Thinking Outside the Box: Creativity in Self-Programming Systems

Stefan Leijnen

Institute for Computing and Information Sciences, Radboud University Nijmegen,
Heyendaalseweg 135, 6525 AJ Nijmegen, Netherlands
stefan@leijnen.org

Abstract. Self-programming systems are capable of producing their own constraints. However, what a program produces is already implicitly present in its initial set of instructions. The capability for tranformational creativity turns out to be a crucial factor for self-programming. In order to create new constraints, room has to be made available first through a reduction of existing constraints.

Keywords: self-programming, constraint, exploration, transformation, artificial creativity.

1 Introduction

What causes a program's execution? Is it the design of the program, or the system on which it runs? In order for a program to execute as intended many times over, a stable environment is required in which instructions are flawlessly converted into physical operations. Computers take care of exactly that: they have been meticulously designed to maintain stability against noise and interaction, except for the operations of the program they provide an environment for. Systems have a natural tendency to degrade and fall apart, and computers serve to constrain the physical surroundings of a program. They are the embodiment of perfect unnatural mechanism, not only faster and more precise than the human mind, but also of a kind of dullness that we humans aren't capable of sustaining for a long period of time [2]. A program, then, is a set of ordered instructions that further constrains the executions that are possible on a computer. In order to guarantee that the execution goes as intended by the programmer, the program should be in a maximally constrained state.

If programs are essentially ordered sets of instructions, then for a system to be self-programming it has to be capable of producing its own instructions. That is, new constraints that affect its operation. But if a program is a maximally constrained system, to what extent were these changes not already part of the original program? Have we encountered a paradox? Either the new constraint is already present in the program, and therefore not new. Or it is created by the program itself - in which case it is not a constraint, since the system could not be constrained any further. It appears that some constraints need to be broken down first.

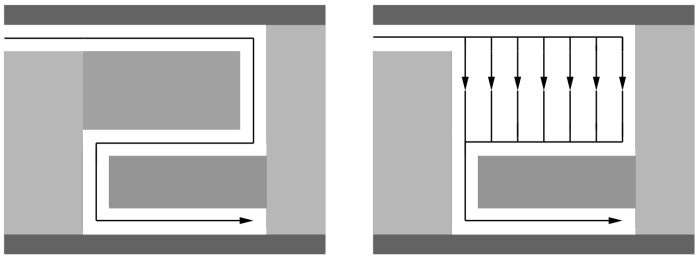
2 Combination, Exploration and Transformation

The ways in which constraint can be broken corresponds to the kind of creativity involved in the process. When dealing with computational creativity, a distinction is often made between combinatorial, exploratory and transformational creativity [1]. These types differ in the level of surprise that is generated by an idea. Combinatorial creativity is the discovery of a statistically unusual occurrence. Exploratory creativity is the discovery of a new idea that had been a possibility all along. And transformational creativity has to do with ideas that had previously been thought impossible.

It is argued that combinatorial and exploratory creativity can be modeled with computer programs, for example when a program iterates through a list to look for an optimal solution. As for computational transformational creativity, it can be disputed whether it is capable of inventing anything radically new. Ultimately, the search space of a program is still determined by its programmer. Unlike humans, if a program transforms its search space, this new space would still be part of the original program.

3 Self-Programming Constraint

By definition, a deterministic program explores a single trajectory. This trajectory is maximally constrained in order to guarantee the achievement of an intended outcome (Figure 1a). That is, the constraints of the space are implicit in the constraints of the program. No room is left for new constraints that were not already implicitly present within the original constraints of the program. And with that, there is also no room left for transformational creativity and self-programming.



(a) Maximized constraint (b) Lowered constraint

Fig. 1: Schematic illustration of constraints affecting the execution of a program.

This is in agreement with the notion that for tranformational creativity to occur, partial independence from intentional control is a prerequisite [6]. How-

ever, if the level of constraint were to be lowered below the maximum, that *would* leave room to create new constraint. Constraints can be lessened or removed by a number of causes that infere with a program's execution. Examples of these may include programming errors, deviations in the physical embodiment, interaction by users or other programs, or (pseudo-)randomized functions. Such factors have in common that they can be noisy and unpredictable to a certain extent, features which can be employed to lessen the amount of constraint on a program. Consequently, instead of a constricted path there now exists a space of possibilities in which multiple paths are available, depending on the cause of interference and how it affects the program (Figure 1b).

Such a deconstrained space allows for the system to self-program new constraints. New, because the resulting path is not completely determined by the algorithm, but also by how the space is tranformed by the interplay between program and the deconstraining interference. What appears to be noise disturbing the computational process, actually provides a breakdown of constraint that is necessary for transformational creativity and self-programming to occur [3].

4 Conclusions

A deterministic program uses exploratory creativity to change its state. A self-programming system needs to be capable of transformational creativity, which in turn requires a breakdown of constraint in order for new constraint to come about. This breakdown is established by deviations from the maximally constrained path that the algorithm was intended to follow, which are henceforth not part of the program itself.

In the end, the mere capability of changing its constraint is not sufficient for callling a system self-programming; they also need to be changed in a meaningful way. Creativity, contrary to random exploration, requires purpose [4]. So while bugs, errors and random numbers provide a means to get out of the box, they do not specify where to go next, nor how to get there. An interesting side-effect is that the reduction of constraint causes a system's operations to become less well-defined. This makes the system lose some of the qualities that we tend to attribute to a computer program [5]. As a system becomes more self-programming, it becomes less of a program itself.

References

1. Boden, M.: The Creative Mind: Myths and Mechanisms. Routledge, London (2004).
2. Bohm, D.J.B.: On Creativity. Routledge, London (1998).
3. Deacon, T.W.: Emergence: The Hole at the Wheel's Hub. In: The Re-Emergence of Emergence. Oxford University Press, pp. 111–150 (2006).
4. Hausman, C.R.: Mechanism or Teleology in the Creative Process. The Journal of Philosophy. 58(20), pp. 577–584 (1961)
5. Koestler, A.: The Act of Creation. Penguin Book, New York (1964).
6. Kronfeldner, M.E.: Creativity Naturalized. The Philosophical Quarterly. 237, pp. 577–592 (2009)