# On the Use of Randomization for System of Systems (SoS) Design of Intelligent Machines

Stuart Rubin[1] and Gordon K. Lee[2]

[1]SPAWAR Systems Center
53560 Hull Street
San Diego, CA USA
stuart.rubin@navy.mil

[2]Dept. of Electrical & Computer Engineering
San Diego State University
5500 Campanile Drive
San Diego, CA 92182-1326

## ABSTRACT

This paper takes a System of Systems (SoS) approach to the realization of machine intelligence. Random differences are learned by the system, generalized, and made available for subsequent replay in design transformations. It is empirically demonstrated that cross-domain symmetries can play a major role in design generation in particular and in the design of SoSs in general. The fundamental theory of randomization is the science, which underpins the practice. The approach is illustrated by an example of the design of a refrigeration system.

***Keywords:*** *Heuristics, KASER, Randomization, Symmetry, System of Systems (SoS)*

## 1. INTRODUCTION

Intelligence is generally defined as the ability to complete a task or achieve a goal in an uncertain environment. Hence, some characteristics of intelligence include adaptability, capability for self-optimization based upon some goal or goals, ability for performing self-diagnostics and self-maintenance, and the ability to learn and reason. Currently, there are many systems which can play chess, perform character/image recognition, have decision-making capabilities, and do control/compensation. These systems may be characterized as having some form of intelligence. The idea of intelligence has been applied to robotics and automation to enhance the applicability to a variety of problems.

Some systems try to mimic human intelligence (e.g., understand speech or recognize handwriting). New technologies have served to advance the field of artificial intelligence – including greater computational power, new algorithms based upon cognitive science, better sensors, and smaller devices requiring less power. The field of intelligent systems has also advanced because of the integration of several disciplines in neuro-computing, evolutionary computing, probabilistic algorithms, fuzzy, neural architectures, and machine learning techniques. Research has progressed from simple symbolic manipulation to information fusion, syntactic onthologies, smart multi-agents, information re-use, and embedded intelligent systems.

Intelligence implies some form of information is extracted for the learning process. A sequence of 010101… contains little information while a *random* binary sequence is rich in information. In developing learning algorithms that capture information of the environment as well as the system itself (awareness, another form of intelligence), nonparametric methods provide an appealing architecture in that the model of the system and environment are not required in the learning process. Further, randomization improves the learning process as seen in an example in section 2.

This paper addresses the rapid prototyping of System of Systems (SoS) and shows how randomization can be used to improve the design process. Randomization is not recursively enumerable (r.e.) and, *a fortiori*, not recursive. Heuristics work, but the best heuristics may not be effectively discoverable, or if discovered not be effectively provable.

## 2. RANDOMIZATION AND INTELLIGENCE

One of the main metrics in autonomous machines is the capability to perform complex tasks independent of human intervention. The emphasis may be on the system's ability to transform from knowing almost nothing to knowing everything about its assigned tasks in the shortest period of time. For this to happen, the system needs to be able to reorganize its internal structures according to the inputs it receives from the external environment as well as through the experience of its own internal states and processes. There are currently many learning methods in the AI field including symbol-based, connectionist, reinforcement, and genetic or evolutionary learning.

Randomization plays an important role in learning. Consider, for example, a simple maze for which a robot must traverse [1] (Figure 1). Randomization allows the robot to explore different paths, which may not be searched in a more constrained strategy. The theory of randomization, first published by Chaitin and Kolmogorov [2] in 1975, may be seen as a consequence of Gödel's Incompleteness Theorem [3]. That is, were it not for essential incompleteness, then a universal knowledge base could, in principle, be constructed – one that need employ no search other than referential search. Lin and Vitter [4] proved that learning must be domain-specific to be tractable. The fundamental need for domain-specific knowledge is in keeping with the Unsolvability of the Randomization Problem [5].

Clearly, if a design engineer can supply random or symmetric knowledge, then it follows that if that knowledge can be captured in a knowledge-based system(s), then the requested tasks can be automated. The only thing not automated would be the acquisition of random knowledge by auxiliary systems. Hence randomness can be defined along a continuum in degrees. A network of cooperating grammar-based systems requests knowledge that is random in proportion to the size of its collective randomized knowledge base(s).

The degree of possible randomization is in proportion to the magnitude of the information, where there is no upper bound. Also, the processor time required to randomize such information is unbounded. It is clear that component randomization came first because just as was the case in evolution, there was no higher-level knowledge bootstrap initially available. Then, it follows that knowledge-based randomization evolved from component randomization. This means that components evolved in two simultaneous directions. First, they continued to evolve as domain-specific components. This is termed, *horizontal* randomization. Second, some components mutated into transformative software. This is termed, *vertical* randomization. Transformative software evolved just as enzymes evolved in the course of biological evolution. Such software can indeed be captured by the grammar. Transformative components are saved in the single grammar in the form of rules, which are even capable of randomizing each other, just as they randomize their containing grammar! The answer which is sought is that the grammar must randomize components, some of which become active in the form of transformation rules –

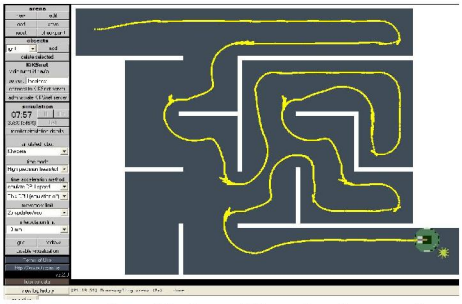

**Figure 1a:** *Robot Simulation using Random Controller Maze # 1*

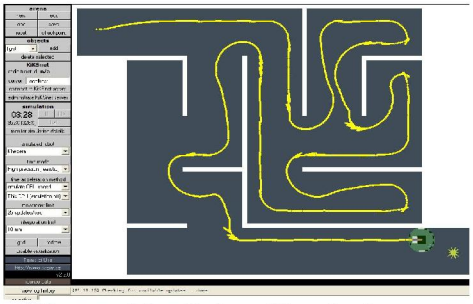

**Figure 1b:** *Robot Simulation using Random Controller Maze # 2*

again in the same grammar. It follows that the Incompleteness Theorem applies, which implies that a countably infinite number of novel suggestions can be made by the grammar (e.g., when using it to design UML diagrams), which while ultimately true cannot be proven.

## 3. REFRIGERATION DESIGN EXAMPLE

In this section, an example is discussed to demonstrate the utility of randomization in learning to abstract design principles for a Systems of Systems (SoS) and apply those principles to assist design engineers. The refrigeration example was chosen here for the sake of clarity; although, it should be clear than many other examples will do as well.

To begin, consider the design of a simple Carnot-cycle refrigeration system (Figure 2).

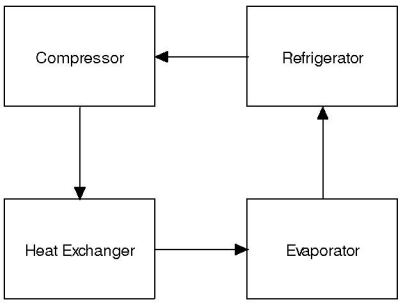


**Figure 2:** *A Single-Stage Carnot-Cycle Refrigeration System*

This design has the simple predicate representation:

Next (Compressor, Heat Exchanger)
Next (Heat Exchanger, Evaporator)
Next (Evaporator, Refrigerator)
Next (Refrigerator, Compressor)

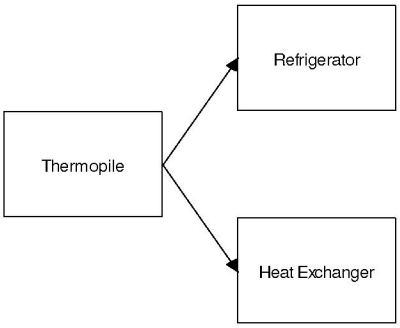


**Figure 3:** *A Two-Stage Carnot-Cycle Refrigeration System*

Consider a simple thermoelectric refrigerator (Figure 3), designed as an improvement upon our simple Carnot-cycle refrigeration system:

This design has the simple predicate representation:

C
Next (Thermopile, Refrigerator)
Next (Thermopile, Heat Exchanger)

Here, the problem is to create a version space of possible maps from A to C as a prelude to the automatic design of a multi-stage thermoelectric refrigerator. The idea is to automatically port knowledge from one related design to another. The rules in the version space will be automatically constrained by other cases in system memory, which may not be contradicted. In this manner, the system will automatically get smarter with use. At this point, here are two viable maps in the version space, where the second is a generalization of the first:

| A | | C |
|---|---|---|
| Next (Compressor, Heat Exchanger) | | Next (Thermopile, Refrigerator) |
| Next (Heat Exchanger, Evaporator) | | Next (Thermopile, Heat Exchanger) |
| Next (Evaporator, Refrigerator) | → | |

Next (Refrigerator, Compressor)

A C
Compressor → Thermopile
Evaporator → NIL
Next (X, NIL) → NIL
Next (NIL, Y) → NIL
Equal (Refrigerator, Thermopile) (Thermopile, Refrigerator)

Now, consider applying this generalization to the design of a multi-stage thermoelectric refrigeration system. That is, A → C B´:

| A | | C | B´ |
|---|---|---|---|
| Next (Compressor, Heat Exchanger) | | Next (Thermopile, Heat Exchanger) | |
| Next (Heat Exchanger, Evaporator) | | | Next (Heat Exchanger, Heat Exchanger) |
| Next (Evaporator, Refrigerator) | → | | NIL |
| Next (Refrigerator, Compressor) | | Next (Refrigerator, Thermopile) | Next (Freezer, Thermopile) Equal (Refrigerator, Freezer) |

The initial equivalent depiction of this two-stage thermoelectric freezer follows.

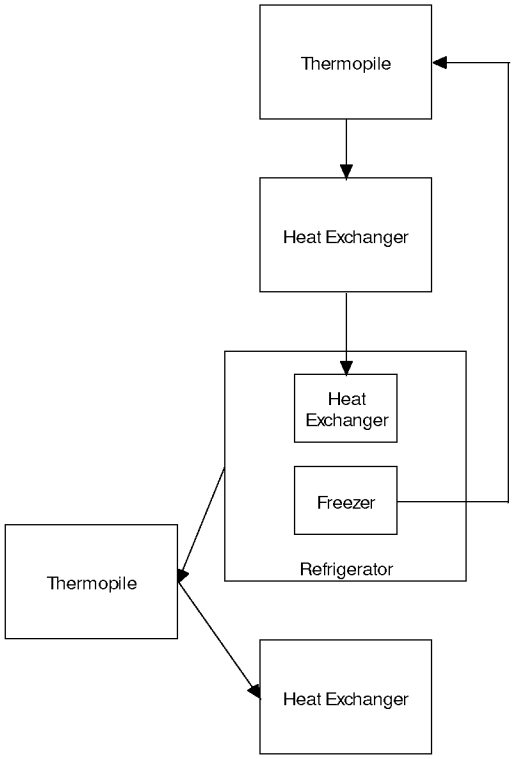


**Figure 4:** *The Untransformed Two-Stage Thermoelectric Freezer*

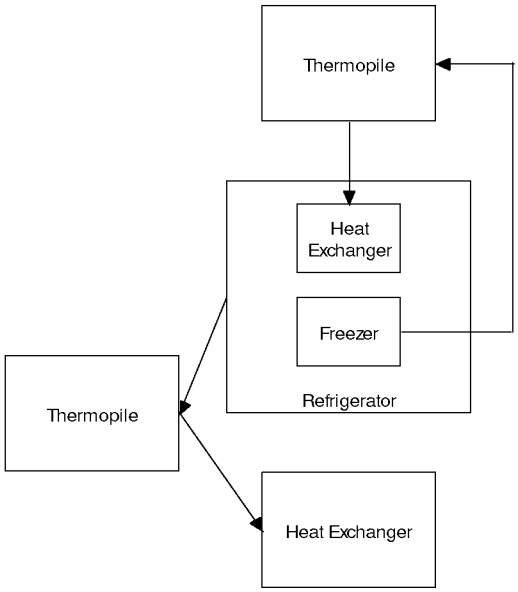


**Figure 5:** *The First Transformation Result for a Two-Stage Thermoelectric Freezer*

This design is not quite correct though due to a random variation. That is, the translation from fluid mechanics to thermo-electrics is not perfectly symmetric. We observe that while it makes sense to cool a compressed gas in stages to conserve energy, this is not practical to do using thermocouples. Thus, we need to add the domain-specific (context-sensitive) transformation rule (discovered automatically by the KASER algorithm [7]):

{Next (Thermopile, Heat Exchanger), Next (Heat Exchanger, Heat Exchanger)} → {Next (Thermopile, Heat Exchanger)}.

The corresponding flowchart follows (Figure 4). Notice that this rule captures this essential difference in thermoelectric systems design for broadly applicable reuse (and further specialization). Notice that this rule would not fire for the case of compressors. If we had designed the thermoelectric refrigerator first and now wanted to transform our solution to a gas refrigerator, then we would have the rule:

{Next (Thermopile, Heat Exchanger)} → {Next (Compressor, Heat Exchanger), Next (Heat Exchanger, Evaporator), Next (Evaporator, Refrigerator)}, where {Next (Heat Exchanger, Evaporator)} → {Next (Heat Exchanger, Evaporator), Next (Heat Exchanger, Heat Exchanger)}.

Observe that right recursion will not be a problem. If we look closely at Figure 5, we find a design flaw; namely, a thermopile and its heat exchanger must be maintained at the same ambient temperature. Figure 5 evidences that this is not the case for the second-level thermopile. Given that the graphics package here may not embody such domain-specific knowledge, we need to add the predicates, namely, Same_Temp (Thermopile, Heat Exchanger), Colder (Refrigerator, Ambient), and possibly Colder (Freezer, Refrigerator). A first or second-order predicate calculus can be employed here to deduce relations. For example, a thermopile may not be found to be at ambient temperature with its heat exchanger in the freezer because we can deduce Colder (Freezer, Ambient), which violates Same_Temp (Thermopile, Heat Exchanger). Figure 6 depicts the working two-stage thermoelectric freezer model.

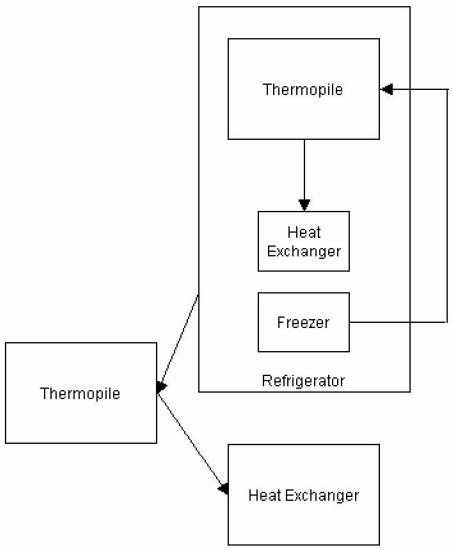


**Figure 6:** *The Second Transformation Result for a Two-Stage Thermoelectric Freezer*

This simple example does not include (or preclude) the use of informative connectives (e.g., a dotted line indicating that the Heat Exchanger and Freezer must not be too close to each other, and the like). Just like the directed arrow translates into the "Next" predicate, the labeled line segment here might be translated into the "Distant" predicate. Furthermore, each non-primitive box is hierarchically defined. Of course, decision boxes and similar constructs (e.g., to capture concurrency as in, Concurrent (Apply Front Brakes, Apply Rear Brakes)) may augment our block diagrams for use in more complex designs. Also, facilities may eventually be needed to support development by simultaneous users. Moreover, so far all generalizations have been made in the first-order predicate calculus through the simulated application of the KASER language translation algorithm [7]. Finally, fuzziness in system design is captured by an allowance for nondeterministic (probabilistic) rewrite rules. For example, the predicate relation, Equal (Refrigerator, Freezer) can induce non-determinism into the design process.

## 4. CONCLUSIONS AND FUTURE WORK

The example presented in this paper makes it clear that knowledge can be effectively transferred from one domain to another. The transference process will necessarily involve machine learning because it is inherently incomplete. However, simple learning on a case-by-case basis will make for an intractable learning system with scale (e.g., SoS). That is why it is necessary to introduce symmetry into the learning process – reflecting on previous design knowledge. That randomness and symmetry are both necessary was made clear by the above example and follows from the theory of randomization, which was first exemplified by the KASER [7]. Moreover, this paper should also make it clear that the issue of representation permeates any system design [8].

Randomization and symmetry are core AI concepts. Other papers have shown that they serve as a basis for much of machine learning [5]-[7]. In this paper, we have seen that machine learning, which is predicated on randomness and symmetry can serve well in the design of SoSs. In the future, the challenge is to find representational formalizations that allow for the extension of these examples into practice. The first step has been taken.

## 5. REFERENCES


[1] J. Kothari, "Evolutionary Tuning of an Adaptive Neural Fuzzy Inference System Controller", MS Thesis, San Diego State University, May 2005.

[2] G.J. Chaitin, "Randomness and Mathematical Proof," *Sci. Amer.*, vol. 232, no. 5, pp. 47-52, 1975.

[3] V.A. Uspenskii, *Gödel's Incompleteness Theorem*, Translated from Russian. Moscow: Ves Mir Publishers, 1987.

[4] J-H. Lin and J.S. Vitter, "Complexity Results on Learning by Neural Nets," *Mach. Learn.*, vol. 6, no. 3, pp. 211-230, 1991.

[5] S.H. Rubin, "Computing with Words," *IEEE Trans. Syst. Man, Cybern.*, vol. 29, no. 4, pp. 518-524, 1999.

[6] S.H. Rubin, "On the Auto-Randomization of Knowledge," *Proc. IEEE Int. Conf. Info. Reuse and Integration*, Las Vegas, NV, pp. 308-313, 2004.

[7] S.H. Rubin, S.N.J. Murthy, M.H. Smith, and L. Trajkovic, "KASER: Knowledge Amplification by Structured Expert Randomization," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 34, no. 6, pp. 2317-2329, December 2004.

[8] S. Amarel, "On Representations of Problems of Reasoning about Actions," *Machine Intelligence*, vol. 3, pp. 131-171, 1968.