Lints, T. (2011). Adaptation, and robust flexible reflexive bytecode. In *Info- ja kommunikatsioonitehnoloogia doktorikooli IKTDK viienda aastakonverentsi artiklite kogumik: 25.-26. novembril 2011, Nelijärve*.

T. Lints, "Adaptation, and robust flexible reflexive bytecode," in *Info- ja kommunikatsioonitehnoloogia doktorikooli IKTDK viienda aastakonverentsi artiklite kogumik: 25.-26. novembril 2011, Nelijärve*, 2011.

```
@inproceedings{Lints11_AdAndBytecode,
  author = {Taivo Lints},
  title = {Adaptation, and Robust Flexible Reflexive Bytecode},
  year = {2011},
  booktitle = {Info- ja kommunikatsioonitehnoloogia doktorikooli
IKTDK viienda aastakonverentsi artiklite kogumik: 25.-26.
novembril 2011, Nelij\"{a}rve}
}
```

Adaptation, and Robust Flexible Reflexive Bytecode

Taivo Lints
Research Laboratory for Proactive Technologies
Department of Computer Control
Tallinn University of Technology
taivo@taivo.net

Abstract— **This short report gives an overview of my current research in progress about adaptation and adaptivity. It provides an update to my working definition of adaptation, and very briefly describes a fresh idea of a possible program code representation that might be good for certain software implementations of adaptive systems, particularly, but not limited to, in genetic programming.**

I. Motivation

The concept of adaptation has a considerable importance in many areas of research (as well as daily life), including in Artificial Life and Artificial Intelligence, which are among my key research interests. But although *adaptation* is a widespread and frequently used concept, there still seems to be a lack of good interdisciplinary, yet thorough, well-systematized overviews of adaption in different kinds of systems and of underlying processes of adaptation in specific cases and in general, and even the very definition of adaptation, if given at all, is typically either too informal or too discipline-specific. Therefore, in order to move towards more sophisticated ALife and AI systems, I have been initially focusing on trying to better understand adaptation and adaptivity by exploring and refining the definitions and looking at some properties and processes that support adaptivity.

II. Defining Adaptation

An overview and discussion on the diversity of definitions of adaptation, together with the first steps on distilling from them the conceptual essence, is available in my published papers (see taivo.net) as well as unpublished work (available at request).

In general, adaptation as a process is apparently about *changing something (itself, others, the environment) so that it would be more suitable or fit for some purpose (or, to avoid the teleological terms, would just be rated higher by some fitness function) than it would have otherwise been.* This includes reacting to disturbances by lessening their negative impact and, if possible, by restoring the pre-perturbation fitness levels, as well as improving the system and / or situation in an otherwise stable environment. More precisely, we can consider defining adaptation in the following way:

> Given a time period and a fitness function or goal, we can say that (the process of) adaptation occurred in a system within this time period and with regard to that fitness function or goal, if within this time period there was a (set of) change(s) in this system (possibly, but not necessarily, also propagating outside the system) that made the system more fit with regard to the given fitness function or goal within this time period than it would have been without that (set of) change(s) (all else being equal, except those additional changes inside and outside the system that were triggered by this (set of) change(s)).

Assuming a probabilistic world it might at first seem to be necessary to complement the explicit causality of "were triggered" with the more indirect "or made more probable", i.e., "except those additional changes inside and outside the system that were triggered or made more probable by this (set of) change(s)". However, to really be sure that the positive effect on fitness was due to that particular (set of) change(s), we cannot allow for unknown probabilistic changes outside that set – *all else being equal* must also cover the results of all probabilistic / random choices in the system and in the environment that are not unavoidably altered by the specified (set of) change(s). Even if that specific (set of) change(s) alters some probability distribution into what *might* cause worse functioning in the long run, it does not matter if that worsening does not occur within the time period we are looking at, because the concept of adaptation deals strictly with a specific time frame. Even if we know that the potential worsening *will* actually manifest somewhen later, then as long as it is outside the specified time frame, it does not influence the fact of adaptation occurring within that time frame. Or, alternatively, if we really want it to influence, we must explicitly extend that time frame to include that later event and observe it to occur. Or, possibly, in some practical cases, to make an explicit relaxation of certainty and say how confident we are in our estimations of the worsening occurring and within which conditions our estimation is reliable and so on, but that is already a weakening of the definition for practical purposes, and forcing such relaxations to be made explicit is exactly what I prefer the theoretical definition to do. What the potentially detrimental alterations of probability distributions do affect, however, is the *adaptivity* of the system, but that is a concept distinct from (although surely related to) adaptation and is scheduled for analysis in my later research.

As of the "(set of) change(s) in this system (possibly, but not necessarily, also propagating outside the system)" – we can only say that adaptation occured *in* the system if at least some of those changes also occured *within* that system.

We, surely, also want to allow for cases where the positive influence on fitness is achieved by the system modifying its environment instead of itself, but this is, actually, already included – for the system to create some change in its environment that *would have otherwise not happened*, some change must have first occurred within the system that then led to the change in the environment. Yes, the system can also create changes in its surroundings by just being there, without changing anything inside itself, but those are a conceptually different type of changes – just a change of the environment in time, as opposed to a change *compared to what would have happened otherwise, without a change within the system*, which is what we need, if we want to say that adaptation occurred in the system.

Now, this new definition I proposed may seem quite satisfactory at first, but it has a potential deep conceptual problem – it does not require any influence from the fitness function towards the change. For example, it would say that the process of adaptation occurred in the following case:

> There is a robot that once in an hour changes totally randomly the orientation of its solar panel, and it keeps doing so regardless of what results these changes cause. The fitness function is taken to be a function that depends on the power output of the solar panel: the higher the output, the higher the score. And it just so happened that within the one-hour time period of interest the absolutely random reorientation of the panel resulted in a very favorable position that produces high ouput.

Initially I considered this to indeed be an edge case of adaptation, because I looked at the issue from a practical viewpoint – in real life it can sometimes be impossible to find out if there actually was any influence from the fitness function towards the change or not. However, it seems that calling this purely accidentally favorable reorientation a process of adaptation feels wrong to most people – either the triggering or at least the retainment of the change should, intuitively, have at least something to do with the potential that fitness increase results from this change.

I have already dismissed the direct practicality argument for the definitions I propose and instead strive for definitions that capture the concept itself as precisely as possible, leaving the application issues for later analysis and forcing all relaxations for practical purposes to be made explicit. Therefore, to resolve the described conceptual problem, we can update the definition of adaptation with an additional clause that requires some influence from the fitness function towards the change:

> Given a time period and a fitness function or a goal, we can say that (the process of) adaptation occurred in a system within this time period and with regard to that fitness function or goal, if within this time period there was a (set of) change(s) in this system (possibly, but not necessarily, also propagating outside the system) that made the system more fit with regard to the given fitness function or goal within this time period than it would have been without that (set of) change(s) (all else being equal, except those additional changes inside and outside the system that were triggered by this (set of) change(s)), and the probability of that (set of) change(s) occurring was increased by at least some factor(s) that were at least partly correlated with that increase in fitness.

Now, at first a counterexample against such influence might come to mind from one of the foundational examples of adaptation – natural evolution. Namely, the mutations are generally considered to be strictly NOT teleological: they just "happen" and are only later, possibly, selected for or against. But at a closer look it becomes apparent (as is probably fully obvious for most evolutionary biologists) that actually the process of adaptation is not the mutations as such, but rather the whole variation-selection loop, and the selection is already directly tied to the fitness function (or even the very defining process of it, because in natural selection the fitness function is not explicitly formulated anywhere in the system, but rather is a description of a statistical tendency [1]). So, according to the definition that requires fitness related influence on the change, what would account as a change in the context of that definition within a natural evolutionary system is, typically, the (active) retainment or spreading of a variation. Though there can be exceptions, too – if the mutation rate was increased due to low fitness then some influence is already present during the mutation.

Such an approach can, admittedly, make the analysis a lot more complicated. For example, in some sense, the typical mutation rate is also likely to have been evolved, over time, to provide a positive influence on the fitness of the species (rather than, say, driving the species extinct through mutational meltdown), and thus there probably still is some fitness-correlated factor already present in any occurrence of a mutation. But at least we are then forced to make all such factors explicit, which contributes a lot to our awareness of the adaptational processes happening in the given system.

The definition most certainly requires further analysis and refining, for example the notion of *change* needs to be specified. Additionally, it might be very useful to describe ways to also provide a formal quantifiable definition of adaptation, as well as of adaptivity. These are among the next steps of my research.

III. Robust Flexible Reflexive Bytecode

Now, not directly stemming from the definition above, but highly relevant to creating adaptive software systems nevertheless, is a recent idea of mine that I describe very briefly in the following, with the hope of getting some feedback and constructive criticism both about how novel and how sensible it seems.

A. Motivation

Various processes of adaptation, especially evolution, have been explored and exploited in the world of software for decades – most notably in optimization and in Artificial

Life and Artificial Intelligence. I am particularly inspired by the ALife effort to create artificial ecosystems that, if left running for long enough, start exhibiting wonderfully complex behaviors (and, as a side effect, provide us with unconventional methods of optimization, generative art, automatic code generation and so forth).

There are, however, several problems associated with such systems. Firstly, they are typically built to simulate the world in a way that makes it easy for humans to detect interesting behaviors (which is good), but, on the flip side, is very computationally expensive to run (simulations of physics and such). This makes it difficult or even infeasible to get very high levels of complexity to emerge, because it would take too much time. Secondly, there is a widespread suspicion that most of the rule sets used to construct these systems may lack sufficient support for open-ended development – that even if provided with sufficient computational resources, the growth of interesting complexity in the systems flattens out way before we would like it to, thus producing less interesting systems than might be possible in principle.

So it would be desirable to try building some artificial worlds in ways that are "native" to the computers: conceptually centered on building blocks that are the fastest for computers to deal with – bits and bytes and such – instead of imitations of the outside world (e.g., laws of physics are unnatural in the software world). And, along the way, to try to design the underlying rules to be more supportive of multilevel emergence and of sustained growth of complexity. Robust flexible reflexive bytecode is one of the ideas that occurred to me when looking for solutions that satisfy the aforementioned goals.

B. A Simple Example

The quickest way to convey the basics of the idea is via a simple example. It is just an illustration of the basics and does *not* exhibit the full range and depth of possibilities.

The program resides in a memory space as a string (or array) of characters (bytes, values), for example:

BBABCCCCCAADABBBC

The memory space is warped (circular), so that going "off" one end will move you unnoticeably to the other end. For example when you keep moving to the right, you will endlessly loop over and over the string from left to right.

The system loops over the string and "executes" each character, i.e., the system looks up from a dictionary of some kind if the given character at execution point refers to some executable code: if yes, then executes the code, otherwise just moves on to the next character.

The operations that the characters refer to can be very simple (e.g. "copy myself to the neighboring location on my right) as well as however complex (whole programs). This correspondence between characters (operation codes) and actual instructions is what makes it *bytecode*.

The operations can, among other things, read from and write to string locations, and the addressing of these locations is relative, for example "current position + 5". This relativity together with warping guarantees that all addresses are valid, which is crucial for easy mutability of the code. How the operations interpret and use the data they read is up to them.

Another part of the ease of mutability comes from the aforementioned approach that each character either refers to executable code or is ignored by the execution system. Thus, changing the characters will again produce valid code, except only if the instructions referred to by the characters are somehow dependent on when they are called. If the system designer wants to maximize the robustness of the system, it is necessary only to make sure that these building blocks (pieces of executable code) work well in various circumstances. The mutation mechanisms work *with* these blocks, not *within* them. The flip side of the latter is, admittedly, that the set of building blocks is not expanding, but there are also ways to solve this by providing mechanisms of encapsulation, as briefly explained later, so that while elementary blocks indeed remain unmutable, it is possible to aggregate them into new larger blocks. In principle it is, of course, possible to allow even the elementary blocks to be mutated, it just will make maintaining the executability, or finding executable blocks from amidst the invalid ones, more difficult.

It is worth noting that the mentioned *mutation mechanisms* are first and foremost the self-changes of the memory space by the operations themselves. Adding evolutionary algorithms on top of that system will most probably be a good idea, but in principle it is optional, not inherent.

To get something interesting out of the system, it is generally necessary to let it run for a while and see (possibly check automatically) what happens. If nothing particularly interesting occurs, then changing the initial conditions, the building blocks, and so forth, are the options to turn to, either via evolutionary algorithms or otherwise.

C. Further Elaboration

While the previous section portrayed a simple implementation of the idea, there are many possibilities to make the system more complex and more potent. Most of the described mechanisms and approaches are easily changeable, for example the way the instruction pointer(s) move (looping or randomly or also guided by the code itself), how the string is updated (immediately or buffered), what the code does (operations may, in principle, do anything the programmer wants to, including read from and write to other locations like screen, disk, etc.), the bytecode representation (does not have to be bytes / chars, can be longer or shorter), ways of doing I/O (if necessary), initial conditions, whether the behaviors are deterministic or probabilistic, and more.

As of automatic encapsulation of pieces of bytecode into a single new building block in order to make it easier to reach higher levels of abstraction and complexity, in principle it may in some cases even emerge by itself, without the system designer having to put it in explicitly. But if this does not happen, there are ways to engineer it in, too. For example the underlying system (i.e., not the bytecode itself, but the implementation) can be made to search for regions of bytecode where interactions (reads, writes) within the region are more intense than between the region and the rest of memory space, and to “push” such regions to “another level”, so that in the original space they would each occupy only one memory location. If that location gets executed, then the whole pushed-out region gets executed. For how the operations that try to do something with that location will see it, there are several possibilities. Some operations may be allowed to see it as just a single location, and to, say, overwrite it with something else or copy the whole region to another location, whereas some other operations can be made to see the memory space so that all the pushed-out regions are flattened and all locations within those regions addressable as usual.

D. Related Work

I am currently not aware of anybody having already proposed such robust flexible reflexive bytecode, but many of the ideas underlying it are surely *not* novel. Traditional bytecode is used widely. Self-modifying memory space with circular addressing, created for developing artificial ecosystems, is also present in various systems like Core War [2], Tierra [3] and Avida [4]. However, common bytecode is not (as) robust, flexible and reflexive, and systems like Core War, Tierra and Avida are based on simple register machine instruction sets and do not allow for advanced building blocks. If, however, you are aware of any ideas similar to the described robust flexible reflexive bytecode, then I would be very glad to hear about them!

IV. ACKNOWLEDGMENTS

Thanks to Leo Mõtus for providing me the possibility to do highly unconstrained research. This work has been supported in part by Research Laboratory for Proactive Technologies and Department of Computer Control in Tallinn University of Technology, Estonian Doctoral School in ICT, and grants SF0140113As08 and GART5.

REFERENCES

[1] “Survival of the fittest is a tautology,” http://evolutionwiki.org/wiki/Survival_of_the_fittest_is_a_tautology.
[2] “Corewar - the ultimate programming game,” http://corewar.co.uk/.
[3] “Tierra home page,” http://life.ou.edu/tierra/.
[4] “Avida, digital life platform,” http://avida.devosoft.org/.