Making Universal Induction Efficient by Specialization

Alexey Potapov[1,2] and Sergey Rodionov[1,3]

[1] AIDEUS, Russia
[2] National Research University of Information Technology, Mechanics and Optics, St. Petersburg, Russia
[3] Aix Marseille Université, CNRS, LAM (Laboratoire d'Astrophysique de Marseille) UMR 7326, 13388, Marseille, France
{potapov,rodionov}@aideus.com

Abstract. Efficient pragmatic methods in artificial intelligence can be treated as results of specialization of models of universal intelligence with respect to a certain task or class of environments. Thus, specialization can help to create efficient AGI preserving its universality. This idea is promising, but has not yet been applied to concrete models. Here, we considered the task of mass induction, which general solution can be based on Kolmogorov complexity parameterized by reference machine. Futamura-Turchin projections of this solution were derived and implemented in combinatory logic. Experiments with search for common regularities in strings show that efficiency of universal induction can be considerably increased for mass induction using proposed approach.

1 Introduction

Knowledge about the appropriate degree of universality of intelligence is essential for artificial general intelligence, since it determines the strategy of research and development in this field. Relatively generally accepted point of view supposes that intelligence consists of a set of specialized modules, which cooperation yields a synergetic effect [1]. On the other hand, models of universal intelligence exist [2, 3], which in theory possess capabilities unachievable by compositions of narrow methods, but possibly indispensable for general intelligence.

However, strong objection against these models is their computational infeasibility, which makes doubtful that desirable capabilities can be really achieved in practice. In part of induction, the most direct way to improve their efficiency is to select some appropriate reference machine, which specifies inductive bias agreed with reality [4]. This operation is useful, but insufficient, because it can help to find only limited number of simple regularities in reasonable time, while more complex regularities inevitably encountered in nature will be unrecoverable. More efficient practical approximations of universal intelligence models lose universality (e.g. [5]).

One can possibly conclude that efficient universality cannot be achieved, and universality is mostly of theoretical interest, or inefficient universal methods can work in parallel with efficient pragmatic intelligence as a last resort.

However, universal intelligence can be probably much more useful in practical sense. Principal possibility of automatic construction of efficient narrow methods

B. Goertzel et al. (Eds.): AGI 2014, LNAI 8598, pp. 133–142, 2014.
© Springer International Publishing Switzerland 2014

from inefficient general methods via program specialization (partial evaluation) was noticed 30 years ago [6]. Recently, this idea was put into the context of AGI research [7]. In particular, it was indicated that specializing a universal intelligence w.r.t. some problem and solving this problem afterwards can be computationally more efficient than solving this problem immediately. This result implies that models of universal intelligence can be made more efficient without violating universality. Indirect utilization of universality in the form of construction of specialized methods is attractive since it also bridges the gap between the two approaches, and shows that systems composed of a set of specialized modules are insufficient for AGI, because this set is fixed and is not automatically extended.

However, no analysis of possible specialization of concrete models of universal intelligence has been given yet. In this paper, we make the first such attempt focusing on Solomonoff's universal induction (of course, specialization of decision-making is also of interest). We consider this problem on example of mass induction tasks for which benefits from specialization should be more evident.

2 Background

Universal Induction

For the sake of simplicity we will consider method of universal induction (instead of universal prediction or universal intelligence), which consists in searching for the shortest program that reproduces given data [4]:

$$p^* = \arg\min_p [l(p) \mid U(p) = x], \tag{1}$$

where each program p reproducing data x being executed on universal machine U is treated as the (generative) model of this data, and p^* is the best model.

This is universal solution to the problem of induction, because it possesses two main properties: it relies on the universal (Turing-complete) space of models, in which any computable regularity can be found, and it uses universal complexity-based criterion for model selection (which universality also relies on Turing-completeness since any two universal machines can emulate each other using interpreters with length independent of programs being executed).

This criterion is incomputable, but can be replaced with a computable counterpart (e.g. Levin search, *LSearch* [8], based on Levin complexity instead of Kolmogorov complexity), however required number of operations for identifying p^* will be proportional to $2^{l(p^*)}T(p^*)$, where $T(p^*)$ is required time for p^* to terminate.

This estimation cannot be reduced without violating universality, optimality or without imposing some restrictions. Thus, the question is how to do this in a reasonable way. We cannot build a method that will work better in any environment, but we can build a method that will do much better in environments possessing some specific properties (preserving its universality in other environments). These properties are exploited in narrow methods.

Mass Induction and Data Representations

In order to bridge the gap between efficient narrow methods and inefficient universal methods, one should understand difference between them. Of course, practical non-universal methods usually work with Turing-incomplete model spaces. That is why they can be made computationally efficient. Their success in turn is conditioned by correspondence between a class of data to be processed and a model space. Moreover, does each practical method not only fit a specific class of input data, but it is also applied to different data instances from this class independently meaning that each such method is designed to solve some mass induction problem (a set of individual induction tasks to be solved independently).

Indeed, computer vision systems, for example, are most usually applied to different images (or video sequences) independently, and they rely on specific image representations (e.g. based on Fourier or wavelet transforms, contours, feature points, etc.), which define corresponding model spaces.

Mass induction with introduced representations as a possible connection between universal and narrow methods has already been considered [9]. One can state the following mass induction task. Let the set $\{x_i\}_{i=1}^n$ of input data strings be given. Instead of searching for the shortest program reproducing concatenation $x_1 \ldots x_n$, one can simplify the problem and search for programs S and input strings $y_1 \ldots y_n$ for them such that $U(Sy_i)=x_i$. The program S is called representation for the data set $\{x_i\}$, while y_i are the models obtained within this representation. For the sake of simplicity, we will write $S(y)$ instead of $U(Sy)$, when U is fixed. Criterion for selecting the best representation and models can be derived from Kolmogorov complexity:

$$K_U(x_1 x_2 ... x_n) \approx \min_S \left(l(S) + \sum_{i=1}^n K_U(x_i \mid S) \right),$$

$$S^* = \arg\min_S \left(l(S) + \sum_{i=1}^n l(y_i^*) \right),\ y_i^* = \arg\min_{y:S(y)=x_i} l(y), \tag{2}$$

which is a version of the representational minimum description length principle that extends usual minimum description length principle [9].

Such consideration gives only the criterion for selecting models for data of specific types and to reduce computational complexity in terms of decomposition of the high-dimensional task with data $x_1 \ldots x_n$ into lower-dimensional subtasks with data x_i.

However, each such subtask remains computationally intractable requiring enumeration of $2^{l(y_i^*)}$ models for each x_i. At the same time, it might be not necessary if S is not a universal machine. Actually, in most practical methods observational data x_i are directly mapped to their descriptions y_i by some program S', which can be referred to as a descriptive representation (in contrast to a generative representation S). For example, if x_i are images, then S' can be a filter, which outputs image edges, feature points or something else as y_i.

However, descriptive representations cannot be directly utilized in universal induction. The problem is in criterion. Generative framework assures that data compression

is lossless, and Kolmogorov complexity is the correct measure of amount of information. In descriptive framework, we can calculate lengths of models, but we do not know how much information is lost. Indeed, descriptive models impose constraints on data content instead of telling how to reconstruct data. It can be seen on example of (image) features. Value of each feature contains additional information about data, but in indirect form. Given enough features, data can be reconstructed, but this task is rather complex, if features are non-linear. Thus, it is difficult to tell from nonlinear feature transform itself, whether it is lossless or not. This situation is similar to difference between declarative and imperative knowledge. For example, programs in Prolog impose constraints on solution, but don't describe, how to build it.

In the field of weak AI both representations and methods for searching models within them are constructed manually (or chosen automatically from narrow classes). Artificial general intelligence should have same capabilities as humans, namely, should be able to construct new (efficient) methods by its own. Here, we consider this process as specialization of AGI with respect to specific tasks to be solved.

Program Specialization

The idea of partial evaluation comes from observation that if one has a program with several parameters, and value of one of these parameters is fixed and known, the program can be transformed in such way that will work only with this value, but will do it more efficiently. One simple common example is the procedure of raising x to power n using a cycle, which can directly compute $x * x$ without cycles or iterations, when it is specialized w.r.t. n=2. In general, if there is a specializer *spec* for programs in some programming language R, then the result $spec(p, x_0)$ of specialization of a program $p(x, y) \in R$ w.r.t. its first parameter x=x_0 is the program with one parameter, for which $(\forall y) spec(p, x_0)(y) = p(x_0, y)$.

Most frequent application of partial evaluations is interpreters and compilers. Well-known Futamura-Turchin projections [10] show that if there is an interpreter $intL \in R$ for programs in a language L, then the result of its specialization w.r.t. some program $p_L(x)$, $p_L \in L$, is the program p_L compiled into the language R, since $spec(intL, p_L)$ is the program in R such that $(\forall x) spec(intL, p_L)(x) = intL(p_L, x)$ meaning that the result of execution of this program is the same as the result of interpretation of the program p_L.

Further, since specializer takes two arguments, it in turn can also be specialized w.r.t. interpreter *intL* yielding a compiler *spec*(*spec*, *intL*) from L to R, because $(\forall p_L, x) spec(spec, intL)(p_L)(x) = intL(p_L, x)$. One can further specialize *spec* w.r.t. itself *spec*(*spec*, *spec*), which can be used in particular as a generator of compilers, since $(\forall intL) spec(spec, spec)(intL)$ is the compiler from L to R.

Main interesting property of specialization consists not simply in the condition $(\forall y) spec(p, x_0)(y) = p(x_0, y)$, but in fact that evaluation of a program specialized w.r.t. some parameter can be much faster than evaluation of original program. However, traditional techniques of partial evaluations can guarantee only linear speedup [11], which is enough for compilers, but inappropriate for our tasks. Some metacomputation techniques allow for polynomial and even exponential speedup, but unrelia-

bly [11]. Of course, in general the problem of optimal specialization is algorithmically unsolvable, but humans somehow manage to use it (of course, the principal question is either human intelligence is efficiently general due to powerful specialization, or visa versa).

Specialization of Universal Induction

The most direct application of partial evaluation in mass induction consists in consideration of the generative representation S as the interpreter of some language. Models y_i will be programs in this language. Then, one can directly apply Futamura-Turchin projections. For example, specialization of S w.r.t. y_i will yield compiled programs $spec(S, y_i)$ in the language of the reference machine U. Such "compilation" of models or construction of a compiler $spec(spec, S)$ might have sense, but its effect on computational efficiency will be insignificant. Instead, one should consider specialization of procedures of universal induction.

Consider the following extension of the Levin search procedure for the task of mass induction (universal mass induction method):

Given strings $x_1,\ldots x_n$ enumerate all programs S in parallel (allocating resources for each program proportional to $2^{-l(S)}$); for each program and for each x_i enumerate all possible strings y_i, until the first set $\{S^*, y_1^*,..., y_n^*\}$ is found such that $S^*(y_i^*) = x_i^*$.

We will refer to this algorithm as *RSearch* (representation search). *RSearch* has a subroutine (let refer to it as *MSearch*, model search), which searches for the best y for given S and x: $y=MSearch(S, x)$ such that $S(y)=x$ implying

$$(\forall x)S(MSearch(S,x)) = x\,. \tag{3}$$

MSearch uses exhaustive search to find the best model within the utilized representation. However, the task of searching for the shortest generative model in Turing-incomplete space (that corresponds to a set of possible input strings to some program S) can have simplified or even explicit solution in the form of an algorithm, which directly maps given x to its best model y.

Imagine that we have some specializer *spec*, which can accept the algorithm $MSearch(S, x)$ and any fixed representation S and produce its computationally efficient projection such that $(\forall x)spec(MSearch, S)(x) = MSearch(S, x)$ by definition of *spec*. Let denote $S'=spec(MSearch, S)$, then $(\forall x)S'(x) = MSearch(S, x)$. Substituting this result in (3), one can obtain $(\forall x)S(S'(x)) = x$.

Among all possible programs possessing this property, the program S should be chosen, for which (2) is held. We have proven the following theorem.

Theorem 1. The result of specialization of the model search in the universal mass induction method (*RSearch*) w.r.t. the fixed representation S optimal for the set $\{x_i\}$ of input data is the program S' such that $(\forall x)S(S'(x)) = x$ and $\sum_i l(S'(x_i))$ is minimal.

This theorem shows that S' is the right-inverse to S, but not an arbitrary one, since it should satisfy the information-theoretic optimality criterion.

It should be noted that equality $S'(S(y))=y$ can be true not for all y, since for some representations (e.g. interpreters of universal machines) many models producing the

same x can exist implying that if y is not an optimal model of x=S(y) then $y^* = S'(S(y)) \neq y$. Thus, S' is not necessarily left-inverse.

Theorem 1 shows that a descriptive representation is a result of partial evaluation of (extended) universal induction with some generative representation. One can go further and apply the second Futamura-Turchin projection in these settings, i.e. specializes the *spec* program itself w.r.t. *MSearch* algorithm. The program with the following property will be immediately obtained.

Theorem 2. Let *spec*(*spec*, *MSearch*)=*Inv'* then $(\forall S, x)S(Inv'(S)(x)) = x$.

This theorem might seem trivial, but it shows a connection between inversion and specialization, which are usually considered as different tasks of metacomputation.

Again, it is not true that $(\forall S, y)Inv'(S)(S(y)) = y$, because S is not injection. Also *Inv'* is not an arbitrary inversion, but it constructs *S'*=*Inv'*(*S*) optimal in terms of induction (as it is indicated in Theorem 1).

It is interesting to note that is usually considered as a particular case of metacomputations, which also include program inversion (which is also assumed to be more difficult than specialization). However, Theorem 2 shows that inversion can also be obtained as the result of specialization in case of search procedures.

Of course, one can consider the third Futamura-Turchin projection also, which will be *spec*(*spec*, *spec*) again. In this context, it appears to be not only a generator of compilers, but a generator of procedures for solving inverse problems. This is quite interesting abstract construction, but it is a way too general in terms of what it should be able to do. Indeed, self-application *spec*(*spec*, *spec*) is supposed to be done without knowing, which programs or data will be further passed to *spec*. The third projection should probably be put in online settings (self-optimization of *spec* in runtime while receiving concrete data) to become relevant to AGI.

3 Search for Representations

Inferred theoretical results reveal only general properties of specialized induction procedures, but don't give constructive means for building them. Straightforward way to further this approach is to try applying partial evaluation and metacomputation techniques directly to *MSearch*. Such an attempt can be very valuable, since it can help to understand usefulness of program specialization as possible automated transition from general inefficient to narrow efficient intelligence.

However, in our particular case, the result of specialization *spec*(*MSearch*, *S*) can be unknown together with *S*. Here, we don't solve the problem of efficient automatic construction of representations, but limit ourselves with the problem of efficient model construction within somehow created representations. For this reason, we use the *RSearch* procedure modified in such a way that instead of searching for all y_i for each *S*, it searches for pairs of *S* and *S'* with *S'* satisfying Theorem 1. We will refer to this procedure as *SS'-Search*.

Let us estimate computational complexity of different search procedures. The number of operations in *RSearch* will be proportional to $2^{l(S)}\sum_i 2^{l(y_i)}$. It should be pointed out that in the worst case *LSearch* time will be proportional $2^{l(S)\sum_i l(y_i)} = 2^{l(S)}\prod_i 2^{l(y_i)}$ since no decomposition is done.

SS′-Search time will be proportional to $2^{l(S)}2^{l(S')}$ meaning that *SS′-Search* can even be much more efficient than *RSearch* in cases, when y_i are longer than S' (and this should be quite common in sensory data analysis especially when data contain uncompressible noise). However, the opposite result can also be possible. The main advantage of *SS′-Search* should be in construction of S', which can help to search for models of new data pieces more efficiently.

4 Implementation in Combinatory Logic

We built one resource-bounded implementation of universal induction using combinatory logic (CL) as the reference machine and genetic programming as the search engine [12]. Because it is not biased toward some specific task, it is not practically useful. However, it is appropriate for validating theoretical results. Here, we extend this implementation on the case of mass induction problems using *RSearch* and *SS′-Search*.

We used combinators **K**, **S**, **B**, **b**, **W**, **M**, **J**, **C**, **T** with the following reduction rules

K $x\ y \to x$	**S** $x\ y\ z \to x\ z\ (y\ z)$	**B** $f\ g\ x = f\ (g\ x)$
b $f\ g\ x = g\ (f\ x)$	**W** $x\ y = x\ y\ y$	**M** $x = x\ x$
J $a\ b\ c\ d = a\ b\ (a\ d\ c)$	**C** $f\ x\ y = f\ y\ x$	**T** $x\ y = y\ x$

where x, y, *etc.* are arbitrary CL-expressions.

We supplement CL alphabet with non-combinatory symbols 0 and 1 (and in some experiments with other digits, which are treated simply as different symbols). Reduction of CL-expression can yield some non-reducible expression with combinators or with only non-combinatory symbols.

Representations can be easily introduced in CL. Indeed, one can construct a CL-expression as a concatenation of two expressions S and y treating S as a representation and y as a model of data x within this representation, if concatenation Sy is reduced to x. Then, being given a set of data pieces $\{x_i\}$ in mass induction settings, one should search for one common S and different y_i such that CL-expressions Sy_i are reduced (possibly imprecisely) to x_i.

In order to select the best S and y_i the criterion (2) can be made more concrete:

$$S^* = \arg\min_S \left[H_S\, l(S) + H_y \sum_i l(y_i^*) + H_x \sum_i d(x_i, S(y_i^*)) \right]. \qquad (4)$$

where y_i^* are models obtained by *MSearch* or by applying S' to x_i in *SS′-Search*; $l(S)$ and $l(y_i^*)$ are lengths (number of symbols) in corresponding strings, $d(x_i, S(y_i^*))$ is

the edit distance between two strings (number of symbols to be encoded to transform $S(y_i^*)$ to x_i); H_S, H_y and H_x are number of bits per symbol to be encoded in corresponding domains.

Application of genetic programming here is similar to our previous implementation, in which CL-expressions were represented as trees with typical crossover in the form of sub-tree exchange. The main difference is that solutions in mass induction have more complex structure, and crossover should be performed independently for each part. It also appeared to be useful to allow modifications of only one part of the solution per iteration (population).

Our implementation can be found at https://github.com/aideus/cl-lab

5 Experimental Results

We conducted some experiments with several sets of binary strings. Seemingly simplest set was composed of string 11100101 repeated 10 times as $x_1 \ldots x_{10}$. Obvious optimal generative representation here coincides with the same string 11100101 (if empty strings as models are acceptable) since this CL-expression will always be reduced to itself. However, *RSearch* failed to find this solution. The best found solution appeared to be 1110 for representation (S), and 0101 for models (y_i) for each string. Surely, reduction of Sy_i will yield 11100101, but this solution is not optimal. Why consequent optimization of this solution is difficult for genetic programming? String 11100 as the representation can be obtained from 1110 by single mutation, but this will lead to necessity of rebuilding all models (since Sy_i will become equal to 111000101).

Impracticity of general methods in application to mass induction tasks as a conclusion is trivial. At the same time, *SS′-Search* successfully solved this problem and found S'=**J**(**bMJK**)**T** and S=**W**110010. Here, $S'x_i$ is reduced to y_i=1, and Sy_i is reduced back to x_i=11100101. In contrast to *RSearch*, incremental improvement of solutions is achievable here easier, since modification of S requires consistent modification of only S' instead of many y_i.

Consider other sets of strings. The next set was composed of 16 strings 0101101101010, 0001101001011, 0111111110011, etc. These are random strings starting with 0. *RSearch* failed to find any precise solution. Found representations had such a form as **B**(**BW**(**BWM**)(01)) with corresponding models 10111, 10011, 11010, etc. At the same time, *SS′-Search* found optimal solution – S=0, S'=**CK**, in which S' removes the first bit of the string producing models of data strings (which are difficult to find blindly in *RSearch*), and S adds 0 as the first bit.

The next set contained the following strings 00000000, 00010001, 00100010,… 11111111. Both methods managed to find good solutions (although in 25% runs). RSearch found S=**SSbBBM** and y_i=0000, 0001, … 1111. *SS′-Search* found S=**BBB**(**BM**) and S'=**B**(**SJCK**) such that S' transforms x_i to appropriate y_i by removing a duplicating half of a string, which are then transformed back by S.

We also conducted some tests with an extended alphabet of non-combinatory symbols including 0..9 (which were not interpreted as digits though). One set included

such strings as 159951, 248842, 678876, 589985, 179971, etc. (i.e. strings with mirror symmetry). *RSearch* completely failed on this set, while *SS'-Search* found an optimal solution – *S*=**B**(**S**(**BST**))**M**, *S'*=**JKK**. Of course, a solution with the same representation and models is valid for *RSearch* also, but the search problem in purely generative settings appeared to be too difficult.

Another set contained such strings as 307718, 012232, 689956, 782214, etc. Common regularity for these strings is coincidence of 3rd and 4th symbol. Again, *RSearch* was unsuccessful, while *SS'-Search* found *S*=**KBbW** and *S'*=**BK**, which add and exclude redundant symbol correspondingly.

Our main intention to consider specialization of universal induction was to avoid expensive search for individual models for every new data strings. Once *S'* is constructed, it can be directly applied to construct models in all considered cases. This is the main benefit from using specialization of universal induction. We didn't try to solve the problem of automatic construction of arbitrary representations, but increase in performance of universal methods in solving also this problem is importance.

Of course, capabilities of *SS'-Search* based on uninformed search are quite limited. Indeed, in our experiments it failed to discover many seemingly simple regularities especially in strings of varying length (partially because their representation in combinatory logic can be rather complex). Examples of unsuccessful tests include {1221333, 3331221333, 2233313331333, 22122122122, …}, {00, 11, 000, 111, 0000, …}, {491234, 568485, 278412, 307183, 098710, …}, and others. Thus, this solution is far from efficient universal induction. Nevertheless, comparison of *RSearch* and *SS'-Search* shows that efficiency of universal induction can be considerably increased, and there is a principal way to bridge the gap between efficient and universal methods.

6 Conclusion

We considered universal induction in application to mass problems. Solutions of such problems include representations that capture common regularities in strings, and individual models of these strings. Such methods as LSearch can be directly extended to solve mass problems. However, this leads to direct enumeration of both representations and models. At the same time, model search can be made much more efficient for particular representations as it is done in efficient narrow methods of machine perception and learning.

We studied a possibility to perform specialization of universal induction w.r.t. some representation (reference machine). The result of such specialization should correspond to a descriptive representation that maps inputs into models as efficient as possible. However, the most difficult problem consisting in construction of representations themselves remains.

We proposed the *SS'-Search* method that can be treated as the generalization autoencoders [13] for the Turing-complete space of representations. This method consists in searching for descriptive and generative representations simultaneously. It was implemented using combinatory logic as the reference machine. The *SS'-Search*

appeared to be much more efficient for mass induction tasks than the direct search for generative models for each given string, but it still allows solving induction tasks of rather low complexity. Further research is needed to increase efficiency of universal methods. Also, analysis of specialization of concrete universal intelligence models (in addition to universal induction) is of interest.

Acknowledgements. This work was supported by the Russian Federation President's grant Council (MD-1072.2013.9) and the Ministry of Education and Science of the Russian Federation.

References

1. Hart, D., Goertzel, B.: OpenCog: A Software Framework for Integrative Artificial General Intelligence. In: Frontiers in Artificial Intelligence and Applications, Proc. 1st AGI Conference, vol. 171, pp. 468–472 (2008)
2. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability. Springer (2005)
3. Schmidhuber, J.: Gödel Machines: Fully Self-Referential Optimal Universal Self-improvers. In: Goertzel, B., Pennachin, C. (eds.) Artificial General Intelligence. Cognitive Technologies, pp. 199–226. Springer (2007)
4. Solomonoff, R.: Algorithmic Probability, Heuristic Programming and AGI. In: Baum, E., Hutter, M., Kitzelmann, E. (eds.) Advances in Intelligent Systems Research, Proc. 3rd Conf. on Artificial General Intelligence, vol. 10, pp. 151–157 (2010)
5. Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A Monte-Carlo AIXI Approximation. J. Artificial Intelligence Research 40(1), 95–142 (2011)
6. Kahn, K.: Partial Evaluation, Programming Methodology, and Artificial Intelligence. AI Magazine 5(1), 53–57 (1984)
7. Khudobakhshov, V.: Metacomputations and Program-based Knowledge Representation. In: Kühnberger, K.-U., Rudolph, S., Wang, P. (eds.) AGI 2013. LNCS (LNAI), vol. 7999, pp. 70–77. Springer, Heidelberg (2013)
8. Levin, L.A.: Universal sequential search problems. Problems of Information Transmission 9(3), 265–266 (1973)
9. Potapov, A., Rodionov, S.: Extending Universal Intelligence Models with Formal Notion of Representation. In: Bach, J., Goertzel, B., Iklé, M. (eds.) AGI 2012. LNCS (LNAI), vol. 7716, pp. 242–251. Springer, Heidelberg (2012)
10. Futamura, Y.: Partial Evaluation of Computation Process – an Approach to a Compiler-Compiler. Systems, Computers, Controls 2(5), 45–50 (1971)
11. Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Prentice-Hall (1993)
12. Potapov, A., Rodionov, S.: Universal Induction with Varying Sets of Combinators. In: Kühnberger, K.-U., Rudolph, S., Wang, P. (eds.) AGI 2013. LNCS, vol. 7999, pp. 88–97. Springer, Heidelberg (2013)
13. Hochreiter, S., Schmidhuber, J.: Nonlinear ICA through Low-Complexity Autoencoders. In: Proc. IEEE Int'l Symp. on Circuits and Systems, vol. 5, pp. 53–56 (1999)