$a_n = a_{n-1} * fib(n, 1, 3)$
NS16	121,144,169,196,225	$a_n = (n + 10)^2$
NS17	7,7,14,42,168	$a_n = a_{n-1} * n$
NS18	16,48,51,153,156	$a_{n=2k-1} = a_{n-1} * 3$; $a_{n=2k} = a_{n-1} + 3$
NS19	2,3,6,18,108	$a_n = a_{n-1} * a_{n-2}$
NS20	3,3,9,36,252	$a_n = a_{n-1} * fib(n, 1, 3)$

4.1.2 Method and Results of the Human Study

To run the test with humans, [GHSST12] created four online questionnaires where the number series appeared to the subjects as random as possible. This was done to avoid the sequence effect, so that subjects could not recognize the "pattern" of complexity of number series.

First, subjects were asked some general questions about their age, gender and occupation. After that, the subjects were given two examples of how number series completion can be solved and built. Also, subjects were aware that the time spent on number series solving was recorded for each page. Besides that, subjects were requested not to use any additional material, e.g. calculator or the internet. The task was to complete the number series on each page and rate the difficulty level on a scale from 1 to 5.

Table 8: Human study results

#Number Series	**Number Series**	**Nr. of Correct Results (total 46 partici-pants)**	**Mean time for Solving NS in sec**	**Min-Max Time for Correct Result in sec**
NS1	3,5,7,9,11	46	13,78	5-48
NS2	2,7,13,20,28	44	27,21	8-84
NS3	5,7,10,12,15	46	19,15	8-61
NS4	2,2,4,6,10	35	37,68	12-138
NS5	1,2,4,7,12	20	88	23-341
NS6	107,291,475,659,843	39	81,04	29-210
NS7	237,311,386,462,539	34	78,65	33-327
NS8	128,254,381,507,634	36	97,76	29-337
NS9	103,103,206,309,515	40	73,02	18-505
NS10	1,24,47,93,162	18	144,84	211-218
NS11	1,4,9,16,25	42	29,41	8-95
NS12	1,1,2,6,24	31	82,28	17-317
NS13	4,6,12,14,28	43	37,76	9-176
NS14	2,2,4,8,32	38	43,08	12-217
NS15	1,1,3,12,84	8	185,60	35-1237
NS16	121,144,169,196,225	43	44,65	19-97
NS17	7,7,14,42,168	37	77,28	14-266
NS18	16,48,51,153,156	41	48,89	16-127
NS19	2,3,6,18,108	37	80,43	21-275
NS20	3,3,9,36,252	9	218,65	54-2380

The results of the human study (see table 8) show the performance of 46

participants, 16 female, 29 male subjects and one unspecified. The mean time for solving number series was calculated for all results, even for incorrect ones. The min-max time was calculated only for those subjects who gave the correct answer.
[GHSST12] came to the conclusion that the number series: NS5, NS10, NS15 and NS20 were hard to solve for the most participants due to their high complexity scale (4 and 5). The group 5 of number series was the hardest to solve according to the average solving time. Also, for all number series that have numerical or operational complexity variations the mean time increased dramatically. On the other hand, the results showed how participants needed more time to solve numerically complex and operationally simple number series.
Even though this study faced some problems (see [GHSST12]), it provided foundations for further testing of number series with IGOR and comparing the results.

4.2 Scope and solution times for IGOR

The number series created by the project group (see table 8) were used and implemented in IGOR without being modified. In this chapter the performance of IGOR will be discussed, in order to see for what number series IGOR can induce a function and at which cost.

4.2.1 Performance and Material

All of the 20 number series from the table 8 were tested with IGOR. In order to avoid the misrepresentation, the number series in general were not changed, but taken as they appeared in the human study and modified in Maude modules. To represent the number series in Maude the representation type 1 was used. For the illustration, the example of the first number series 3,5,7,9,11 in representation type 1 is shown in appendix A.1. The number of input/output equations is the result of the element number of each number series. The background knowledge was presented in such order that IGOR could solve number series. The time limit for IGOR was 1 hour and 25 minutes.
This test provides an overview of number series that IGOR is able to solve with regards to the complexity and structure of number series.

4.2.2 Preliminary Study and Performance results

The results of the 20 number series are shown in the table 9. Out of total of 20 number series, IGOR could solve 13 number series correctly within the limited time frame of 1 hour and 25 minutes. A correct response implies that IGOR induced the function which can predict the successor of a given number series.
If we take a look at the number series elements and their value (see table 9), they can be split into two groups:

1. *lower* number series values or *small numbers*, such as NS1, NS13, NS15, etc. where the elements value is < 100 and
2. the *higher* number series values or *big numbers*, such as NS6, NS7, NS8, etc. where the elements value is > 100.

This factor of the *small* vs. *big* number had influence on IGOR performance. For example, number series NS9 or NS16 have obviously higher iteration number than other number series. This can be explained by considering the big number of the start value, which is in the case of NS16: 121. A value of the elements in the NS16 increases constantly, and that can be time consuming and iteration increasing for IGOR.
In the case of *fibonacci* number series (NS5, NS10, NS15 and NS20), IGOR solved two out of four: NS5 and NS10. Here, the problem for not solving NS15 and NS20 is the complexity of the arithmetical operator $*$, which requires more rewriting and longer time limit than 1 hour and 25 minutes. Still, for *fibonacci* number series, IGOR has better performance results then humans, which will be discussed in the next section.
The results show another interesting fact: IGOR has a lot of difficulties with problem solving of the alternating number series. Out of 4 alternating number series (NS3, NS8, NS13 and NS18) IGOR could not solve any of these number series correctly, regardless of the operator complexity or big vs. small numbers. Therefore, for the future data analysis the alternating number series were left out from the study. For better performance of such number series problems [Kit10] suggested the implementation of the *if construct* with the investigation of *even* or *odd* as background knowledge. In this Master thesis the *if construct* was not implemented and tested.
Out of 20 number series, IGOR was able to induce a function for 18 number series and finish before time limit (see table 9). The number series NS7 and NS8 could not be solved within the time limit (1h and 25min) due to the

big numbers problem. Therefore, the preliminary study on the NS7 and NS8 was done to define the maximum constant value that IGOR can handle.

Table 9: IGOR's performance on 20 number series

#Number Series	Number Series	Terminated	Correct solved	Iteration Number
NS1	3,5,7,9,11	x	C	2414
NS2	2,7,13,20,28	x	C	442338
NS3	5,7,10,12,15	x	I	215440
NS4	2,2,4,6,10	x	C	34037
NS5	1,2,4,7,12	x	C	37916
NS6	107,291,475,659,843	x	C	77683
NS7	237,311,386,462,539	t.o.	I	0
NS8	128,254,381,507,634	t.o.	I	0
NS9	103,103,206,309,515	x	C	231424127
NS10	1,24,47,93,162	x	C	6121662
NS11	1,4,9,16,25	x	C	180493
NS12	1,1,2,6,24	x	C	144464
NS13	4,6,12,14,28	x	I	25277519
NS14	2,2,4,8,32	x	C	35190
NS15	1,1,3,12,84	x	I	3310524
NS16	121,144,169,196,225	x	C	450551559
NS17	7,7,14,42,168	x	C	650993
NS18	16,48,51,153,156	x	I	609836228
NS19	2,3,6,18,108	x	C	460287
NS20	3,3,9,36,252	x	I	72693460

x = terminated
t.o. = time-out (after 1h 25min)
C = correct, I = incorrect, N = none

The preliminary study

In the preliminary study two number series were tested, NS7 and NS8, which due to big numbers and big constant value could not be solved within the time limit. Therefore, the search for the right constant value, which resulted

in IGOR solving the number series, is explained in the following section.
The general function for solving NS7 is defined as follows: $a_n = a_{n-1}+(n+73)$ where n is the number series element position and 73 the constant c. The explanation of the function is simple: for the NS7 "237,311,386,462,539" where the starting element is 237, the successor or in this case, the second element of the number series is defined as $237+1+73 = 311$ (predecessor + position in the number series + constant 73), the third element of the number series as $311 + 2 + 73 = 386$ and so on.
The general function for solving NS8 is defined as follows: $a_{n=2k-1} = a_{n-1} + 126; a_{n=2k} = a_{n-1} + 127$, which is a representation of alternating number series. The function can be explained as follows: for the NS8 "128,254,381,507,634" where the starting element is 128, the successor is defined as $128+126 = 254$, the successor of "254" is defined as $254 + 127 = 381$ and so on.
Under the assumption that IGOR can handle big numbers, neither the big numbers nor the starting element values were changed. Therefore, the constants of the number series NS7 and NS8 were changed and new number series were created. This was done in order to see if IGOR can terminate before time limit and induce a function. The constant value was intuitively changed (see table 10), e.g. for NS7 $c = 63, c = 50, c = 40$ etc. and for NS8 $c_1 = 83, c_2 = 84$ and $c_1 = 63, c_2 = 64$. In addition new number series were tested in Maude with new constant values.
The results are shown in the table 10. For NS7, IGOR could not deal with following constants $c = 63, c = 50, c = 40$, meaning that IGOR has neither finished within the time limit nor induced a function. By changing the constant value into $c = 35$ a new number series "237,273,310,348,387" was created for which IGOR could complete the task and induce a function within the time limit. At this point, it is important to mention that the iteration number is extremely big, what makes new NS7 the most complicated number series for IGOR. Again, the reason for that could be the value of the starting element, which is "237". For NS8, IGOR could not deal with $c_1 = 83, c_2 = 84$ constants, but with $c_1 = 63, c_2 = 64$ yes. The new number series were created "128,191,255,318,382".
Although IGOR completed the task in both cases within time limit, it was able to induce the correct function only for NS7. The reason for not inducing a correct function for NS8 is the alternating number series problem for IGOR mentioned before. According to the given results in the table 10, new NS7 is introduced and will be used for further data analysis.

Table 10: The results of the preliminary study on NS7 and NS8

#Number Series	**Original NS & Function**	**Changed constant**	**New NS**	**Terminated**	**Correct**	**Iteration Nr.**
NS7	237,311,386,462,539	c = 63	237,301,366,432,499	t.o.	-	-
		c = 50	237,288,340,393,447	t.o.	-	-
	$a_n = a_{n-1} + (n + 73)$	c = 40	237,278,320,363,407	t.o.	-	-
		c = 35	237,273,310,348,387	x	yes	2979218599
NS8	128,254,381,507,634	$c_1 = 83$; $c_2 = 84$	128,211,295,378,462	t.o.	-	-
	$a_{n=2k-1} = a_{n-1} + 126$; $a_{n=2k} = a_{n-1} + 127$	$c_1 = 63$; $c_2 = 64$	128,191,255,318,382	x	no	1134526435

x = terminated
t.o. = time-out (after 1h 25min)

The goal of the preliminary study was, firstly, to determine which number series problems IGOR can solve correctly within the time limit, and secondly, to eliminate the incorrect number series from further analysis, in this case, the alternating number series NS8.
In addition, the new table 11 was created in order to focus only on number series that IGOR is able to solve correctly. Therefore, all alternating number series (NS3, NS8, NS13 and NS18) and two *fibonacci* number series (NS15 and NS20) were eliminated from further data analysis. The number series are put into four groups shown in table 11, which are defined as follows:

1. Group - number series with small numbers and + operator
2. Group - number series with big numbers and + operator
3. Group - number series with small numbers and * operator
4. Group - number series with big numbers and * operator

These groups are a combination of different complexity factors that could influence the performance of IGOR. Grouping number series allows better comparison between them and humans, which will be discussed in the next section.

Table 11: Performance of IGOR after preliminary study

1st Group of NS				2nd Group of NS			
#NS	**Term.**	**C.S.**	**It.Nr.**	**#NS**	**Term.**	**C.S.**	**It.Nr.**
NS1	x	C	2414	NS6	x	C	77683
NS2	x	C	442338	NS7 NEW	x	C	2979218599
NS3	-	-	-	NS8 NEW	-	-	-
NS4	x	C	34037	NS9	x	C	231424127
NS5	x	C	37916	NS10	x	C	6121662
3rd Group of NS				4th Group of NS			
#NS	**Term.**	**C.S.**	**It.Nr.**	**#NS**	**Term.**	**C.S.**	**It.Nr.**
NS11	x	C	180493	NS16	x	C	450551559
NS12	x	C	144464	NS17	x	C	650993
NS13	-	-	-	NS18	-	-	-
NS14	x	C	35190	NS19	x	C	460287
NS15	x	I	-	NS20	x	I	-

Term. = Terminated
C.S. = Correct Solved
It.Nr = Iteration Number
x = terminated
C = Correct
I = Incorrect

4.3 Comparison of Human and Computational Results

The main questions that need to be answered in this research are: Can IGOR solve the same number series that humans do? How many number series can IGOR solve? What could be the reason why IGOR can not solve some number series? Is the performance of IGOR influenced by the higher/lower element values of a number series or by the complex/simple operator within a number series? Can we relate those effects to human performance? What did influence the performance of humans more: the higher/lower element value of a number series or the complexity of an operator? Answers to those

questions will be offered in this chapter.
In the first part the performance of humans and IGOR will be compared and discussed, while in the second part the performance by groups between humans and IGOR will be presented and discussed.

4.3.1 Performance of Human and IGOR

The performance comparison between humans and IGOR is done based on data for mean time and iteration number. Unfortunately, the mean time for number series solving from the human study also includes the number series that were solved incorrectly by participants. The reason why mean time instead of min-max time (which includes only correct solved number series) is used in this comparison is due to the fact, that this data were actually comparable, while min-max time was not represented well. On the other hand, the only "useful" data from IGOR were the rewrites or iteration number. The real time could not be used as comparison factor because it depends on a computer used in this study and its performance capabilities.

The results of humans and IGOR are represented in the table 12, where all solved number series together with performance results are shown. These number series, are split into groups, as mentioned in the section 4.2.2. Therefore, it is to conclude that IGOR is able to solve number series problems from each group, meaning, IGOR can deal with big or small numbers and with simple or complex operators. Different structural complexities mentioned in the human study (see table 6) do not represent a major problem for IGOR to solve. For example, number series of structural complexity *sc_5*: NS5, NS10, NS15 and NS20, IGOR could solve NS5 and NS10 correctly. On the other hand, humans solved number series of the structural complexity *sc_5* with the highest error rate and mean time. If IGOR would be given longer time limit, maybe it could predict a successor also for NS15 and NS20.
Regarding the mean time, iteration number and different complexities, it is to conclude that both humans and IGOR showed similar effects. The figure 9 shows the number series represented in a graph by performance of IGOR vs. humans.

Table 12: Compared results of Humans and IGOR by groups

Problem		Humans			IGOR		
#Group	**#NS**	**Nr. of correct results ($\sum 46$ participants)**	**Mean time in sec**	**Min-Max Time for Correct Result in sec**	**Correct Solved**	**Iteration Nr.**	**Real time in ms**
g_1	NS1	46	13,78	5-48	x	2414	29
	NS2	44	27,21	8-84	x	442338	204
	(NS3)	-	-	-	-	-	-
	NS4	35	37,68	12-138	x	34037	18
	NS5	20	88	23-341	x	37916	20
g_2	NS6	39	81,04	29-210	x	77683	1608
	NS7_NEW	34	78,65	33-327	x	2979218599	4491452
	(NS8)	-	-	-	-	-	-
	NS9	40	73,02	18-505	x	231424127	2117831
	NS10	18	144,84	211-218	x	6121662	5736
g_3	NS11	42	29,41	8-95	x	180493	90
	NS12	31	82,28	17-317	x	144464	97
	(NS13)	-	-	-	-	-	-
	NS14	38	43,08	12-217	x	35190	20
	(NS15)	-	-	-	-	-	-
g_4	NS16	43	44,65	19-97	x	450551559	324020
	NS17	37	77,28	14-266	x	650993	1123
	(NS18)	-	-	-	-	-	-
	NS19	37	80,43	21-275	x	460287	580
	(NS20)	-	-	-	-	-	-

x = terminated

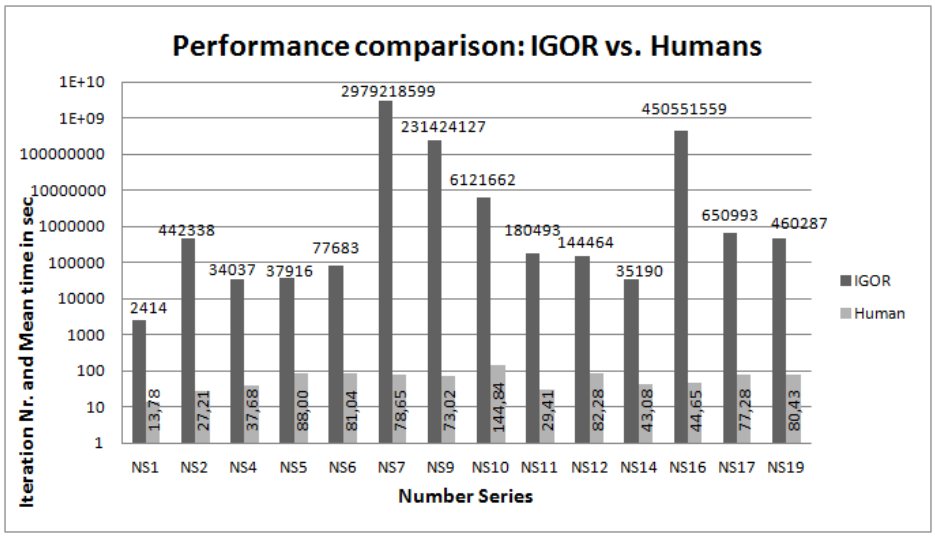

Figure 9: IGOR vs. Humans

The figure 9 shows the performance of IGOR vs. humans by iteration number and mean time. It can be concluded that IGOR needs more iterations than humans need time to solve number series. However, the difference is that IGOR solved number series 100% correctly, while human's mean time consisted of both correct and incorrect answers.

Number series with high numerical complexity and low operational complexity, such as NS6, NS7, NS9 and NS10, were harder for humans to solve (regarding time) [GHSST12]. For the same number series, IGOR needed more iterations to finish. Therefore, the Pearson's correlation was calculated to see if the mean time and iteration number of number series depend on each other. The correlation is presented in the figure 10.

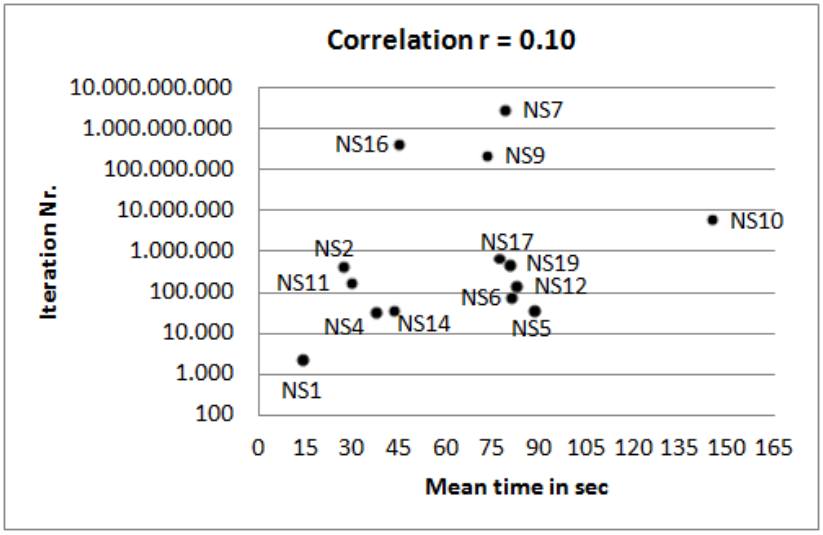

Figure 10: Correlation

The calculated correlation value was r = 0,10. The positive correlation value (between 0 and 1) means that if the mean time (for humans) of number series increases so does the iteration number (for IGOR) of number series. Still, the correlation value of r = 0,10 shows that the mean time and iteration number are weakly related. Number series with the higher mean time do tend to have higher iteration numbers, but knowing only the mean time of a number series it is not possible to predict the iteration number precisely. It is to conclude, that the relationship between the mean time and iteration number is a weak positive relationship.

It is to conclude that IGOR is able to solve the same number series problems as humans. Out of 20 number series IGOR solved 14 number series correctly (together with the new NS7 from the preliminary study). These 14 number series will be used for the group analysis and discussion in the following section.

4.3.2 Performance of Human and IGOR by Groups

The performance of IGOR is influenced by the small or big numbers and by the complex or simple operator within a number series, which can be seen by comparing the groups of number series. These effects can be related to the human performances as well. In the following section, four groups are

defined (see section 4.2.2) in order to compare them and see if there exists a possible relation between groups for humans and IGOR.
Firstly, the group 1 and group 2 will be compared, where the similarities among human and IGOR performances are investigated. Number series from the first group have small numbers (or low value elements) and number series from the second group have big numbers (or high value elements). The figure 11 and 12 represent, to be compared human and IGOR performances.

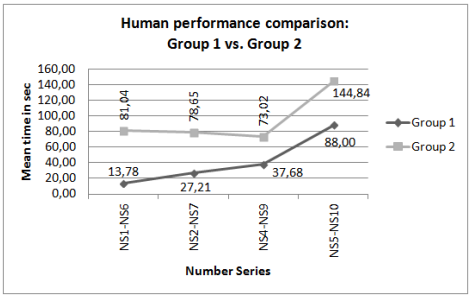

Figure 11: Humans

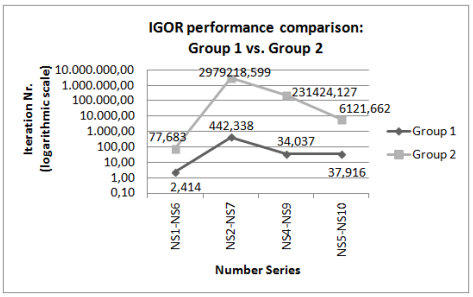

Figure 12: IGOR

Considering human performance (see figure 11), the mean time for number series from group 1 is slowly increasing. For NS5 from group 1, humans needed twice as much time then for NS4, the same effect happens by comparing NS10 and NS9 from group 2. The mean time for number series NS6, NS7 and NS9 from group 2 is slowly decreasing instead of increasing as in group 1. For group comparison, it is to conclude that humans needed at least twice as much time to solve number series problems with big numbers, than to solve number series problems with small numbers.
On the other hand, IGOR's performance on iteration number (see figure 12), for group 2 in contrast to group 1, is significantly higher. At this point, we can conclude that big numbers in number series have greater influence on iteration number than small numbers. Also, it is obvious, that iteration number for NS2 in group 1 and NS7 in group 2 deviate from other number series.
In general, it can be concluded that big numbers within number series have significant influence on both, human and IGOR's performance. Therefore, we can use big vs. small numbers as a comparison factor among human and IGOR.

The following comparison is between group 1 and group 3. At this point, we will analyse the operator influence, where group 1 has number series with + operator and group 3 number series with * operator. Human and IGOR

performances are compared on figures 13 and 14.

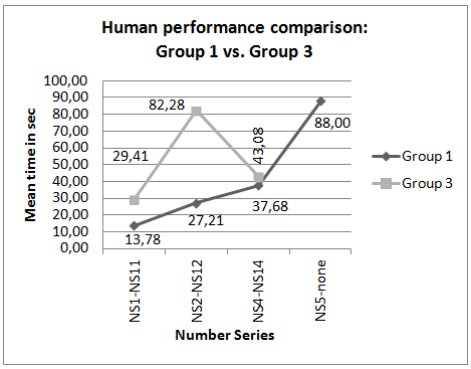

Figure 13: Humans

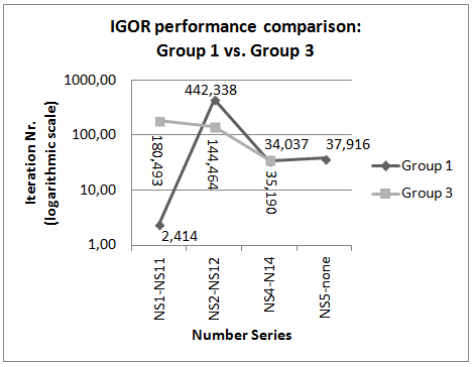

Figure 14: IGOR

According to the human performance graph (see 13), humans need more time to finish number series for group 3 than for group 1, meaning the * operator has influenced the mean time. Still, the performance difference is not that significant for the number series pair NS4-NS14. The mean time for NS14 is hardly increased comparing to the NS4, therefore in this case the operator * makes just a small difference in performance.
For the IGOR performance graph (see 14) in general the operator * is relevant to the iteration number. Although, looking at the NS2-NS12 pair, the operator * makes no difference to iteration number, because IGOR needed less iterations for NS12 then for NS2. Also, the performance of the pair NS4-NS14 is similar to the humans performance, with slightly higher iteration number for NS14.
It is to conclude, that the use of a complex operator * vs. simple operator + within number series, can be an important factor for human performance, while for IGOR performance is not of such great importance.

The following comparison is between group 2 and group 4, where the similarities among human and IGOR performances are investigated. The comparison is based on a combination of big numbers and operator + (group 2) vs. big numbers and operator * (group 4) within number series. The figures 15 and 16 compare human and IGOR performances.

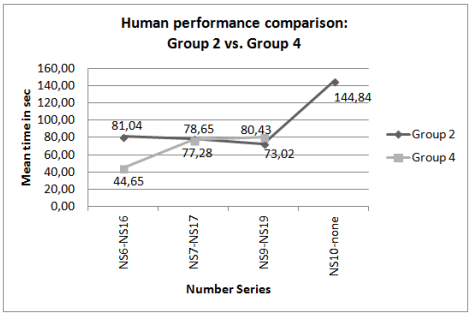

Figure 15: Humans

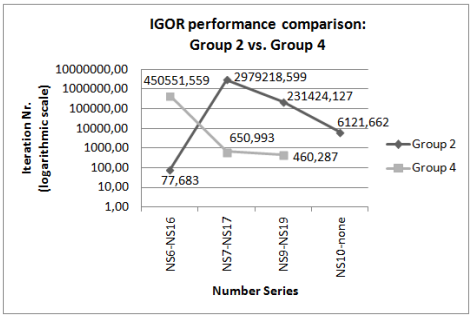

Figure 16: IGOR

Looking at human performance (see figure 15) for group 2 and group 4, we can see better performance results for group 4 than for group 2. Number series NS16 and NS17 demanded from humans less time for solving then the NS6 and NS7. Therefore, combination of a complex operator * and big numbers within a number series is not automatically reason for time increasement.
If we compare the IGOR performance (see figure 16) for group 2 and 4, we can notice better performance for group 4. Number series NS17 and NS19 showed remarkably lower iteration number then NS7 and NS9. It is to conclude that IGOR can deal better with number series that have complex operator * and big numbers then with number series with operator + and big numbers.
Although, the similarities among human and IGOR performances can not be applied over the same number series pairs, in general, we can conclude that humans and IGOR can deal better with number series of group 4. The reason for that might be, that in group 2 the interval of big number is between 100 and 800, while in group 4 between 100 and 250.

The last comparison is between group 3 and group 4, where the similarities among human and IGOR performances are investigated. The comparison is based on a number series with small numbers and operator * (group 3) vs. big numbers and operator * (group 4). The figures 17 and 18 show comparison of human and IGOR performance.

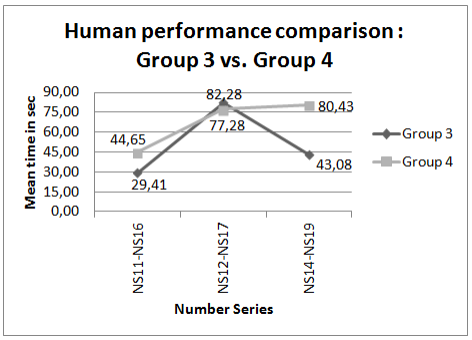

Figure 17: Humans

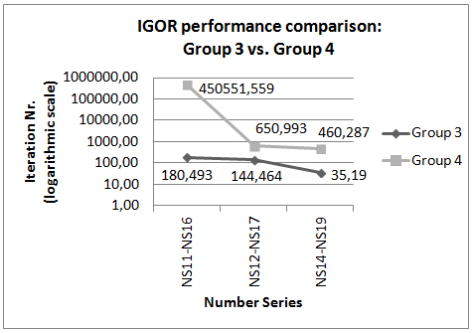

Figure 18: IGOR

The human performance comparison (see figure 17) between group 3 and 4, showed in general better performance for group 3. Humans needed less time in solving number series NS11 an NS14 then NS16 and NS19. Still, in one number series of group 4: NS12 humans showed better performance results then for NS17 from the group 3.
On the other hand, IGOR performance comparison (see figure 18) between group 3 and 4, showed better performance of group 3. For each number series from group 3 the iteration number was extremely lower than in group 4, therefore, the combination of small numbers and operator * within number series is better for IGOR then combination of big numbers and operator *.
We can also notice a lot of similarities between humans and IGOR performances. The effect of better results for number series with small numbers and operator * is related to both, humans and IGOR.

Discussion and Interpretation. From the above mentioned data and results it can be concluded that IGOR needs more iterations in order to solve number series problems with big numbers then number series with small numbers. Also, from the preliminary study we can see that the start value and the constant value have impact on the time and are from big importance for IGOR to solve number series within the given time limit. Therefore, the run time and iteration number for number series with big numbers and big element values will increase. The effect of big numbers can be related to the human study as well, which showed that humans need more time to deal with number series containing big numbers.
The complexity of the operator can also influence the performance of IGOR. The differences in performance in using the operator + instead of the operator * did not have major significance for IGOR, meaning IGOR had no problem with solving several number series problems containing the operator

* with better performance than for number series problems containing the operator +. While humans on the other hand, required more time to solve number series containing complex operator *.
Another interesting fact was that, number series containing a complex operator * and big numbers surprisingly showed better performance results for both humans and IGOR than number series containing a simple operator + and big numbers. The big number interval of number series between groups should be further investigated, because it may be relevant for the performance results.
Also, the combination of small numbers and operator * within number series showed better performance results for IGOR and humans than the combination of big numbers and operator *. Here, we can conclude that, the influence of big numbers within number series is significantly affects the performance results among IGOR and humans.
To conclude, the above represented results show the interaction effects between number series groups for humans, which can be compared and related to IGOR. Those effects between humans and IGOR are compared on a specific pattern that occurs within each group, such as big numbers or small numbers within number series. Therefore, the comparison of humans and IGOR relies on such patterns and allows a different level of interpreting computational and human performance results.

5 Conclusion

This Master thesis shows that IGOR, a computer system, can solve number series problems that occur in IQ tests, as good as humans, as already mentioned in chapter 4.1. A big impact on solving number series and IGOR's performance had the complexity level of a number series and high or small element value within number series (see 4.2). Another interesting aspect for observing IGOR's performance is the run time limit, which also had impact on IGOR's performance. During the testing, the time limit was set to 1 hour and 25 minutes. Even though some number series demanded such an extended time limit, IGOR showed excellent time performance for some number series. For example, for number series from group 1 and 3 IGOR needed less then 100 milliseconds to solve, while humans on average needed around 45 seconds to solve the same number series.

Despite the positive summary, this study was faced with some problems. For example, IGOR could not predict a correct successor for alternating number series. For the future research of alternating number series problems with IGOR, the background knowledge should be more investigated. IGOR must discover and search for equations from the background knowledge that are essential in order to solve number series. This approach is time consuming and here [Hof12] suggested the automatic limitation of the background knowledge. Moreover, [Kit10] suggested introducing the *if construct*.

Considering performance comparison between humans and IGOR, it was hard to compare both performances. Firstly, considering the time needed to solve number series, we used mean time in the human study. The mean time consisted of both correct and incorrect answers, which was hard for a direct comparison with IGOR, because IGOR solved the number series either correctly or incorrectly. Secondly, the real time from IGOR's performance could not be used as a comparison factor because it mostly depends on the computer's performance, where number series were tested. Therefore, the results of the performance comparison of IGOR's iteration number and humans mean time showed a weak relationship. According to the correlation (see 4.3.1) calculation number series with higher mean time tend to have bigger iteration number, but it is a weak positive relationship.

Still, there are some number series characteristics, such as complexity of the operator or small and big numbers, which can be used as pattern for comparison between humans and IGOR. According to the group comparison (see 4.3.2), the effects of the human groups comparison are related to the

IGOR's group comparison. Hence, the effect of small and big numbers can be seen among humans and IGOR (group 1 vs. group 2), then the effect of big numbers in a combination with operator + or * (group 2 vs. group 4) and the effect of small numbers in a combination with operator * (group 3 vs. group 4). The comparison of group 1 and group 3 showed similarities between humans and IGOR in a small measure.

To continue the research on this field it would be good to continue the testing start values and constant values of a number series with IGOR. This would inform on IGOR's ability to handle big start and constant values. Also, another human study could be carried out, which would give better representation of performance results, especially time results and could, therefore, be better compared to the presented IGOR results. Moreover, one could consider a new way of decreasing the run time limit and iteration number in IGOR for complex number series.

References

[BS16] BINET, Alfred ; SIMON, Theodore ; KITE, Elizabeth S. (Trans): *The development of intelligence in children (The Binet-Simon Scale).* Williams & Wilkins Co, Baltimore, 1916

[DWZ12] DÜSEL, Matthias ; WERNER, Alexander ; ZEIßNER, Theresa: Solving Number Series with the MagicHaskeller. (2012)

[ESC+12] ELIASMITH, Chris; STEWART, Terrence C. ; CHOO, Xuan ; BEKOLAY, Trevor ; DEWOLF, Travis ; TANG, Yichuan ; RASMUSSEN, Daniel: A Large-Scale Model of the Functioning Brain. In: *Science 338*, 1202 (2012)

[EKF+03] ERBRECHT, Prof. Dr. R. ; KÖNIG, Dr. H. ; FELSCH, Matthias ; KRICKE, Dr. W. ; MARTIN, Karlheinz ; PFEIL, Wolfgang ; WINTER, Dr. R. ; WÖRSTENFELD, Willi: *Das große Tafelwerk - Formelsammlung für die Sekundarstufen I und II.* Cornelsen, 2003

[Gar83] GARDNER, Howard: *Frames of mind: The theory of multiple intelligences.* Basic Books, New York, 1983

[GHSST12] GRÜNWALD, Roland; HEIDEL, Elke; STRÄTZ, Alexander; SÜNKEL, Michael; TERBACH, Robert: *Induction on Number Series.* Bamberg: Otto-Friedrich-Universität Bamberg, 2012

[Hof10] HOFMANN, Martin: IgorII - an Analytical Inductive Functional Programming System (Tool Demo). In: *Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation* (2010), pp. 29-32

[Hof12] HOFMANN, Jaqueline: *Automatische Induktion über Zahlenreihen - eine Fallstudie zur Analyse des induktiven Programmiersystems IGOR2.* Bamberg, Bachelor's Thesis, 2012

[HHNT86] HOLLAND, John H. ; HOLYOAK, Keith J. ; NISBETT, Richard E. ; THAGARD, Paul R.: *Induction: processes of inference, learning, and discovery.* Cambridge, MA, USA : MIT Press, 1986

[HKS08] HOFMANN, Martin ; KITZELMANN, Emanuel ; SCHMID, Ute: Analysis and Evaluation of Inductive Programming Systems in a Higher-Order Framework. In: *KI 2008: Advances in Artificial Intelligence* (2008), pp. 78-86

[HKS09] HOFMANN, Martin ; KITZELMANN, Emanuel ; SCHMID, Ute: Porting IgorII from MAUDE to HASKELL. In: *Proceedings of the ACM SIGPLAN Workshop on Approaches and Applications of Inductive Programming* (2009), pp. 65-74

[HKS14] HOFMANN, Jaqueline; KITZELMANN, Emanuel ; SCHMID, Ute: *Applying Inductive Program Synthesis to Induction of Number Series - A Case Study with IGOR2.* Bamberg, 2014

[HPG82] HOLZMAN, Thomas G. ; PELLEGRINO, James W. ; GLASER, Robert: *Cognitive Dimension of Numerical Rule Induction.* In: Journal of Educational Psychology (1982), pp. 360-373

[HPG83] HOLZMAN, Thomas G. ; PELLEGRINO, James W. ; GLASER, Robert: *Cognitive Variables in Series Completion.* In: Journal of Educational Psychology (1983), pp. 603-618

[Kat11] KATAYAMA, Susumu: MagicHaskeller - System Demonstration. In: *Workshop on Approaches and Applications of Inductive Programming, Proceedings* (2011), pp. 63-70

[Kat12] KATAYAMA, Susumu: An Analytical Inductive Functional Programming System that Avoids Unintended Programs. In: *Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation* (2012), pp. 43-52

[Kit07] KITZELMANN, Emanuel: Data-driven Induction of Recursive Functions from Input/Output-Examples. In: *Proceedings of the Workshop on Approaches and Applications of Inductive Programming* (2007), pp. 15-26

[Kit10] KITZELMANN, Emanuel: *A combined Analytical and Search-Based Approach to the Inductive Synthesis of Functional Programs.* Bamberg, Diss., 2010

[KS06] KITZELMANN, Emanuel ; SCHMID, Ute: Inductive Synthesis of Functional Programs: An Explanation Based Generalization Approach. In: *Journal of Machine Learning Research* 7 (2006), pp. 429-454

[KKW12] KIPMAN, Ulrike ; KÖHLBÖCK, Gabriele ; WEILGUNY, Walburga: *Psychologische Testverfahren zur Messung intellektueller Begabung.* ÖZBF, Salzburg, 2012

[Kor98] KOROSSY, Klaus: Solvability and Uniqueness of Linear-Recursive Number Sequence Tasks. In: *Methods of Psychological Research Online* Volume 3, (1998), pp.43-68

[KS73] KOTOVSKY, Kenneth ; SIMON, Herbert A.: *Empirical tests of a theory of human acquisition of concepts for sequential patterns.* In: Cognitive Psychology 4 (1973), No. 3, pp. 399-424

[Kri07] KRIESEL, David: *Ein kleiner Überblick über Neuronale Netze.* (2007), available on http://www.dkriesel.com

[LBBA07] LIEPMANN, D. ; BEAUDUCEL, A. ; BROCKE, B. ; AMTHAUER, R.: Intelligenz-Struktur-Test 2000 R (I-S-T 2000 R). In: *Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie* (2007), Volume 32, No. 3, pp. 166-169

[RK11] RAGNI, Marco ; KLEIN, Andreas: Predicting Numbers : An AI Approach to Solving Number Series. In: *KI 2011: Advances in artificial intelligence* (2011), pp. 255-259

[SS12] SIEBERS, Michael ; SCHMID, Ute: Semi-Analytic Natural Number Series Induction. In: *KI 2012: Advances in Artificial Intelligence* Volume 7526, (2012), pp. 249-252

[Spe04] SPEARMAN, Charles: General Intelligence, Objectively Determined and Measured. In: *The American Journal of Psychology* 15 (1904), No. 2, pp. 201-292

[Ste85] STERNBERG, Robert J. : *Beyond IQ: A Triarchic Theory of Intelligence.* Cambridge University Press, Cambridge, 1985

[SAU13] STRANNEGÅRD, Claes ; AMIRGHASEMI, Mehrdad; ULFSBÄCKER, Simon : An anthropomorphic method for number sequence problems. In: *Cognitive Systems Research* Volume 22-23, (2013), pp. 27-34

[SNSE13] STRANNEGÅRD, Claes ; NIZAMANI, Abdul Rahim ; SJÖBERG, Anders ; ENGSTRÖM, Fredrik : Bounded Kolmogorov Complexity Based on Cognitive Models. In: *Computer Science* Volume 7999, (2013), pp. 130-139

[Sum77] SUMMERS, Phillip: A Methodology for LISP Program Construction from Examples. In: *Journal of the ACM* (1977), pp. 161-175

[Thu38] THURSTONE, Louis L. : *Primary Mental Abilities*. The University of Chicago Press, Chicago, 1938

[Yeg09] YEGNANARAYANA, B. : *Artificial Neural Networks*. Prentice-Hall of India, New Delhi, 2009

A Appendix

A.1 NS1 Module in Representation type input: list - output: successor

```
fmod NS1 is

sorts MyItem MyList MyNat InVec .

*** DT definitions (constructors)
    op 0 : -> MyNat [ctor] .
    op s_ : MyNat -> MyNat [ctor] .
    op nil : -> MyList [ctor] .
    op _ _ : MyNat MyList -> MyList [ctor] .

*** input encapsulation
    op in : MyList -> InVec [ctor] .
    op Add1 : MyList -> MyNat [metadata "induce"] .
***3,5,7,9,11
eq Add1((s s s 0) nil) = s s s s s 0 .
eq Add1((s s s s s 0)(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)(s s s 0) nil) = s s s s s s s s s 0 .
eq Add1((s s s s s s s s s 0)(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 0 .

***BKQ
    op add : MyNat MyNat -> MyNat [metadata ""] .
    op in : MyNat MyNat -> InVec [ctor] .

    eq add(s s s 0, s s 0) = s s s s s 0 .
    eq add(s s s s s 0, s s 0) = s s s s s s s 0 .
    eq add(s s s s s s s 0, s s 0) = s s s s s s s s s 0 .
    eq add(s s s s s s s s s 0, s s 0) = s s s s s s s s s s s 0 .

endfm
```

Ich erkläre hiermit gemäß §17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum	Unterschrift