Machines with Autonomy & General Intelligence: Which Methodology?

Kristinn R. Thórisson
School of Computer Science, Reykjavik University
Icelandic Institute for Intelligent Machines
Iceland
thorisson@gmail.com

Abstract

From its early beginnings artificial intelligence (AI) targeted the ambitious goal of creating machines with high levels of autonomy and intelligence. A major determinant of scientific and engineering progress is the methodology employed. Instead of being specifically designed for this goal, the main methodologies employed so far in AI have closely followed those of general software development: Algorithms are developed for particular (pre-defined) tasks in particular (pre-defined) conditions. At the center are programming languages designed for humans to construct software architectures by hand, who implements systems line by line, like a construction worker laying down bricks. This constructionist approach has produced a diverse set of isolated AI solutions to relatively small problems, the generally intelligent machine still being a distant dream. We argue that going beyond current systems and realize AI's original and ambitious goal requires a different kind of approach, one that specifically takes account of the nature of the system to be built. Here we look at arguments for this claim, identify some features of general intelligence likely to dictate features of such a methdology, and discuss what an appropriate methodology may look like.

1 Introduction

Methodology and research tools are a key determinant of the rate of scientific progress: The microscope impacted significantly the rate of progress in the study of the microscopic; gene sequencing equipment radically sped up the rate of progress in the investigation of genomes, and the spatio-temporal resolution of fMRI equipment has had a profound effect on rate of progress in brain research, to take some examples. Over the course of history these were not singular events, never to be seen again: Improvements in methodology regularly transform research and progress in all fields of scientific endeavor. Methodology, with its concrete and conceptual tools, is no less important for progress in AI than in other fields of science and engineering. If we want to build a system with *general intelligence*—a property of cognition directly related to a system's propensity for *autonomy*—it behooves us to choose our methodology carefully. Considerations of methodologies in turn requires us to take a close look at our background assumptions of the phenomenon we want to re-create artificially.

If the history of AI methodology teaches us anything, surely one thing must be that hanging our hat on a single *task* as *the* candidate for elucidating the central principles of general intelligence is a terrible idea. We are talking about e.g. the hypothesis, central in the early days of AI and still prevalent to day in various guises, that any machine capable of beating a world champion in chess would *necessarily* possess *general intelligence*. It wasn't just partly wrong, or somewhat off target: It could in fact not have been more wrong.[1] Granted, when the hypothesis was fielded researchers did not have the 50 years of history we now have for telling them otherwise. A bit of skepticism, built on this fact – and the limited progress towards general AI in the past 60 years – seems a healthy approach to take today.

To be *autonomous* means to be independent from outside forces and influences. *Self-adaptation* is the process of a system to change its own operation, behavior, or structure, to better achieve its own goals in a particular environment. Here we assume that an agent's *drives* (top-level goals) are provided from the outside, just like evolution has imbued living entities with survival and reproduction drives, by a designer that ensures that her artifacts serve the purpose they were created for. In highly autonomous systems very little

[1] After Deep Blue defeated Gary Kasparov in 1997 the the developers of the technology at IBM struggled for years trying to adapt the technology and knowledge gained on the project to other tasks, projects, systems, and fields—to virtually no avail [Stork 1998].

other knowledge is needed—just a small "seed" to bootstrap the system's knowledge acquisition, also called "learning".[2]

A general learner with self-adaptation capabilities will be *more autonomous* the more *diverse tasks* it can learn, the more skills it can bring to bear on a task at any single point in time, and the more *diverse environments* and it can handle. Its generality may also be determined by how quickly it can learn and adapt to changes of various sizes and kinds, and the diversity of *novel* and *unforeseen* tasks and environments it can handle. A third way in which generality can be measured is by a system's ability to handle underspecification of task and environment.

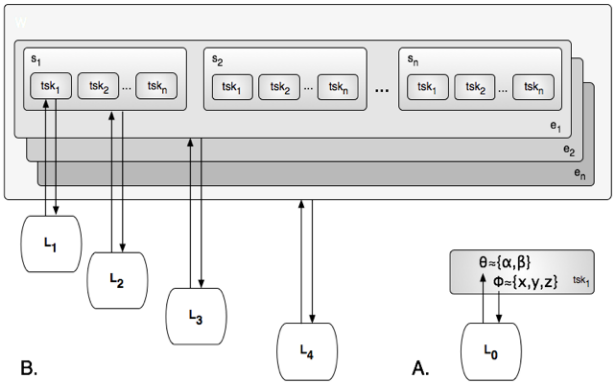

FIG. 1. A: Simple machine learners (L_0) take a small set of inputs (x, y, z) and make a choice between a set of possible outputs (α, β), as specified by a system's designer. B: More complex learners (L_1, … L_4) handle a number of tasks (tsk_1, … tsk_n) in many environments (e_1, … e_n). *See text for details.*

Current state-of-the-art learners come with severe limitations compared to the kind of learning found in nature (FIG. 1, L_0). For instance, increasing the set of inputs, or increasing the number of required outputs, will either break a learning algorithms or slow learning down to impractical levels. When taught something different from what they already know, they will take equally long (or longer!) to learn the new thing (because they cannot bring prior experience to bear on a new task), and will in fact forget everything they knew before.

In reference to FIG. 1, let task tsk_i refer to relatively non-trivial tasks, e.g. assembling furniture and moving office items from one room to another, simple learner L_0 is limited to only a fraction of the various things that must be learned to achieve such a task. Being able to handle a single such task in a specific type of situation (S_1) with features that were unknown prior to the system's deployment, L_1 is already more capable than most if not all automatic learning systems available today. L_2, L_3 and L_4 take successive steps up the complexity ladder beyond that, being able to learn *numerous* complex tasks (L_2), in *various situations* (L_3), and in a wider *range of environments* and mission spaces (L_4). Only towards the higher end of this ladder can we hope to approach truly general intelligence—systems capable of learning to effectively and efficiently perform *multiple a-priori unfamiliar* tasks, in multiple *a-priori unfamiliar situations*, in *multiple a-priori unfamiliar* environments, *on their own.*

While artificial learning machines have existed for decades, their methodological and theoretical foundations still limit them to a handful of input and output parameters, typically on what we would in general parlance call a *single pre-defined task* in a *well-defined*, *unchanging* environment. Current machine learning algorithms are also a prime demonstration of what is called *negative transfer of training*—and close to a worst-case incarnation of that kind.

Researchers targeting generality must not only overcome this fundamental limitation of modern machine learning, they must go beyond it and address *learning of many tasks*, by systems *situated* in *information-rich* environments, in which simultaneous and continuous acquisition of relevant knowledge, and contemporaneous knowledge of multiple skills which can be transferred to new tasks, may be the primary or even only option for successful adaptation. To advance the state of the art on autonomy and generality we should ask "What kinds of control mechanisms can support *that kind* of adaptation across *multiple novel* tasks, situations, and domains, *autonomously*?" This is of course a central question in the field of AI, and clearly one that will not be answered in a short paper. The immediate follow-on question, and the central topic of this paper, is:

> *What kinds of methodologies can help us develop such systems?*

We will look at this question from four different but related perspectives, all leading to a similar conclusion: Present AI methodologies, all of which we argue can be categorized as *constructionist methods* [Thórisson 2012], fall short of being a sufficient and necessary framework for addressing the requirements relevant to creating machines with high levels of general intelligence and autonomy.

The rest of the paper is organized as follows: After analyzing the limitations of constructionist methodologies (section 2) we look at the operational characteristics of adaptive systems that may be important for the task at hand (section 3), followed by dissecting how constraints on the system we seek to build put requirements on the methodology (section 4). In section 5 we look at two defining properties of higher natural intelligences: introspection, or *reflection*, and *cognitive growth*. Lastly (section 6) we pull together some main themes from the preceding sections, and finish (section 7) by drawing conclusions and discussing future work.

2 Which Methodology: Limitations of the Constructionist Approach

Artificial intelligence researchers use methodologies that are fundamentally the same as those used in computer science and software engineering in general—the same programming tools, same hardware, same operating systems, same

[2] We could also use other measures, e.g. the level of dependency of a developing system on particular environmental circumstances for its bootstraping or survival; while important, this issue is out of the scope of this paper.

"algorithmic thinking", and the same software development techniques. Just like in any other software development efforts, the approach puts the AI researcher in the role of a "construction worker", building systems by hand. Virtually all available methodologies for AI are of this *constructionist* kind—the system programmer laying down each instruction like a construction worker lays down the brick to a house.

2.1 Constructionist Approaches in a Nutshell

Constructionist approaches lead system development along the path familiar to all software developers: A *task* and its execution environment is identified and dissected (by humans), solutions are proposed (by humans), subsequently algorithms are developed (by humans), and the resulting system behavior evaluated (by humans)—all by hand.[3]

Building a vacuum-cleaning robot,[4] for instance, will proceed by a system designer (a) identifying all relevant aspects of target rooms and floors, (b) anticipating the variability to be expected in these with future customers, and (c) defining all major and minor aspects of the system's operations to achieve its vacuum-cleaning task, acceptably accurately and reliably. However implemented, the system receives pre-defined high-level pragmatic rules – heuristics – to decide how to achieve what typically is a single operational goal, designed and written by humans, and a relatively detailed specification of its operating environment from its designers is "baked into" its design. The interaction between the rules may be complex, even unforeseeable, and thus the ultimate behavior of the system may be difficult or impossible to predict in detail, at least without running the system in some set of scenarios intended to evaluate its actual behavior under various conditions.

These systems are typically not written to come up with their own methods for *achieving* their goals. Likewise, they are incapable of *coming up* with their own (sub-)goals. Nevertheless, making these systems more autonomous would requires precisely such capabilities. Such systems match the old cliché very nicely: They only do what the programmer programmed them to do.

We can of course augment such hand-crafted systems with modern machine-learning techniques, and this will typically proceed by the designer identifying which parameters shall be handled by machine learning algorithms (cf. FIG. 1). The input and ouptut variables of the leaner will be hand-picked by the designer, and the training will be directed by hand by the designer. All in all, while some limited aspects of the system may now have been automatically programmed, the addition of machine learning will not fundamentally change the methdology otlined above: All system goals and subgoals are identified and specified by the designers and baked into the design, and the correctness of the system's behavior will be verified by the system's designers. The overall system design still follows a constructionist methodology.

2.2 Some Limitations of Current Systems

To see if this methodology is suited for the kinds of systems we target – with increased levels of intelligence – we must look at what kinds of processes are missing. One such activity is the ability of the system to not only to be able classify its inputs, but rather the more general ability of being able to *learn* to classify them, in light of its top-level goals (drives).

To take some examples, a thermostat's method for achieving its task is selected by the designer, and operationalized explicitly in its design. The input variable (temperature) is hand-selected, and its range (maximum, minimum meaningful value) is also decided by the designer and made implicit in the device's design. Granted, a thermostat is typically not considered "intelligent". Note, however, that the exact same way of construction is used for expert systems, which are generally considered AI systems and often cited as a major milesone for AI of the past four decades. These systems have in common that they take input, e.g. heat (in the case of thermostat) or a description of patient symptoms (in the case of medical diagnosis), and process this without concern for its *relevance* to their own task, as this is guaranteed by the system's users, as is also the guarantee that the input be *noise-free* – not containing irrelevant data. These aspects are shared by virtually all modern AI systems. Because they are thus hard-wired to do a sigle task, and their input is *fixed* rather rigorously beforehand, the input's *meaning* is decided for them. This means that the relation of the input to what they do with it, and the behavior they are capable of producing, is *implicit* in their design from the outset. It should be clear then that such systems cannot become truly autonomous: Since they cannot *learn to classify* inputs (after they leave the lab), they cannot handle diversity, and thus cannot become general. This limitation is *inherent in them* as a result of the inherent properties of *the methodology* used to develop them.

A learning system using modern artificial neural networks (ANN) to do the heavy lifting is in some ways a bit more capable than the above systems, but only slightly so. ANN-based systems still have their task handed to them by their designer, their input variables defined beforehand, including their operating ranges. They can only learn a single task, whether it will be identifying letters on number plates of automobiles, finding human faces in photographs, or adjusting the autofocus on your digital camera in realtime, because they have *no mechanism for separating out that which is new and that which is old*, as already mentioned in the discussion of classification above.

Another important limitation of modern machine learners, and one which affects their potential for generality and autonomy, is that after they are built and deployed their learning must be turned off. This is because their design prevents them from changing in predictable ways after they leave the lab: "In the wild" they *change unpredictably*. Thus, currently no system exists that can handle input that is

[3] Granted, several tools may be used, but these generally don't go beyond what may justifiably be called "semi-automatic hand-operated tools".

[4] These will be the same whether the system is physical and operates in the physical world, or virtual and operates in some more abstract information environment.

unexpected or unforeseen (by the system's designer), as handling such input would require autonomous learning. Current systems must therefore "call home" when facing circumstances outside of their strictly defined operating ranges—i.e. re-design by the system's engineers.

2.3 Semantics and Autonomy

The problem with constructionist methodologies lies in *forced semantics:* The meaning of the data to be imbued to the system is provided directly by the system's designers when the systems are designed. Current AI systems cannot handle the unforeseen or unexpected because the methods encourage a task-driven design, where task-specific features are transferred directly into software via human-designed huristics. What is needed are methods that help the designers move to the next level up, by supporting and encouraging designs where mechanisms for *system-created operational semantics* are sought instead. Put more succinctly, constructionist methods are strictly *allonomic* – they rely on the semantics and intelligence of the designer, instead of allowing the system itself to generate its own meaning. An autonomous and general system is independent of particular tasks, and achieving such task independence calls for the system to be provided with the *mechanisms* that allowed the human designer himself to come up with task-specific heuristics in the first place: The system itself should be rigged so as, when given a specification of one or more tasks, to come up with useful heuristics on its own.

3 Which Methodology: Operational Characteristics of Adaptive Systems

Let's now look a bit closer at the features that we seek for an autonomous and general artificial intelligence [Thórisson & Nivel, 2009]. Adaptation involves the ability to achieve high-level goals in spite of obstacles presented by the environment. Such obstacles come in many forms, but generally they represent a mismatch between the system's knowledge needed for achieving its goals and the way the world operates or is currently structured. An intelligent system must be able to (a) assess the world it operates in, (b) realize ways of achieving its goals, (c) produce the effects on the environment that achieve those goals, and (d) evaluate this effect in light of its goal(s).[5] We can say that for an agent $\boldsymbol{A}$ with knowledge $\boldsymbol{k}$, in situation $\varkappa$, seeking goal $\boldsymbol{g}$ in $\varkappa$, an obstacle is a state of $\varkappa$ that prevents $\boldsymbol{A}$ from directly applying $\boldsymbol{k}$ to change (some subset of) $\varkappa$ to (subset) $\boldsymbol{g}$. To know whether k will achieve $\boldsymbol{g}$, $\boldsymbol{A}$ must have knowledge of the relevance of $\boldsymbol{k}$ to $\varkappa$, and the ability to apply it to $\varkappa$ to achieve $\boldsymbol{g}$, as well as to assess whether and when $\boldsymbol{g}$ has been achieved.

3.1 Learning for Generality

Concerning capabilities that set systems whose intelligence is *general* apart from whose intelligence is not, let's look at four already alluded to above.

1. Although it is given in this context, we will mention first the fundamental ability to *learn*. This requirement excludes all expert systems and similarly "hard-wired" systems (e.g. chess programs of the 90s), as well as many others touted as milestones in the field. It is no surprise that these are not considered "general", but perhaps somewhat surprising the high regard in which many have held them.

2. Second, the learning must be *always on*, most notably *after the system leaves the lab* – after the designer releases the system for good into its target operating environment. This excludes all artificial neural networks, since these have to be frozen at release time due to the unpredictability in their behavior that otherwise would arise. While any kind of learning is better than no learning, the most useful learning is the one accompanied by the capacity for abstraction, what has been called *induction*—the ability to generalize from experience [Wang 2006].

Now is a good time to remind ourselves that the kinds of systems we are interested in should be able to operate with *insufficient knowledge*—in fact, the ability to learn is not necessary if we have complete knowledge already [Wang 2006]. The same holds for induction with respect to incomplete knowledge: Induction is most useful when combined with an ability to hypothesize causal links between newly discovered phenomena.

3. So, third, a necessary function for generality is the ability to accept inputs with little or no prior familiarity to the system, identify patterns in the input, model their relation, and generalize from it. It could be argued, in fact, that this is a central mechanism of *any* learning system embodying lifelong continuous – or *cumulative* – learning, since the same mechanism is needed to separate irrelevant patterns from relevant ones, and to separate old and valid classifications of patterns from new ones, so as not to unlearn what has already been learned (avoiding negative transfer of training).

4. Fourth, armed with these capabilities, the system must (a) be capable of modifying its models of the environment, which due to incomplete knowledge will have unavoidable conflicts and errors, based on the *quality of their usefulness* for achieving goals and predicting how the environment works, through a feedback loop outside the system. This, in effect, is the *learning by experience* mechanism, which is necessary for all systems capable of adapting to environments not known at design time.

5. Fifth and lastly, a mechanism that will greatly affect a system's adaptability its *capacity for cognitive growth*—of adapting its adaptation mechanisms. In other words, its ability to *learn to learn*. A general-but-mostly-static learning

[5] Biological and robotic systems assess the environment via some form of vision, touch, and hearing, the adaptation is done via learning, thinking, and meta-cognition, and the behavior is produced via controlled, targeted actions of a body. Disembodied intelligences – e.g. an intelligent software system that "lives in the cloud" – while not being in control of an actual physical body or having physical perception mechanisms such as cameras and microphones, will also consist of these functional structures, otherwise it would not be an independent system separate from its environment. Here we will assume that such an explicit separation between the system and its environment exists.

mechanism may be sufficient for a certain set of environments with a particular range of complexity. But an ability to develop *new learning methods* would allow a system to develop improvements or even brand-new learning mechanisms, and to adapt "from scratch" to a wider range of environments. It would enable an intelligent system to develop and subsequently choose the most appropriate learning mechanism for the environment and task at hand at any time. This can likely increase a system's learning and adaptation abilities by orders of magnitude. A prerequisite for this to be possible is a system's propensity for pattern classification: Without a sufficient level of input *classification fidelity* an appropriate learning mechanism (or any other cognitive mechanisms for that matter) cannot be correctly chosen in each circumstance. (Another requirement is that the system must be able to turn this classification capability onto its own control architecture, and equally importantly, that it can program/re-program itself. We will get back to this in more detail below.)

3.2 An Illustrative Example

To put this more succinctly, given environment $\boldsymbol{e}$, in a current state $\boldsymbol{s_1}$ ($\boldsymbol{e_{s1}}$), and agent $\boldsymbol{A}$ with (compound) goal $\boldsymbol{g_1}$ to bring about state $\boldsymbol{s_2}$ using method $\boldsymbol{m_1}$, let's say behavior $\boldsymbol{b_1}$ could satisfy $\boldsymbol{m_1}$ and bring $\boldsymbol{e}$ directly from the current state $\boldsymbol{s_1}$ to $\boldsymbol{s_2}$ ($\boldsymbol{e_{s1}} \rightarrow \boldsymbol{e_{s2}}$) to achieve $\boldsymbol{g_1}$. In this example the agent has never seen $\boldsymbol{e_{s1}}$, however, and does not contain $\boldsymbol{b_1}$ in its memory. Thus, from identifying $\boldsymbol{e_{s1}}$ $\boldsymbol{A}$ cannot produce $\boldsymbol{b_1}$ immediately or directly. This calls for *adaptation behavior*, which requires sub-goal(s) to be produced that entail any, some, or all of the following covert and overt operations on part of $\boldsymbol{A}$: **1**. Reasoning about how to achieve $\boldsymbol{s_2}$, resulting in any, some, or all of the following operations; **2**. Experimenting on the world to obtain $\boldsymbol{b_1}$ or an alternative suitable behavior; **3**. Analyzing $\boldsymbol{g_1}$ to see how much variation in $\boldsymbol{s_2}$ and $\boldsymbol{m_1}$ will still count as $\boldsymbol{g_1}$ having been achieved; **4**. Modifying $\boldsymbol{g_1}$, or producing (a set of) alternative goal(s) which result in an acceptable modifications of, or alternatives to, $\boldsymbol{m_1}$ and/or $\boldsymbol{s_2}$; **5**. Evaluating through reasoning and/or experimentation on the world whether $\boldsymbol{g_1}$ is well formulated in its present form, leading to 4 or 6; **6**. Reconsidering $\boldsymbol{g_1}$, leading to 4 or to abandoning $\boldsymbol{g_1}$ altogether.

Adaptive behavior 3 requires $\boldsymbol{g_1}$ to have associated with it some range of acceptable deviations, or being amenable to computation of such ranges. While adaptation behaviors 3, 4, 5 and 6 are typically not seen, strictly speaking, as "achieving one's goal(s)", they frequently happen in the real world, and it should be obvious that any learning human being will occasionally abandon goals, as well as selecting some in favor of others. Note that *such considerations are only relevant for an agent with multiple goals situated in an environment with noticeable uncertainty*. In this kind of adaptation, evaluation of the *cost of* modification, abandonment, and acceptable deviations, are an important – and most likely a *necessary* – part of the learning.

3.4 Conclusion from Operational Characteristics

We can only conclude, again, that the machine learning we would need for a machine with general intelligence and high levels of autonomy are not well served by a methodology that keeps system design close to one or more particular tasks. Current popular programming languages used in AI (C++, Python, LISP, etc.) seem a rather poor match to help us develop the meta-skills, such has learning to learn, reflection (see below), and flexible pattern matchings, needed to move to higher levels of intelligence and autonomy.

4 Which Methodology: The Argument from Constraints

Traditional software systems' architectures consist of a set of software units, or modules, which in turn run on a particular hardware implementation, whose joint runtime behavior defines the system's capabilities. A software architecture contains thus, by definition, several mechanisms that interact to produce a system's overt and covert behaviors. What we call *control architecture* is the software and hardware concerned with managing the long-term holistic behavior of the system; hardware must be included in that because no computation or other real-world effect can be realized without a physical embodiment. Natural intelligent systems, such as humans, are capable of exhibiting a variety of complex overtly observable behaviors, most of which are the result of years and decades of learned adaptation. Some part of this variety stems from fundamental principles of the underlying cognitive architecture, and some of it are merely side-effects of nature's specific implementation(s).

The covert behaviors of the various kinds of control architectures that can bring about intelligent behavior have, however, turned out to be very difficult to reverse-engineer. While cognitive scientists are dedicated to reverse engineer the *particular* mechanisms of *natural intelligences*, artificial intelligence is more focused on the *general principles* of that *class* of mechanisms. Nevertheless, both groups are in effect looking to reverse-engineer the special natural phenomenon we call "intelligence". This reverse-engineering effort proceeds through the application of particular methodologies, whose features tend to be heavily guided by the background assumptions. And since intelligence is a natural phenomenon[6] we should use the phenomenon in question to help us pick the best methdology. One way is by scrutinizing the *constraints* that that phenomenon must satisfy.

[6] If intelligence were a mathematical phenomenon we could proceed in our AI research as we do with other mathematical phenomena—through sound mathematical methods. This not being the case, however, means that such methods are fundamentally inappropriate as a main approach—experimentation is the obvious choice for any experimental phenomenon; baring that possibility, as for instane in astrophysics, software modeling and simulation is the second most appropriate. How difficult it turns out to characterize and understand intelligence mathematically remains unclear; as of today, finding "the solution" to intelligence solely through mathematics is about as likely as it would have been for the Wright brothers build a flying machine by studying mathematics. As it turned out, airodynamics was developed much later, by reverse-engineering machines already airborne [Thórisson 2013].

4.1 Understanding Constraints

For any system $\boldsymbol{S}_x$ that we may want to model, and which is subject to *few* constraints, a large set of possible implementations can produce the observed behaviors of $\boldsymbol{S}_x$; practically speaking we will have quite a bit of freedom in how we capture the system in our re-creation – a large set of model variants could be conceived of that replicate the targeted features of $\boldsymbol{S}_x$ equally well. If it is rolling behavior we want to model, a round rock, a melon or an orange all suffice to replicate the rolling behavior in a slope or when pushed: Reverse-engineering "rolling behavior" of an object means discovering that its design must approximate a circle. As the number of constraints that $\boldsymbol{S}_x$ must meet is increased, a decreasing set of model variations could possibly reproduce its capabilities. The steering mechanism on automobiles has for instance numerous possible implementations, some better than others, but all capable of making the car turn. It took centuries for the current mechanisms with this purpose to be developed, and they are based on many inventions and innovations, most of which are necessary to achieve the full behavioral range of modern automobiles. Because the list of constraints on a modern steering mechanisms includes a lot more than simply "rolling", the set of possible designs is heavily *limited* by these constraints.

Now, if we want to uncover the set of control architectures capable of producing the kind of adaptive behavior we see in natural intelligences we must identify the key principles that enable these architectures to produce their results. This is not as simple as it sounds, in particular because these principles must be inferred from opaque covert behaviors that, while produced by the control system, are only a tiny fraction of the full repertoire of what that – or any – cognitive system found in nature are capable of.

4.2 Constraints on Systemic Behavior

A biological cognitive system $\boldsymbol{A}$ is capable of achieving goals through a wide range of covert behaviors $\boldsymbol{b}$ of which some $\boldsymbol{b}_n \subset \boldsymbol{B}_A$ will be observed to be produced in a task-environment $\boldsymbol{e}_n \subset \boldsymbol{E}$, such that:

$\boldsymbol{b}_1$ are produced in $\boldsymbol{e}_1$
$\boldsymbol{b}_2$ are produced in $\boldsymbol{e}_2$
…
$\boldsymbol{b}_n$ are produced in $\boldsymbol{e}_n$

and this full set being the behavioral repertoire of $\boldsymbol{A}$: $R_B(\boldsymbol{A})$. Now, $\boldsymbol{b}_1$ is picked by a system designer as an example behavior that his artificial system A' should be able to produce in a simplified environment $\boldsymbol{e'}_1 \subset \boldsymbol{E}$; let's say $\boldsymbol{b}_1$ is e.g. winning games of blitz chess (the behavior), $\boldsymbol{e'}_1$ is blitz chess games (the task-environment consisting of the rules of chess plus time constraints), and $\boldsymbol{E}$ is a set of board games. Scientists and engineers now venture to build A' using current (constructionist) methodologies, using $\boldsymbol{b}_1$ as a test case to measure the evolving design against. Their hope is that the system thus built, and the principles and mechanisms that enable A' to produce $\boldsymbol{b}_1$ in $\boldsymbol{e'}_1$, will replicate some important *fundamental principles* behind the full system capable of $R_B(\boldsymbol{A})$, and turn out to be general enough to allow A' to produce a larger set from $R_B(\boldsymbol{A})$ than just the example behaviors $\boldsymbol{b}_1$ (preferably the full repertoire $R_B(\boldsymbol{A})$).

Even though A' may ultimately be able to produce $\boldsymbol{b}_1$ in $\boldsymbol{e'}_1$ ($R_B(\boldsymbol{A}) = \boldsymbol{b}_1|\boldsymbol{e'}_1$), assuming the constraints that $\boldsymbol{e'}_1$ brings on the behavioral repertoire are from those of $\boldsymbol{e'}_2$, $\boldsymbol{e'}_3$, ... $\boldsymbol{e'}_n$, A' is unlikely to be capable of $\boldsymbol{b}_2$, $\boldsymbol{b}_3$, and $\boldsymbol{b}_4$, not to mention the full behavioral repertoire of $\boldsymbol{A}$, because (a) the constraints of $\boldsymbol{e}_2$, $\boldsymbol{e}_3$, etc. *were never considered as design targets* for A', and (b) neither were any of the $\boldsymbol{b}_2$, $\boldsymbol{b}_3$, etc. As the complexity[7] of the targeted task-environment increases from which the example task is drawn, the larger the number of constraints on the target system brought by it. Given the complexity of tasks and environments that general (human-level) intelligence is capable of addressing, we must assume that the number of constraints it brings on the target system – general intelligence – is rather large. The resulting artificial system A' thus built is therefore much more likely to be limited to $\boldsymbol{e'}_1$ than to generalize beyond it, especially for task-environments more complex than chess.[8] A system built to play e.g. blitz chess is more likely to be strictly limited to chess than to be extendable to Backgammon and Go, as the rules of these games were never considered by the system's designers. Using this methdoology, counter to the the original hope of going beyond the example task, the machine is stuck *within the confines of the original constraints chosen for its design*.

What we see here is a limitation of this kind of approach as *a method of systems research:* While reverse-engineering ultimately requires us to actually build our model to see if it can produce the full set of behaviors we hoped for, this only works *if the correct constraints* are brought into the engineering effort *from the outset*. This is because each additional $\boldsymbol{e}_n$ requires the system to be capable of producing an increasingly larger yet more specific set of behaviors, including the meta-ability to differentate between $\boldsymbol{e}_n$s. Considering a wider range of behaviors, tasks and environments pushes the design considerations to include a wider scope than a single example task can provide. A system that meets a large set of constraints in its behaviors (i.e. each $\boldsymbol{e}_n$ requiring the system to behave in specific ways and not others, thus imparting constraints on the system) has fewer degrees of freedom—fewer ways of realization—than another that must meet fewer constraints. Bringing (too) few constraints to bear on the development effort, from among the

[7] By "complexity" we mean something like "intricacy", such that when the intricacy of the task-environment increases, the set of behaviors that can achieve the goals of that task-environment is reduced, relative to any and all behaviors that could be expressed in that task-environment but do *not* achieve the goal(s). In the vernacular this means "the task gets harder". In our discussion complex task-environments are also harder to formalize.

[8] It is of course not categorically impossible that, through pure luck, such an approach *could* produce a system that generalizes beyond the inital task. However, with an increasing number of constraints the probability of such luck grows quickly and asymptotically towards zero.

myriad of tasks natural intelligences are observed to be capable of, such as building "intelligent" systems that target a small isolated task, is likely to completely leave out systemic features that are *absolutely necessary* for producing the capabilities of natural cognition.

Considering too few constraints opens up the design possibilities and reduces the chance of finding the "golden key" (if it exists); on the other hand, considering a large set of constraints complicates things, and often makes the research exceedingly more difficult. Increased effort and closer attention to *how we pick our constraints* will help us end up with the necessary and sufficient constraints that allow us to build truly general systems.

4.2 Two Lines of Inquiry

Our starting point for exploring what kinds of control architectures can bring about intelligent behavior must include two lines of inquiry. First, we need to try – using any relevant method available – to hone in on which observable behaviors and features are more likely than others to serve as good ("the right") constraints to steer design and reverse-engineering efforts to the small subset of architectures capable of producing thems. Second, and in parallel, we need to hone in on the key candidate architectural principles that enable such intelligent behavior to be produced, through any sensible means possible (including using the observable behaviors hypothesized to help with that effort).

The combination of these two lines of inquiry brings about a new insight into what our methodology really must look like: Yes, we need to identify a set of behavioral features and characteristics $\boldsymbol{B} = \boldsymbol{a}, \boldsymbol{b}, \ldots \boldsymbol{n}$ sufficiently representative of the ultimate target range of behaviors aimed for. We must ensure that this subset of target behaviors is as wide as possible, because we want to use these to hone in on the kinds of characteristics are critical for higher-level cognitive control (e.g. human-level intelligence). The obvious way to do this is by observing natural intelligence in action: Constraints for our reverse-engineering efforts *must come from the phenomenon* that we wish to understand and (selectively) imitate. We must not forget that in AI research the tasks that natural intelligences are observed to be capable of are *not* our target—our target is the *general architectural principles* hidden inside the animal's skull that enable this.

Selecting any single behavior $\boldsymbol{b}_n$ may be tempting, and seem like a good strategy—if we could only find a good representative task that promises to help us hone in on the architectural principles of cognition. But to do this would be a mistake – unless – we take extra steps to ensure that that subset is highly likely to contain behaviors and capabilities that help us *constrain the set of potential architectural mechanisms* underlying the intelligent system's key principles of operation. This is, of course, just as difficult as it sounds. Some key candidates for this role have, however, already been mentioned: Operating under assumptions of continuous external time (world clock), learning – continuously and cumulatively – new tasks in new environments. Other characteristics include operation and adaptation in spite of incomplete knowledge, insufficient time, and limited computational power.

But identifying a promising set of behaviors that helps us constrain the possible architectural principles is not enough. We must, at the same time, propose and evaluate *practically implementable mechanisms* that could, in the right combinations, bring about the kind of computations that might produce the range of representative behaviors thus identified. One stands out as being especially interesting in this respect: self-inspection, also called *reflection*.

5 Which Methodology: Cognitive Growth

We now turn to our last topic in answering the question "which methodology?": Reflection and cognitive growth.

The ability to deal with new and unforeseen tasks, environments, and situations, is central to the notion of intelligence. As we have discussed, to achieve such generality it is not sufficient for a system to be pre-programmed for particular task, environment, or domain – more general principles must be involved. One feature of natural intelligence that seems likely to matter for broad generality is cognitive growth—the active development of the cognitive mechanisms themselves, as the system adapts and learns to deal with a new tasks, circumstances, or environments. Cognitive growth calls for the system programming/re-programming parts of itself, which in turn calls for some form of *reflection*—the ability of the system to inspect and evaluate itself, to assess the development course it is on, to reason about which ways are more likely to put it on a desired path of growth than others, and so on. If we want an autonomous system, such self-programming must be largely self-organizing to enable a system to adapt to challenges unforeseen by the system's designer. Sure, every system needs of course some fundamental basic principles of operation that are protected from self-re-programming, otherwise the system's course of cognitive development would be completely unpredictable.

In nature, a system capable of cognitive growth must ensure that some core cognitive control mechanisms are stable enough to guarantee survival of the species, if not the individual. In artificial systems this principle of a protected core is valid as well, but has a slightly different purpose: The core control principles of engineered systems are there to enable the system's designers to guarantee predictability, at some given level of abstraction, of the system's behavior throughout its lifetime.

Central to our discussion here is the hypothesis that cognitive growth cannot occur without some transparency of a system's *operational semantics* [Thórisson & Nivel, 2009] [Thórisson 2012]. Such transparency allows the system itself to inspect its development to e.g. correctly bootstrap its own growth, and re-program itself throughout its lifetime for new tasks, scenarios, and environments. A system with transparent operational semantics has *one* of the *necessary conditions* to re-program itself; such a system is in principle reflective-prepared. To leverage the reflectivity it must also have control of the necessary processes to make use of it.

Another necessary feature to support cognitive growth is *auto-catalytic knowledge acquisition*—the ability of the system to acquire needed knowledge autonomously; *auto-catalysis* and *reflectivity* are thus necessary architectural principles for supporting cognitive growth. Thirdly, such systems cannot rely on a system designer to tweak them in light of unexpected tasks, situations, or environments, meaning that the system must be *endogenous* – free from direct re-programming from the outside. These are important enough for us refer to them with an acronym, *AER*.

FIG. 2. The Constructivist AI Methodology (CAIM) puts the nature of self-constructive systems in the foreground, helping the system designer to think about intelligence less like a hand-crafted artificial machine (top) and more like a complex growing, developing, self-organizing system like a forest (middle) [Thórisson, 2012]. This helps elucidate what kinds of tools are appropriate for the task (bottom).

Humans are most certainly capable of adapting to new tasks, situations, and environments, and of course we should take inspiration from nature, as natural intelligence, in the very least human intelligence, meets the requirement of being auto-catalytic, reflective, and endogenous, and cognitive growth is a fact of human learning and mental development. We humans have an information network to help us bootstrap our cognitive operation – referred to as parents, school, society, etc. Without these, especially the first one, human intelligence is unlikely to bootstrap itself successfully.

6 Self-Constructive Cumulative Learners

We have now reviewed some arguments for why current methodologies, which we can characterize more or less as constructionist, are insufficient to develop truly atuonomous general learning machines. The preceding sections all lead to the same conclusion: That a defining feature of generality and autonomy is the ability for an intelligent system to manage – control – its environment, itself, and its own cognitive growth: A system's general learning mechanism – if it really *is* general – should be applicable to the system's own cognitive system. To build such systems requires a different methodology than what has been most widely used to date. The preceding pages have also addressed aspects that provide a foundation for a new methodology. In light of the AER requirements to achieve continuous cumulative learning, domain-independence, self-construction, and autonomy, we seek a methodology that can help us think more along the lines of *growth* and *development* than manual construction. To paint a caricature, we want to think of autonomous cumulative learners not as machines that must be built brick by brick, bolt by bolt, and line by line, but like a garden or forest whose growth must be nurtured—a methodology for *self-constructive* systems (FIG. 2).

A process of learning entails that as a system $\boldsymbol{S}$ becomes better at achieving a goal $\boldsymbol{g}$ in one or more task-environments $\{\boldsymbol{e}_1 \ldots \boldsymbol{e}_n \mid \boldsymbol{e} \in \boldsymbol{E}\}$ trough experience, where "better" is measured in some relevant way such as the appropriateness of a proposed or executed solution, the speed at which it is produced or executed, and/or its quality. With experience the system may be able to achieve $\boldsymbol{g}$, and over time it achieves it increasingly reliably through particular mental operations, through available sensors and actuators, which we will call tasks $\boldsymbol{T}_1$. If tasks $\boldsymbol{T}_1$ are performed only using its basic/atomic control methods – those provided to it by its designers at the outset – it is likely that by improving the control structures themselves – adapting them to the particulars of $\boldsymbol{T}_1$ – would improve performance on tasks $\boldsymbol{T}_1$. It is also possible that $\boldsymbol{S}$ may produce some modification of $\boldsymbol{T}$ that, given some modification of some set of overt and covert control mechanisms, $\boldsymbol{C}_S$, would improve performance on $\boldsymbol{T}_1$, or potentially enable a set of new tasks $\boldsymbol{T}_2$ that achieve $\boldsymbol{g}$ even better according to the chosen evaluation criteria. Adapting $\boldsymbol{C}_S$ would mean *producing a new or modified set of control mechanisms* that the system was not provided with from the outset, whose form and function is based on the system's experience, and may partly or fully replace the existing $\boldsymbol{C}_S$. This requires some form of self-programming [Nivel & Thórisson, 2009, Thórisson et al. 2012].

Self-programming, in the sense of a cognitive system re-designing and re-implementing parts or full sets of its own cognitive functions, and especially repeated such operations – recursive self-improvement – requires *models of self* [Steunebrink et al. 2016]. As a controller of controllers, such self-programming calls for a *meta-controller* whose target modeling task is the system itself in its domain. The same

rules apply to the meta-controller as to the lower-level controller: It must obtain a model (or models) of its domain and use it to propose, test, and implement new (and/or modified) cognitive functions that partially or fully replace those that were used before. In addition, should the environment change, such modifications need to be accompanied by methods for the system to identify when situations call for a reverting or switching between alternative ways of operating, for instance if the task or environment changes in ways that make the new methods less effective or inappropriate for the kinds of bootstrapping that may need to happen to handle new situations.

7 Conclusions & Future Work

It would seem clear that even if only half of the topics discussed in this paper were relevant and important for achieving general intelligence in a machine, a thorough revision of current methodologies offered by computer science and AI would be in order. Together, the arguments strongly favor a different approach to the one presented by all standard software development methodologies, as these take a human-centric approach where the human designer is responsible for turning a task definition into task-specific algorithms, with little or no serious consideration for bringing autonomy to the system being designed. As a methodology for standard software projects this makes perfect sense; as a methodology for developing autonomous generally intelligent machines this presents a serious mismatch between the target system and the approach to its design.

We have done some work along the lines of considering what a more appropriate methodology might lok like; the Constructivist AI Methodology (CAIM) targets the *nature of general intelligence* and *autonomy*, with an emphasis on self-construction through cumulative learning based on modeling causal relations. We propose a set of assumptions – which, just like the central thesis of AER – are scientific refutable hypotheses. Details on these can be found in [Thórisson 2012], [Nivel et al. 2013], [Nivel et al., 2014a] and [Nivel et al., 2014b]. While this takes some steps in the right direction, plenty of work is neede on which methodologies can help us get more quickly the results we are interested in: machines with autonomy and general intelligence.

Acknowledgments

This work was sponsored in part by the School of Computer Science at Reykjavik University and by a IIIM Centers of Excellence Grant from the Science & Technology Policy Council of Iceland.

References

[Bongard et al. 2006] J. Bongard, V. Zykov & H. Lipson. Resilient Machines Through Continuous Self-Modeling. *Science,* **314**(5802):1118-1121, 2006.

[Cully et al. 2015] A. Cully, J. Clune, D. Tarapore & J-B. Mouret: Robots that can adapt like animals. *Nature,* **521**(7553):503-507.

[Nivel & Thórisson 2009] E. Nivel & K. R. Thórisson. Self-Programming: Operationalizing Autonomy. *Proc. 2nd Intl. Conf. on Artificial General Intelligence*, 150-155, 2009.

[Nivel et al. 2013] E. Nivel, K. R. Thórisson, B. R. Steunebrink, H. Dindo, G. Pezzulo, M. Rodríguez, C. Hernández, D. Ognibene, J. Schmidhuber, R. Sanz, H. P. Helgason, & A. Chella, & G. Jonsson. Bounded Recursive Self-Improvement. Reykjavik University School of Computer Science Technical Report, RUTR-SCS13006. arXiv:1312.6764 [cs.AI]

[Nivel et al. 2014a] E. Nivel, K. R. Thórisson, B. R. Steunebrink, H. Dindo, G. Pezzulo, M. Rodríguez, C. Hernández, D. Ognibene, J. Schmidhuber, R. Sanz, H. P. Helgason, & A. Chella. Bounded Seed-AGI. *Proc. 7th Intl. Conf. Artificial General Intelligence*, 85-96, 2014.

[Nivel et al. 2014b] E. Nivel, K. R. Thórisson, B. R. Steunebrink, H. Dindo, G. Pezzulo, M. Rodríguez, C. Hernández, D. Ognibene, J. Schmidhuber, R. Sanz, H. P. Helgason, A. Chella & G. Jonsson. Autonomous Acquisition of Natural Language. *Proc. IADIS International Conference on Intelligent Systems & Agents 2014*, 58-66, 2014.

[Stork 1998] David G. Stork (ed.). *HAL's Legacy: 2001's Computer as Dream and Reality*. Cambridge, MA: MIT Press. 1998.

[Steunebrink et al. 2016] Bas R. Steunebrink, Kristinn R. Thórisson & Jürgen Schmidhuber. Growing Recursive Self-Improvers. *Proc. 9th Intl. Conf. Artificial General Intelligence (AGI-16)*, 129-139, 2016.

[Thórisson & Nivel 2009] K. R. Thórisson & E. Nivel. Holistic Intelligence: Transversal Skills and Current Methodologies. *Proc. 2nd Intl. Conf. Artificial General Intelligence*, 220-221, 2009.

[Thórisson 2012] K. R. Thórisson. A New Constructivist AI: From Manual Construction to Self-Constructive Systems. In P. Wang and B. Goertzel (eds.), *Theoretical Foundations of Artificial General Intelligence*. Atlantis Thinking Machines, **4**:145-171, 2012.

[Thórisson et al. 2012] K. R. Thórisson, E. Nivel, R. Sanz & P. Wang. Approaches & Assumptions of Self-Programming in Achieving Artificial General Intelligence. Editorial, *Journal of Artificial General Intelligence Special Issue, Self-Programming*, **3**(3):1-10, 2012.

[Thórisson 2013] K. R. Thórisson. Reductio ad Absurdum: On Oversimplification in Computer Science and its Pernicious Effect on Artificial Intelligence Research. *Proc. Workshop Formalizing Mechanisms for Artificial General Intelligence & Cognition*, 31-35. Institute of Cognitive Science, Osnabrück, 2013.

[Wang, 2006] P. Wang. *Rigid Flexibility:The Logic of Intelligence*. Springer, 2006.