Applying Inductive Program Synthesis to Induction of Number Series – A Case Study with IGOR2*

Jacqueline Hofmann[1], Emanuel Kitzelmann[2], and Ute Schmid[3,**]

[1] University of Constance, Germany
jacqueline.hofmann@uni-konstanz.de
[2] Adam-Josef-Cüppers-Berufskolleg Ratingen, Germany
ekitzelmann@gmail.com
[3] Faculty Information Systems and Applied Computer Science, University of Bamberg, 96045 Bamberg, Germany
ute.schmid@uni-bamberg.de

Abstract. Induction of number series is a typical task included in intelligence tests. It measures the ability to detect regular patterns and to generalize over them, which is assumed to be crucial for general intelligence. There are some computational approaches to solve number problems. Besides special-purpose algorithms, applicability of general purpose learning algorithms to number series prediction was shown for E-generalization and artificial neural networks (ANN). We present the applicability of the analytical inductive programming system IGOR2 to number series problems. An empirical comparison of IGOR2 shows that IGOR2 has comparable performance on the test series used to evaluate the ANN and the E-generalization approach. Based on findings of a cognitive analysis of number series problems by Holzman et al. (1982, 1983) we conducted a detailed case study, presenting IGOR2 with a set of number series problems where the complexity was varied over different dimensions identified as sources of cognitive complexity by Holzman. Our results show that performance times of IGOR2 correspond to the cognitive findings for most dimensions.

1 Introduction

The ability to reason inductively is one of the core features of any intelligent system (Holland, Holyoak, Nisbett, & Thagard, 1986; Schmid & Kitzelmann, 2011). In cognitive science research, inductive reasoning is considered as basic mechanism for knowledge expansion by exploiting previous experience (Tenenbaum, Griffiths, & Kemp, 2006). Inductive reasoning is also a central component of intelligence tests – often completion of number series is used to measure inductive reasoning ability. In the context of research on the Turing test

* The work reported here was conducted by Jacqueline Hofmann in her bachelor thesis in Applied Computer Science at University of Bamberg, Dec. 2012.
** Corresponding author.

C. Lutz and M. Thielscher (Eds.): KI 2014, LNCS 8736, pp. 25–36, 2014.

to evaluate machines, researchers proposed algorithms which are able to solve number series as one of several problems (Dowe & Hernández-Orallo, 2012).

Typically, a number series completion problem is presented as a series of numbers whose elements feature a special relation among one another which can be described algorithmically. To solve the series, a generalized pattern has to be identified in the given series and this pattern has then to be applied to generate the most plausible successor number. In the 1980s, Holzman et al. analyzed the influence of several characteristics of number series – like the magnitude of the starting value and the kind of operator involved in the series – on human performance (Holzman, Pellegrino, & Glaser, 1982, 1983). These characteristics can be used for systematic evaluation of computational approaches to number series problems and their comparison to human performance.

In general, computational approaches for number series induction can be intended as a cognitive model or as a general AI approach, they can be based on symbolic or sub-symbolic computation, and they can be designed specifically to solve number series or they can be realized as a specific application of a general induction algorithm. For example, Strannegård, Amirghasemi, and Ulfsbäcker (2013) proposed a special-purpose, symbolic, cognitive model which relies on predefined patterns and which addresses the (small) class of number series problems used in IQ tests. Siebers and Schmid (2012) presented a symbolic AI algorithm based on a pre-defined set of numerical operators (addition, subtraction, division, multiplication and exponentiation). Patterns which capture the regularity in a series are generated by analysis of the given numbers and hypotheses are enumerated based on a heuristics. Ragni and Klein (2011) presented an application of an artificial neural network (ANN) which was trained with a large number of examples of different partial sequences of a number series and learned to (approximately) predict the next number. Burghardt (2005) demonstrated that E-generalization can be applied to generate rules which capture the regularities in number series.

E-generalization extends the notion of anti-unification by the use of equational theories as background knowledge. For an equational theory E, a term t is the E-generalization or E-anti-unification of two terms t_1, t_2, if there are two substitutions σ_1, σ_2, such that $t\sigma_1 =_E t_1$ and $t\sigma_2 =_E t_2$, where $=_E$ denotes the smallest congruence relation containing all equations from E. Given equational theories for $+, *, if$ and *even*, examples can be generalized to patterns such as $v_p * v_p$ for $0, 1, 4, 9$, or *if(even(v_p), v_p, 1)* for $0, 1, 2, 1, 4, 1$ where v_p represents the current position.

Providing specialized algorithms which can deal with inductive reasoning problems typically used to measure human intelligence is interesting for cognitive science research. However, in our opinion, demonstrating that a general purpose machine learning algorithm can be successfully applied to such a task is of interest not only to cognitive AI research but for AI in general. Comparing the two general-purpose systems which have been applied to number series problems, in our opinion, E-generalization is the more convincing approach: In E-generalization, the generalization is based on the identification of the regularities in the number series and the most simple pattern which covers the series is

constructed. In contrast, the ANN application is trained with many partial sequences of the original input. Furthermore, E-generalization learns a generalized rule which can not only be applied to predict the next number in the sequence but which can also be presented as an explanation for the proposed solution. In the ANN approach, on the other hand, the output is an approximative prediction of the next number in the sequence only.

Analytical approaches to inductive programming can be considered as a generalization of the E-generalization approach proposed by Burghardt. Inductive programming systems address the problem of generalizing recursive functional or logical programs from small numbers of input/output examples (Flener & Schmid, 2009). Like E-generalization, analytical approaches generate hypothetical programs by identifying regularities in the given examples and generalize over the found regularities. However, since the hypothesis language is a subset of the set of possible programs, the hypothesis space is larger than for E-generalization and therefore, inductive programming approaches have a larger scope. A current analytical system is IGOR2 (Kitzelmann, 2009). The authors of IGOR2 already demonstrated the applicability of IGOR2 to cognitive domains such as learning from problem solving, reasoning, and natural language processing (Schmid & Kitzelmann, 2011). For that reason, we identified IGOR2 as a suitable inductive programming system to investigate its applicability to automated induction of number series.

The rest of this paper is organized as follows: We shortly introduce IGOR2 and propose some ideas for the representation of number series as an inductive programming problem. Afterwards, we demonstrate that IGOR2 can be successfully applied to number series problems which were investigated by (Ragni & Klein 2011) and (Burghardt, 2005). We present a detailed case study we conducted to ascertain the specific characteristics of number series – including characteristics identified in psychological research – which influence the performance of IGOR2. We conclude with a short evaluation and discussion of future research.

2 Solving Number Series with IGOR2

IGOR2 is an inductive programming system which learns functional (MAUDE or HASKELL) programs from small sets of input/output examples. For instance, given examples for reversing a list with up to three elements, IGOR2 generalizes the recursive *reverse* function together with helper functions *last* and *init* (see Figure 1). IGOR2 relies on constructor-term-rewriting (Baader & Nipkow, 1998). That is, besides the examples for the target function, the data types have to be declared. For lists, the usual algebraic data type `[a] = [] | a:[a]` is used.

The algorithm of IGOR2 was developed from IGOR1 (Kitzelmann & Schmid, 2006) which is in turn based on Summers's inductive system THESYS (Summers, 1977). While many inductive programming systems are based on generate-and-test strategies (Quinlan & Cameron-Jones, 1993; Olsson, 1995; Katayama, 2005), IGOR2 uses an analytical, example-driven strategy for program synthesis. That is, generalized programs are constructed over detected regularities in the

I/O Examples

```
reverse  [] =  []       reverse [a,b]   = [b,a]
reverse [a] = [a]       reverse [a,b,c] = [c,b,a]
```

Generalized Program

```
reverse   []   = []
reverse (x:xs) = last (x:xs) : reverse(init (x:xs))
```

Automatically Induced Functions (*renamed* from $f1$, $f2$)

```
last [x]    = x                     init [a]    = []
last (x:xs) = last xs               init (x:xs) = x:(init xs)
```

Fig. 1. Inductive Programming with IGOR2: Generalizing *reverse* from Examples

examples. IGOR2 incorporates concepts introduced in inductive logic programming (Muggleton & De Raedt, 1994), mainly, the possibility to use background knowledge (in form of additional functions which can be used while synthesizing a program besides the pre-defined constructors) and the invention of additional functions on the fly (see *init* and *last* in Figure 1). A detailed description of IGOR2 is given by Kitzelmann (2009), an empirical comparison of the performance of IGOR2 with other state-of-the-art inductive programming systems is given by Hofmann, Kitzelmann, and Schmid (2009).

The task of solving number series with IGOR2 differs from its primary application area, that is, induction of recursive functions over lists. However, IGOR2 was already applied successfully to several domains of cognitive rule acquisition, namely learning from problem solving, reasoning, and natural language processing (Schmid & Kitzelmann, 2011). Therefore, we consider the system a good candidate for an approach to solving number series. To apply IGOR2 for induction of a constructor-function which correctly describes and continues a given number series, as a crucial first step, we have to decide how to represent the needed data types, equations, and background knowledge.

Data Types. IGOR2 is able to handle any user defined data type. For our purpose, to represent sequences containing natural numbers, we needed constructors for numbers and for lists. Numbers are defined recursively by the base case 0 and the successor of a natural number, s. The list constructors are defined in the established form of the empty list and a constructor adding an element to an existing list.

Input/Output Equations. To make IGOR2 generalize over the given number series, the sequences have to be presented as input/output equations. There are different possibilities to represent number series in such a way: (1) The input can be presented as a list and the output as successor number, (2) the input can be presented as index position in the list and the output as number sequence up

(1) Input List – Output Successor Value

```
eq Plustwo((s 0) nil) = s^3 0
eq Plustwo((s^3 0) (s 0) nil) = s^5 0
eq Plustwo((s^5 0) (s^3 0) (s 0) nil) = s^7 0
```

(2) Input Position – Output List

```
eq Plustwo(s 0) = (s 0) nil
eq Plustwo(s^2 0) = (s^3 0)(s 0) nil
eq Plustwo(s^3 0) = (s^5 0)(s^3 0)(s 0) nil
eq Plustwo(s^4 0) = (s^7 0)(s^5 0)(s^3 0)(s 0) nil
```

(3) Input Position – Output Value

```
eq Plustwo(s 0) = s 0
eq Plustwo(s s 0) = s s s 0
eq Plustwo(s s s 0) = s s s s s 0
eq Plustwo(s s s s 0) = s s s s s s s 0
```

Fig. 2. Different Representations for the Number Series Problem `PlusTwo` (For representations (1) and (2) successor sequences are abbreviated for better readability)

to this position or (3) as the value at this position. An example for the number series `PlusTwo` $(1, 3, 5, 7)$ is given in Figure 2. To generalize a recursive function from the equations it is necessary to explicitly state every equation which could result from a recursive call of one of the given equations. Therefore, an example series $1, 3, 5, 7$ has to be represented by three equations for representation 1 and four equations for representations 2 and 3.

Background Knowledge. In the context of number series, IGOR2 has to be provided with knowledge about arithmetic operations. These operations can be specified as background knowledge (BK). BK is presented in the same manner as the input/output equations – with the only difference that there is no generalization on the BK equations. This means that every equation which could occur while inducing the series has to be defined explicitly. For example, for the series $2, 4, 8, 16$ we need to define $2 * 2 = 4$, $2 * 4 = 8$ and $2 * 8 = 16$, if we want IGOR2 to be able to use this BK by pattern matching. In this context, the BK given to IGOR2 corresponds to the way arithmetic knowledge is handled in other symbolic approaches such as action planning (Ghallab, Nau, & Traverso, 2004) or in cognitive architectures such as ACT-R (see, e.g., the model discussed in (Lebiere, 1999)). However, the number of equations – input/output as well as background knowledge – is critical for performance.

3 Comparing IGOR2 with E-Generalization and ANNs

We conducted a first test with IGOR2 on solving number series on the sequences which were tested with the ANN (Ragni & Klein 2011) and E-generalization

Table 1. Sample of series tested with ANN and IGOR2

By ANN and IGOR2:	7,10,9,12,11	$f(n-1)+3, f(n-1)-1$
By IGOR2 but not by ANN:	3,7,15,31,63	$2 * f(n-1)+1$
By ANN but not by IGOR2:	6,9,18,21,42	$f(n-1)+3, f(n-1)*2$
Not by ANN and not by IGOR2:	2,5,9,19,37	$f(n-1)*2+1, f(n-1)*2-1$

Table 2. Series solved by E-generalization and IGOR2

0,1,4,9	$v_p * v_p$	0,1,2,1,4,1	*if* $(ev(v_p); v_p; 1)$
0,2,4,6	$s(s(v_p))$	0,0,1,1,0,0,1,1	$ev(v_2)$
0,2,4,6	$v_p + v_p$	0,1,3,7	$s(v_1 + v_1)$
1,1,2,3,5	$v_1 + v_2$	1,2,2,3,3,3,4,4,4,4	—

(Burghardt, 2005) approaches. To examine the general ability to solve these number series we decided to test representations 1 and 2 for the input/output equations and set the time-out to 30 minutes.[1]

Ragni and Klein (2011) presented an initial evaluation of their ANN approach with 20 selected number series as well as an extensive evaluation with over 50,000 number series from the Online Encyclopedia of Integer Sequences (OEIS[2]). We tested IGOR2 with the initial 20 number series. The ANN approach could solve 17, IGOR2 14 of these series correctly. However, IGOR2 was able to correctly generalize two number series which no configuration of the networks was able to predict. Example series are given in Table 1. Here, the first problem was solved by IGOR2 without background knowledge, the second problem with multiplication as background knowledge. Note that the 14 problems were solved with *one instance* of IGOR2 while for ANN 840 network configurations were run where the highest number of series solved by one single network was 14.

To compare IGOR2 with the E-generalization approach we used the 8 example series reported in Burghardt (2005). The results show that both systems solve the same 7 series (see Table 2). Background knowledge for IGOR2 was addition and multiplication. Background knowledge for E-generalization was addition, multiplication, conditional expression, and even.

During this first test we were able to find some general constraints for IGOR2 on the task of correctly generalizing number series. Due to the chosen constructor-symbols which define a natural number as a successor of 0, we found that there is no elegant way to represent *negative numbers*. Even defining a predecessor constructor and binding it with the successor via background knowledge did not lead to success.

Another difficulty came up with *descending number series*. IGOR2 was able to solve these sequences with the needed background knowledge (e.g., division), but

[1] Note that run-times for IGOR2 are typically below 100 ms. Detailed performance results for IGOR2 on all three representations are given in section 4.

[2] `https://oeis.org/`

not with every kind of representation. Even if representation 1 (input: list, output: successor) is the most intuitive one, the input format restricts the solvability of number series. As the input represents the whole number series, it automatically restricts the possible input for the generalized function. If we want IGOR2 to generalize over $16, 8, 4$, the generalized function accepts as lower bound 4, because this is the smallest number presented in the input. This problem results from the definition of natural numbers as successors of natural numbers, which makes it easy for IGOR2 just to count up, but difficult to subtract some s from the given number.

The last limitation appeared by trying to solve *alternating sequences*. To date we were not able to implement some kind of separating function to split functions or divide them by the parity of the current position. Therefore, IGOR2 has only very restricted ability to solve alternating number series, yet. Solving alternating series is possible for IGOR2, if it is able to find a global operation of two or more alternating operations. E.g., for a number series alternating adding 3 and subtracting 1 (see first line in Table 1), IGOR2 induces the operation *add 2* to the pre-predecessor in the sequence. Otherwise, if one alternating sequence could be split up in two individual and independent series without intersection, IGOR2 is able to generalize the original sequence by simply pattern matching over the given elements. E.g., a constructor-function for the series $1, 2, 1, 4, 1$ (see Table 2) can be correctly induced by checking if the predecessor is 1 and if so, adding 2 to the pre-predecessor. If not, the induced function just appends 1 to the number series.

4 A Case Study on Solving Number Series

For a systematic evaluation of the performance of IGOR2 we systematically constructed number series varying in different characteristics. Several of these series are included in OEIS. The evaluation was conducted on a computer with Windows 7, Intel Core i5-2410M with 2.3 GHz and 8 GB RAM.

4.1 Materials

Holzman et al. defined a set of characteristics of number sequences which influence humans on their performance when solving such sequences (Holzman et al., 1982, 1983). Based on these findings we generated a set of 100 number series, varying in the following features: *Operator* $(+, *, */+, =)$, *Number of Operators* (1 or 2), *Magnitude of Starting Value* (low, high), *Magnitude of Argument* (low, high, variable), *Recursion* (linear, cascading), *Reference* (pre-/prepre-/preprepre-decessor, pre- and prepredecessor), *Position* (yes or no).

To be able to compare all three kinds of representation, we omitted operators producing descending number series, like $-$ or $/$. We had to distinguish the *magnitude of argument* for the operators. Multiplication by 2 was considered as low, multiplication by 3 as high. For addition, low is in the range of $2-4$ and high $11-13$. For the equivalence operator $(=)$ the argument is variable. *Magnitude of*

starting value was defined as low in the range of $1-4$ for all operators and high in the range of $14-17$ for addition and $3-8$ for multiplication. *Reference* specifies the element in the number series from which the current value is calculated.

We used mostly number series of a length of five elements. Seven elements were used, if the reference is the preprepredecessor. The background knowledge for every number series was explicitly given and optionally used by IGOR2.

4.2 Hypotheses

Although this was the first time IGOR2 was applied on solving number series, we tried to formulate some assumptions of how it performs:

Assumption for Representations. As representation 1 offers the whole list to IGOR2 as input and just asks for the successor as output, we considered it to be the fastest representation. Furthermore, we expected representation 2 to be much slower than representation 3, because the output of the second version is the whole list, which means additional computation time.

Assumption for Reference. We expected IGOR2 to perform more slowly for larger distances between the reference value and the actual value. As the predecessor is available to IGOR2 before the prepredecessor, IGOR2 will first try to build the constructor-function with the value which is available earlier before moving left in the sequence.

4.3 Results

Every number series was solved ten times by IGOR2 while we recorded the needed time to solve it in milliseconds. Although IGOR2 is a deterministic approach, some variations in run time typically occur, e.g., due to processes of the operating system.

Differences in Representations. With respect to the different representations, the results show that the means of the running times of representation 1 and 3 are quite similar (36 ms vs. 33 ms). Pairwise t-tests showed that there is no significant difference. However, there is a much higher variance in the running times of representation 1 than representation 3 (standard deviations 198 for representation 1 and 74 for representation 3). That is, representation 3 does not differ strongly in its individual results and is therefore more predictable. Representation 2 turned out to be significantly ($p < 0.001$) slower than the others with a mean of 83 ms.

Furthermore, we examined the interaction between the kind of representation and a) the operator and b) the reference in the series with two-factorial ANOVAs and Tukey post-hoc tests. Interaction-analysis showed significant ($p < 0.001$) differences for the *representations* and *operators*. As the Tukey post-hoc analysis showed, the operators = and + differed significantly within the representations.

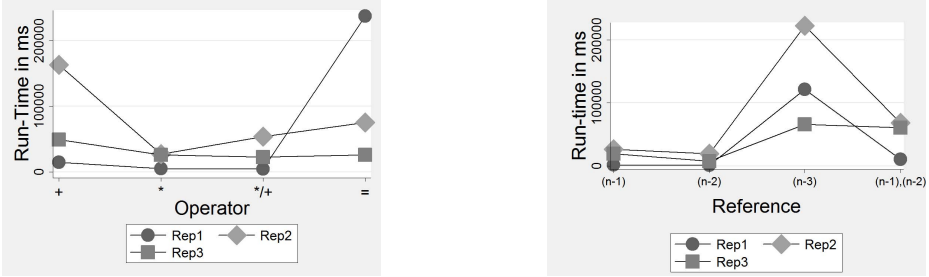

Fig. 3. Interaction of Representation Format and Operator (a) / Reference (b)

This result is illustrated in Figure 3(a). The interaction between *representation* and *reference* in the sequence showed no significant effect. As shown in Figure 3(b), the shapes of the graphs look quite similar, except for the reference point of pre- and pre-predecessor, where inference times for representation 3 were slightly (but not significantly) slower than for representation 1.

Regression Model To Predict Running Time. To assess the relative influence of the different characteristics of the number series, we conducted a multiple regression analysis for the varied characteristics. We excluded the features recursion and number of operators because they can be explained completely through other features (like reference and operator). The parameter position was also ignored because of an imbalanced distribution of the frequencies of its characteristics. The coefficients of the analysis are shown in table 3. As we used dummy variables for all parameters within the model, there are no different scales whereby the coefficients can be directly compared. The model explains 23.98%, 32.48% and 26.40% of the variances of representations 1, 2, and 3.

The magnitude of sequence elements has a comparable impact for all three representations. A higher magnitude of the starting value leads to a significant rise of the mean running time. The magnitude of the argument, however, shows less impact, but also the tendency for higher running times for higher magnitudes.

Regarding the kind of operator, the influence on different representations differ from each other. As interaction-analysis has already shown, the running times of representation 1 are significantly higher for the equivalence operator than for the others. In contrast, representation 3 shows the tendency to be faster for the equivalence operator than for the others. However, this tendency becomes only significant in comparison to addition. For representation 2, running times for addition are significantly higher than for the equivalence-relation, whereas multiplication could be solved significantly faster.

The coefficients for the references were much more surprising. In contrast to our initial assumption, the nearer the reference the faster the algorithm, the regression analysis shows different results. The running times for referring to the pre-predecessor instead of the predecessor do not show significant differences. Even a

Table 3. Multiple regression-analysis for the running times

	Representation 1		Representation 2		Representation 3	
	B	t	B	t	B	t
Starting Value	69343.9***	6.31	127320.7***	11.21	52832.8***	13.07
Arg_high	1905.3	0.15	10659.6	0.80	11601.1*	2.43
Arg_var	33752.1	1.43	-33008.2	-1.36	8916.3	1.03
+ vs. =	-188598.6***	-6.89	67657.0*	2.39	24700.3*	2.45
∗ vs. =	-198261.2***	-7.24	-68438.0*	-2.42	1587.0	0.16
∗/+ vs. =	-199648.3***	-6.69	-59873.6	-1.94	2.207	0.00
(n-2)	-72.65	-0.00	-10445.2	-0.69	-9711.4	-1.80
(n-3)	120029.7***	8.17	192171.5***	12.66	48409.7***	8.96
((n-1),(n-2))	6682.1	0.24	54458.6	1.92	34595.5***	3.42
β_0	128650.5***	3.97	-16099.1	-0.48	-22568.5	-1.89

Table 4. *

***** $p < 0.05$ ****** $p < 0.01$ ******* $p < 0.001$
B = unstandardized coefficient

small tendency to faster running times for the pre-predecessor can be seen (as coefficients are negative). By moving the reference one step further left, the running times become significantly slower than before for all three representations.

4.4 Interpretation

The results of the ANOVAs as well as the regression analysis illustrate in general that with higher values within the number series the running times rise. The fact that the magnitude of the starting value has a much higher influence than that of the argument can be explained by the use of background knowledge. As mentioned above, we decided to explicitly give the needed background knowledge to IGOR2. Therefore, if the argument to be added to an existing value is 3 or 13 does not make a strong difference. Nevertheless, the model shows a tendency towards faster running times for lower arguments, which results from the general increasing values in the number series if a larger argument is added.

Regarding the results of the influence of the kind of operator, the discovered differences are not very evident. The significant increase of the running time for representation 1 for the equivalence operator resulted from the very high running times for two number series with the operator =. Unfortunately, we were not able to detect the reason for these values.

Only about a third of the existing variance within the independent variable running time can be explained with the presented model. However, we expected these values to be a lot higher because we have generated the series completely from the parameters covered in the model. Deeper analyses of the concrete running time values for each number series did not reveal any further systematic variance. We recognized that IGOR2s performance varied strongly by minimal changes of the values of the elements. As our model represents the magnitudes nominally instead of metrically, we developed a second model for a subset of the

number series to see if this explains more variance within the variable. The new model explained a total of 32, 12% of the existing variance for representation 1.

5 Conclusions and Further Work

We could demonstrate that the general-purpose inductive programming system IGOR2 can be applied to solve number series problems. To our knowledge, most approaches to number series induction are based on algorithms specifically designed for this problem class. Two applications of general-purpose algorithms previous to our IGOR2 experiments are the ANN approach of Ragni and Klein (2011) and the application of E-generalization by (Burghardt, 2005). We could demonstrate that IGOR2 can solve a superset of series solvable with E-generalization. Even compared to the ANN approach, its scope is acceptable and IGOR2 can solve some types of series not solvable by the ANN.

Performance constraints of IGOR2 are due to the use of background knowledge and its limited ability to deal with negative numbers and alternating sequences. In our evaluation we presented only those arithmetic operations in background knowledge which were necessary to solve a series. This is necessary because matching examples against background knowledge results in exponential growth of run time. This problem could be reduced by providing a pre-filtering of the possibly needed background knowledge. To deal with negative numbers, one could think of an extra implementation of integers and not only natural numbers. For solving alternating series, it is necessary to implement some kind of query to rely on invariances, such as the parity of the actual position. As a next step, we plan to introduce such extensions.

In our opinion, the generic induction algorithm underlying IGOR2 can be seen as one possible realization of a cognitive rule acquistion device. In Schmid and Kitzelmann 2011 we could show that IGOR2 can be applied to learn generalized strategies for Tower of Hanoi and other problem solving puzzles. In this paper we demonstrated that the same algorithm can be applied to number series problems.

References

Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998)

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

Dowe, D.L., Hernández-Orallo, J.: IQ tests are not for machines, yet. Intelligence 40(2), 77–81 (2012)

Flener, P., Schmid, U.: An introduction to inductive programming. Artificial Intelligence Review 29(1), 45–62 (2009)

Ghallab, M., Nau, D., Traverso, P.: Automated planning: Theory and practice. Morgan Kaufmann, San Francisco (2004)

Hofmann, M., Kitzelmann, E., Schmid, U.: A unifying framework for analysis and evaluation of inductive programming systems. In: Goerzel, B., Hitzler, P., Hutter, M. (eds.) Proceedings of the Second Conference on Artificial General Intelligence (AGI 2009), Arlington, Virginia, March 6-9, pp. 55–60. Atlantis Press, Amsterdam (2009)

Holland, J., Holyoak, K., Nisbett, R., Thagard, P.: Induction – Processes of inference, learning, and discovery. MIT Press, Cambridge (1986)
Holzman, T.G., Pellegrino, J.W., Glaser, R.: Cognitive dimensions of numerical rule induction. Journal of Educational Psychology 74(3), 360–373 (1982)
Holzman, T.G., Pellegrino, J.W., Glaser, R.: Cognitive variables in series completion. Journal of Educational Psychology 75(4), 603–618 (1983)
Katayama, S.: Systematic search for lambda expressions. Trends in Functional Programming, 111–126 (2005)
Kitzelmann, E.: Analytical inductive functional programming. In: Hanus, M. (ed.) LOPSTR 2008. LNCS, vol. 5438, pp. 87–102. Springer, Heidelberg (2009)
Kitzelmann, E., Schmid, U.: Inductive synthesis of functional programs: An explanation based generalization approach. Journal of Machine Learning Research 7, 429–454 (2006)
Lebiere, C.: The dynamics of cognition: An ACT-R model of cognitive arithmetic. Kognitionswissenschaft 8(1), 5–19 (1999)
Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming, Special Issue on 10 Years of Logic Programming 19-20, 629–679 (1994)
Olsson, R.: Inductive functional programming using incremental program transformation. Artificial Intelligence 74(1), 55–83 (1995)
Quinlan, J.R., Cameron-Jones, R.M.: FOIL: A midterm report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 3–20. Springer, Heidelberg (1993)
Ragni, M., Klein, A.: Predicting numbers: An AI approach to solving number series. In: Bach, J., Edelkamp, S. (eds.) KI 2011. LNCS, vol. 7006, pp. 255–259. Springer, Heidelberg (2011)
Schmid, U., Kitzelmann, E.: Inductive rule learning on the knowledge level. Cognitive Systems Research 12(3), 237–248 (2011)
Siebers, M., Schmid, U.: Semi-analytic natural number series induction. In: Glimm, B., Krüger, A. (eds.) KI 2012. LNCS, vol. 7526, pp. 249–252. Springer, Heidelberg (2012)
Strannegård, C., Amirghasemi, M., Ulfsbäcker, S.: An anthropomorphic method for number sequence problems. Cognitive Systems Research 22-23, 27–34 (2013)
Summers, P.D.: A methodology for LISP program construction from examples. Journal ACM 24(1), 162–175 (1977)
Tenenbaum, J., Griffiths, T., Kemp, C.: Theory-based Bayesian models of inductive learning and reasoning. Trends in Cognitive Sciences 10(7), 309–318 (2006)