Resource-Bounded Machines are Motivated to be Effective, Efficient, and Curious*

Bas R. Steunebrink[1], Jan Koutník[1], Kristinn R. Thórisson[2], Eric Nivel[2], and Jürgen Schmidhuber[1]

[1] The Swiss AI Lab IDSIA, USI & SUPSI
[2] Reykjavik University

Abstract. Resource-boundedness must be carefully considered when designing and implementing artificial general intelligence (AGI) algorithms and architectures that have to deal with the *real world*. But not only must resources be represented explicitly throughout its design, an agent must also take into account their usage and the associated costs during reasoning and acting. Moreover, the agent must be intrinsically motivated to become progressively better at utilizing resources. This drive then naturally leads to effectiveness, efficiency, and curiosity. We propose a practical operational framework that explicitly takes into account resource constraints: activities are organized to maximally utilize an agent's bounded resources as well as the availability of a teacher, and to drive the agent to become progressively better at utilizing its resources. We show how an existing AGI architecture called AERA can function inside this framework. In short, the capability of AERA to perform self-compilation can be used to motivate the system to not only accumulate knowledge and skills faster, but also to achieve goals using less resources, becoming progressively more effective and efficient.

1 Introduction

Real-world intelligent systems are bounded by resources, which are consumed during operation but are available only in finite amounts. However, (working) definitions of intelligence do not always stipulate that agents necessarily have to cope with insufficient knowledge and resources [4], leading to different views on which models of behavior can pass as "intelligent."

Many theoretical models of learning and intelligent behavior do not take into account resource constraints, or come with proofs of optimality that only work in the limit, usually of time. But because of the inherently limited nature of resources, such proofs say little about practical worth. For example, Q-Learning has been proved to converge to the optimal policy (under certain assumptions) [15], but there is no real-time limit on how long such convergence will take, i.e., another non-optimal method may produce a better solution before a certain fixed deadline. In fact, when certain RL methods are combined with nonlinear function approximators, the convergence guarantee no longer holds or becomes an open question, yet they work better (and thus more intelligently?) on many practical tasks.

* This research was funded by the EU projects HumanObs (FP7-ICT-231453), IM-CLeVeR (FP7-ICT-231722), and Nascence (FP7-ICT-317662), and by SNF grant #200020-138219.

K.-U. Kühnberger, S. Rudolph, and P. Wang (Eds.): AGI 2013, LNAI 7999, pp. 119–129, 2013.

Table 1. The naming scheme. The leftmost column lists the fundamental resources that constrain real-world agents. The compression (or usage minimization) of each of these resources leads to the properties listed in the second column. Resource-bounded agents should be driven to attain these properties, but only for the drive to learn there exists an appropriate term: curiosity. Unfortunately the English language contains no suitable single words to describe the drive to be energy-efficient or the drive to be time-effective, but they are important drives nonetheless. The rightmost column lists the activities that can be performed to obtain more of the resources listed in the leftmost column. The Work–Play–Dream framework is described in section 5.

Resource	Compression	Drive	To obtain more
Energy	Efficiency	—	Work
Input	Learning	Curiosity	Play
Time	Effectiveness	—	Dream

As another example, AIXI is claimed to model the most intelligent decision making possible [2], yet its algorithm is incomputable and thus AIXI is incapable of producing any decisions at all in this universe. Even its computable variant AIXItl suffers from an exponential factor, which prevents it from making timely decisions in interesting real-world settings, where available computational power is limited. Our criticism here is rooted in the fact that the definition of intelligence behind AIXI, as detailed in [5], does not mention resource constraints. In this paper we follow Wang's working definition [13], which does so.

We argue that an intelligent system that has to function in the same, complex real world in which humans operate, must abide by the following principles *by design*:

1. explicitly acknowledge that there are resource constraints,
2. identify which are those constrained resources (section 2),
3. strive to minimize their usage / maximize their utilization (sections 3 and 4), and
4. explicitly take them into account during all activities, including reasoning and acting, and order activities around their utilization (section 5).

Having just done point 1, we go into each of the other principles in the referenced sections. Finally, an actual system that can do all four points is presented in section 6.

2 Fundamental Resources

Real-world intelligent systems rely on the following, constrained resources:

Energy: To perform work, a machine needs to expend energy. Many "wall-plugged" machines need not worry about their electricity consumption, but autonomously roaming battery-operated machines do. Material resources, including memory space, are strictly speaking a manifestation of energy, but if desired they can be counted separately in certain practical settings.

Input: The set of input data since the first activation of the machine is often referred to as the *history of observations*. For succinctness we will just use the term *input*. It can also be seen as a flow of information from outside the machine through particular sensors to a particular part inside the machine (namely memory, or some reactive module).

Input is a resource because the real world cannot be observed entirely and instantly, so the machine must choose which limited set of observations to pay attention to in the present and what observations to try and elicit in the future [1].

Time: All activities take time, even idling. We insist that time is a resource originating from the external environment, meaning that the passage of time must always be measured using an external clock, and not by counting e.g. internal processor cycles. This means that the speed of the computational architecture does matter, and likewise slowdowns in algorithms that are normally hidden when discussing them using the big-O notation also do matter.

See also table 1, which summarizes the terminology used in this paper.

It should be noted that all resources are related: sensing, reasoning, and acting all consume energy, input, and time. In fact, the consumption of energy and time is inversely related in many situations: a task can be completed faster by consuming more energy, or energy can be conserved by taking it slower. There are exceptions, however, such as a robot trying to swim through a dilatant fluid (e.g., cornstarch mixed with water). But we must realize that energy and time cannot always be exchanged—certain deadlines cannot be met no matter how much energy is available.

3 Resource Compression: What and Why

Let us first introduce the following terminology:

- A machine that is compressing its energy usage is said to be *efficient*.
- A machine that is compressing its input is said to be *learning*.
- A machine that is compressing its time usage is said to be *effective*.

It seems obvious that the usage of resources must be minimized, but let us consider the most important reasons nonetheless. Compression of both energy and time usage is beneficial because getting tasks done more efficiently and faster often results in higher (external) rewards. Moreover, spending less energy and time on current tasks leaves more time and energy for future tasks, since they are finite resources. Energy and time usage compression is thus beneficial for ensuring survival and longevity. Another way to benefit from efficiency and effectiveness is that they leave energy and time for (curiosity-driven) exploration, which provides an opportunity to perform more learning. It should be noted that energy and time usage may have to be traded off, i.e., there may be a Pareto front of efficiency and effectiveness. In many practical settings some resources will be more *precious* than others. For example, time is generally more precious than memory space,[1] meaning that if a machine can work substantially faster by using a bit more memory, it should be motivated to do so.

The compression of input is beneficial because the alternatives are either to delete all observations—which leads to a purely reactive agent—or to store everything, which

[1] Memory / storage space can be seen as a particular manifestation of energy.

is infeasible for resource-bounded machines.[2] Even if storing all observations were feasible, it is unnecessary because the real world is assumed not to be random, but to exhibit certain regularities. In fact, compression of data is equivalent to learning, which is essential for any intelligent system. In the context of resource utilization, learning is important because it can lead to the knowledge necessary for compressing energy and time consumption on future tasks.

Let us now say that a machine's behavior is *acceptable* if its resource usage stays within certain thresholds specified per resource, but it is *better* to use less resources. If a machine is pursuing a certain goal, it is said to have *failed* as soon as a threshold is crossed for any of its resource usages. For example, a deadline can be specified for a goal by setting a threshold on the maximum amount of time that may elapse. Or a budget can be specified for a goal by setting a threshold for the maximum amount of energy that can be spent. Because all resources are ultimately limited, and because we have a lot of work to be done by machines (with their resources also being shared by humans), each machine should be driven not just to perform acceptably, but to perform better and better. Therefore becoming better—in the technical sense defined above—should be implemented as an intrinsic motivation for machines.

4 Driven by Resource Compression Progress

There are two ways to minimize resource consumption: through (1) *knowledge* and (2) *architecture*. Knowledge allows a machine to select efficient and effective actions given what it has learned so far about the external environment and its own operation. Although all intelligent agent architectures must encode the system's knowledge and skills in some way, more powerful architectures allow knowledge and skills to be *re-encoded* to maximize efficiency and effectiveness. In the following two subsections, both methods for achieving resource compression are discussed, respectively.

4.1 The Urge to Learn: Curiosity

Artificial curiosity [9,8] urges an agent to learn new phenomena through better compression of the growing history of observations (i.e., *Input*). An internal reward for "interestingness" is defined to be proportional to the first derivative of the negative number of bits over time needed by an agent's compressor to describe a piece of input. Curiosity is then the drive to select actions that maximize the expected future interestingness.

The principle of curiosity assumes that there are regularities in both the environment and in the tasks that need to be performed in it. Such an assumption is definitely reasonable among real-world environments and most of the simulated ones. An environment responding in completely random ways makes any intelligent system hopeless as the best agent strategy would be a random one too. It is also assumed the environment is not too adversarial, or that danger can be avoided through "parental" supervision.

[2] Although advances in storage techniques may leave room to store all observations in a database, it must be realized that there will probably also be advances in sensor resolution, so it may turn out that the rate at which sensors can generate data grows too fast to store everything after all.

Curiosity is a way for an agent to "fill in gaps" in its knowledge, in anticipation of unexpected events and tasks for which it wants to be better prepared. Although it may seem impossible to prepare for the unknown, it is the assumption of regularity that makes curiosity a reasonable exploration strategy, because then it can be expected that regularities discovered earlier have a chance of being useful for solving future tasks.

Although in certain special cases it may be possible to perform exploration while trying to solve a task (which could be called creative problem solving; *cf.* [10,11]), there will usually be both resource and situational[3] constraints preventing such simultaneous balancing of exploration and exploitation from being feasible.

4.2 The Urge to be Effective and Efficient

Knowledge about the environment and skills obtained through curiosity can be exploited to achieve known and novel tasks using less resources, notably energy and time. However, simply *knowing* how to perform a task more effectively and efficiently does not necessarily make an agent *capable* of doing so. For example, consider a person learning to tie shoelaces for the first time. He is explained the routine and then manages to reproduce a proper knot, but taking ten seconds to do so. To bring the task execution time down to, say, three seconds, the apprentice does not need more knowledge—just more training. What happens in humans as a result of such training? The representation of the skill of tying shoelaces is moved from a high-level system (here probably the motor cortex) and *re-encoded* in a low-level system, where the skill is consolidated as what is known as muscle memory. The result is that the initial, costly, "cognitive" routine for tying shoelaces can now be executed by a fast, automatized routine.

An analogy with computers can be made here by contrasting interpreted code with compiled code. An interpreter typically runs code presented in a high-level form and can provide additional services such as extensive debugging tools. In contrast, compiled code consists of processor-specific instructions which are executed at the lowest level. Save for costly reverse engineering techniques, compiled code is a good as a black box. It should be noted that the analogy does not extend to training e.g. an artificial neural network: although in a sense the encoding of knowledge is changed during training, activating a trained neural network will still work in the same way as activating an untrained one—just the results are better. But with compilation, the meaning of a routine remains exactly the same; it can just be executed faster in compiled form.

Very few intelligent agent architectures have the ability to perform skill-preserving compression of parts of themselves [12]. For example, Soar's chunking allows fast retrieval of a previously successful solution [3], but it does not re-encode the solution if it turns out to be reliable over time. In fact, to date no architecture known to the authors is capable of selective self-compilation, except AERA, discussed in section 6.

Let us emphasize again the two ways to better utilize energy and time on a particular task: (1) find a *new* sequence of actions that solves the task more efficiently and effectively (example: find a smoother reaching trajectory to grab a cup) and (2) take a *fixed* sequence of actions known to solve the task but perform them more efficiently or

[3] An exploratory action may put the agent in a position that conflicts with the position that it needs to be in to achieve the current goal; i.e., an agent cannot be in two places at the same time.

faster (example: run the solution after compilation or on a faster processor). The former requires knowledge acquisition and can thus be found through curiosity; however, the latter requires an agent architecture allowing skills to be represented at different levels. Here "levels" includes both software (e.g., interpreted versus compiled code) and hardware (e.g., run on processors with different clock speeds, store information in memory with different read and write speeds). It should be noted though that many kinds of actuators, such as servomotors, have a maximum operational speed and cannot work faster even when more energy is available. This can impose an upper limit on the speed of a routine which has to control actuators. Nevertheless, many systems are considerably slower in selecting actions than it takes to execute them—a classic example is Shakey the robot, a modern example the Mars rovers—leaving room for progress in efficiency and effectiveness through re-encoding of skills.

So an advantage can be gained by architectures that support (at least) two levels of encoding skills. Having this capability, an agent should be motivated to compress energy and time usage, meaning that it should be motivated to re-encode skills at a lower level. Some care must be taken here though: as re-encoding itself can be a costly process, and because it can result in black-box components, it should only be done for skills which have been shown to be useful and reliable. Here we see that curiosity is a process that can be in service of the drive to be effective and efficient: curiosity can drive the exploration of current uncertain knowledge, which can lead to either its falsification or vindication. Falsified knowledge can be deleted; vindicated knowledge can be re-encoded at a moment when there is enough time to do so. As always, time must explicitly be traded off; a principled way of doing so is presented in the next section.

5 An Operational Framework for Resource-Bounded Machines

During their initial stages of learning about the world and their constraints, machines will inevitably interact with humans—their teachers, supervisors, and users—and will need to adapt to our patterns. These patterns include time for work, time for supervised[4] play, and time for deactivation. An intelligent, resource-bounded machine should utilize the kinds of activities afforded by these patterns to the fullest extend. Specifically, we associate a mode of operation with each of these three patterns of activity, namely Work, Play, and Dream—together the WPD framework:

Work: Every machine has a purpose, so when a user provides it with goals, it will have to fulfil them. During work, all resources of the machine are focused on exploiting the current knowledge and skills. Interesting, unexpected events may be recorded in memory buffers for later analysis (see Dream mode). Any spare processing power may be used for learning from these events, but no time will be allocated for exploration.

Play: At times when the machine is operational but has no goals left to achieve or does not know how to achieve the existing ones, it will activate its play drive. The machine will perform curiosity-driven exploration, constantly challenging itself to do novel yet learnable things. But as curiosity can "kill the cat," the machine will have to be supervised (e.g., a human remains close to a delicate robot's kill switch), just as small

[4] "Supervised" as in parental safeguarding against danger, not as in classifying labeled data.

children should not be left on their own. Even machines operating only in virtual environments require supervision, because there are typically many avenues for exploration that are unrelated to the machine's ultimate purpose. So ideally, the user is capable of interrupting and suggesting more constructive tasks (which would temporarily put the machine in work mode) if she notices the machine exploring such unrelated avenues.

Dream: At times when supervision of the machine is not possible—for example, when the user goes out for lunch or to bed—the machine will have to be disconnected in order to prevent possible damage to either itself or the environment. During such times, the machine becomes a purely computational device without access to the external environment. This time can be put to good use by "dreaming": flushing buffers that contain interesting but unprocessed events observed during work or play, by incorporating them into the machine's knowledge and skill representation. This process has an aspect of real dreaming in that recent events are being re-examined. Furthermore, other computationally intensive tasks such as compression or self-compilation can be performed, to decrease memory usage and speed up skills. If there is still more time left after flushing and compressing—i.e., the machine has not yet been "woken up"—it can start inventing tasks or goals to pursue during play time on the next "day."

We emphasize that Work, Play, and Dream should not be considered as different states, but as different processes which must be scheduled. They may run in parallel, although we expect that there are typically not enough resources available to Work, Play, and Dream simultaneously—notably computational resources (focusing on work at hand does not leave enough power and time to invent novel, intrinsically motivated tasks and perform compression of knowledge and skills) and situated resources (cannot perform work-related and exploratory actions at the same time because they conflict spatially, while there is a deadline to complete work).[5] It may turn out that the need to dream is a necessary side-effect for any system that is constrained in both computational power (because there are more events/inputs than can be processed without forsaking other high-priority tasks) and memory (because unprocessed events/inputs can be stored in buffers but these will fill up over time). Such a system becomes "tired" when its buffers reach capacity, and needs to "dream" to process the input history backlog.

It can even be argued that the combination of work and play gives rise to *creativity*, which is the production of an artifact or action that is both novel/surprising and useful/valuable [7]. The external reward received through work is a measure of the usefulness/value of what the machine produces, while the internal reward received through play allows for inventing novel solutions. The dream mode supports both work and play by dealing with realistic resource constraints: both the computational power and storage capacity of the creative machine are finite, and also the time a supervisor/teacher can spend per day with the machine is limited. Altogether, the WPD framework provides the necessary and sufficient ingredients for "raising" creative machines.

[5] Copying an agent's software to outsource Play and Dream to a separate computer raises huge issues with synchronization—and eventually also ethics.

6 AERA: An Explicitly Resource-Bounded Architecture

AERA is an architecture for AGI systems that has been developed in the European HUMANOBS project, with the aim of learning socio-communicative skills by observing people. The name stands for Auto-catalytic Endogenous Reflective Architecture, which, in a nutshell, refers to the fact that the system is both operationally and semantically closed, as discussed in more detail elsewhere [6].

A functional prototype has been developed as a proof of concept of the architecture, and the preliminary results (yet unpublished) strongly indicate that the architecture is sound and that our approach towards the engineering of autonomous systems is tractable and promising. But implementation and testing details are outside the scope of this paper—here we describe a few features of AERA relevant to the present discussion of resource-bounded machines, showing how the concepts of curiosity, efficiency, and effectiveness can be mapped onto and operationalized in an actual AGI system.

6.1 Principles of AERA

In the main, AERA is designed to reason and achieve goals in dynamic open-ended environments with insufficient knowledge and limited resources—in that we adopt the working definition of intelligence proposed by Wang [14]. Knowledge is understood to be *operationally constructive*: it is executable code, which implements the system's ability to act in the application domain. AERA is model-based and model-driven, meaning that its architecture is unified and consists essentially of dynamic hierarchies of *models* that capture knowledge in an executable form. From this perspective, *learning* translates into building models, integrating them into the existing hierarchies, and revising them continuously. In this way the system controls the expansion and reprogramming of its own code, to keep itself adapted to the environment.

Whereas learning is based on model building, this in turn is driven by goals and predictions, i.e., the evaluation by the system of the observed phenomenology of the domain. In other words, the system infers what it shall do (specification) and observes ways to reach these goals (implementation). Learned specifications and implementations are highly context-dependent, which raises the challenge of identifying when to reuse (or refrain from reusing) learned models. Specifications and implementations are built hierarchically, which means that the system can reuse previously learned skills.

6.2 Executable Models to Represent Knowledge and Skills

The simplified view of a model is as an implication $A \rightarrow B$. A model is a bidirectional program, where both sides can match data: (1) if an A is observed in the input stream, a prediction that B will be observed is generated; and (2) if the system has a B as a goal, an A will be generated as subgoal. *Simulations* can be made using forward chaining, and *planning* can be done using backward chaining. A combination of planning and simulation is performed to check for potential conflicts before committing to a goal.

This is a greatly simplified view though; A and B are actually patterns that can contain variables and guards. Constraints between the variables in A and B and their relative times of occurrence can be defined. Models are also arranged in hierarchies,

meaning that A and/or B can be a reference to another model, such that models can specify contexts and requirements for other models, which are appropriately taken into account during chaining. At the bottom of the hierarchy, A can refer to an actuator command. Predictions generated by models are tracked and used to update the confidence values associated with each model. New models are constructed automatically (i.e., learned) in response to prediction failures and unpredicted goal achievements and state changes. Poorly performing models are temporarily disabled (if there is merely a context change) or deleted (if not functioning well in any context).

Originally conceived for scalability reasons, AERA runs a process that tries to identify and compile chains of models that are useful and reliable.[6] Note that due to regularities in the real world, certain subgoals need to be achieved often, using similar plans, which are represented by backward chained models. Similarly, certain inputs are observed often, leading to similar predictions of next events, which are represented by forward chains of models. Now, the computation performed by a chain of models (forward or backward) can be represented in (C++) code, which can be generated, compiled, and linked back as a black-box DLL to AERA's executable (also implemented in C++) on the fly. The old, white-box models can then be swapped to disk, where they can be retrieved if decompilation is ever needed, e.g., when the compiled "supermodel" starts misbehaving and needs refinement. Thus, besides learning knowledge and skills, AERA is capable of *re-encoding* them in the sense of section 4.2.

6.3 Driven to Self-compilation

Let us consider three resources under the control of an AERA-based machine: real-time (external clock), working memory space (RAM), and storage space (HDD). The latter two are instances of the *energy* resource, but here it makes sense to separate them. Let us furthermore assume that real-time is more precious than working memory space, which in turn is more precious than storage space. (Note the following relations: decreasing the amount of information held in working memory typically speeds up the processing of what remains in working memory—this just reinforces the ordering of preciousness.)

Consider AERA's self-compilation capability: it decreases both real-time and working memory usage, while increasing storage space usage (by swapping the uncompiled models to disk). When weighed by preciousness, self-compilation leads to better resource utilization, notably making the system more *effective*. Thus an AERA-based machine must be motivated to adopt goals that lead to paths of actions that eventually lead to self-compilation. Such goals can be found through a simple analysis of control values already present in models, such as the derivative of their confidence values—a high absolute derivative indicates an unstable model that can be vindicated or falsified by empirical testing, which can for example be done by adopting the left-hand side of such a model as a goal. Such testing then amounts to intrinsically motivated exploration.

But the crux is that actually finding a way of scheduling the mentioned processes is a challenge in itself for a resource-bounded machine: iterating over models to find candidates for experimentation and compilation takes time, and so does planning and simulating to see which candidates can be pursued without conflicts—in time, space,

[6] How exactly usefulness and reliability are determined is out of scope here.

and resulting state—with extrinsic goals and constraints. Here the presented WPD framework offers a principled way of scheduling extrinsic goal achievement (a Work activity), intrinsic goal generation (Dream), experimentation (Play), and self-compilation (Dream).

7 Discussion and Conclusion

We built AERA as a cognitive architecture towards AGI, based on many firm principles [12]. However, curiosity was not one of them. Now it turns out that AERA is missing an exploration heuristic, but that all ingredients are present for implementing Schmidhuber's mathematically sound and biologically plausible definition of curiosity. In fact, generating goals that are expected to activate the existing self-compilation mechanism leads to both forms of input compression identified in section 4: more accurate knowledge, and the re-encoding thereof. We expect this to lead to both faster learning and faster achieving of goals.

We have not twisted AERA to fit the theory of curiosity, nor twisted curiosity to fit AERA. Instead we have searched for a principled middle ground—which we have found in the concept of resource usage compression. Finding this middle ground benefits both AERA and the Formal Theory of Curiosity: it benefits AERA because we strive to base it on solid principles, not to hack a few ideas together; it benefits curiosity because it makes us consider real-world applications and the issues that they bring, which provides an opportunity to refine the theory. Here we have generalized the idea behind the Formal Theory of Curiosity to also take into account other resources, notably energy and time (*cf.* [8,11] for preliminary steps in this direction), paving the way for the construction of machines that are driven to become more efficient and effective.

References

1. Helgason, H.P., Nivel, E., Thórisson, K.R.: On attention mechanisms for AGI architectures: A design proposal. In: Bach, J., Goertzel, B., Iklé, M. (eds.) AGI 2012. LNCS, vol. 7716, pp. 89–98. Springer, Heidelberg (2012)
2. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin (2004) (On J. Schmidhuber's SNF grant 20-61847)
3. Laird, J.E.: Extending the soar cognitive architecture. In: Proceedings of the First Conference on Artificial General Intelligence. Springer, Memphis (2008)
4. Legg, S., Hutter, M.: A collection of definitions of intelligence. In: Proceedings of the First Annual Artificial General Intelligence Workshop (2006)
5. Legg, S., Hutter, M.: A formal measure of machine intelligence. In: Proceedings of the Annual Machine Learning Conference of Belgium and The Netherlands, Benelearn (2006)
6. Nivel, E., Thórisson, K.R.: Self-programming: Operationalizing autonomy. In: Proceedings of the Second Conference on Artificial General Intelligence (2009)
7. Runco, M.A., Jaeger, G.J.: The standard definition of creativity. Creativity Research Journal 24, 92–96 (2012)
8. Schmidhuber, J.: Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. Connection Science 18(2), 173–187 (2006)
9. Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990-2010). IEEE Transactions on Autonomous Mental Development 2(3), 230–247 (2010)

10. Schmidhuber, J.: POWERPLAY: Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem. In: Frontiers in Cognitive Science (in press, 2013) (Preprint (2011): arXiv:1112.5309 [cs.AI])
11. Srivastava, R.K., Steunebrink, B.R., Schmidhuber, J.: First experiments with PowerPlay. Neural Networks (2013)
12. Thórisson, K.R., Helgasson, H.P.: Cognitive architectures and autonomy: A comparative review. Journal of Artificial General Intelligence 3(2) (2012)
13. Wang, P.: On the working definition of intelligence. Tech. rep. (1995)
14. Wang, P.: Rigid Flexibility: The Logic of Intelligence. Springer (2006)
15. Watkins, C., Dayan, P.: Q-learning. Machine Learning 8(3/4), 279–292 (1992)