The Possible Irreducibility of Artificial Software Life

Bruce Edmonds,
Centre for Policy Modelling,
Manchester Metropolitan University.
http://www.cpm.mmu.ac.uk/~bruce

Abstract

I argue that the characterisation of reducibility as computability is too weak to be credible. I propose an improved version "intentional computability" and show that it is indeed a stronger criteria, in that there may be specifications for which there exist programs but where there is no systematic way to build such a program from the specification. This undermines the assumption that just because software life would run on a computer that it must be reducible to a computer program as the result of an intentional plan. This, in turn, opens the way to the possibility that an intentionally irreducible program might arise (e.g. by evolution).

1 Introduction

Sometimes it is assumed that simulating (or even producing) life as software on a computer would be tantamount to reducing it. In fact, it seems that many of the arguments that real artificial life is impossible seem to be driven by the fear that this would necessarily mean that our life is similarly reducible.

This implicit argument can be summarised like this:

(1) Something is reducible if there exists a Turing machine that can accurately model it.
(2) All software can be computed on a Turing machine.
(3) Life is irreducible.

Therefore

(4) Artificial Software Life would be reducible.

And thus

(5) Artificial Software Life is impossible.

The assumption most frequently objected to is (3). This is *not* a question I aim to address here[1]. I also leave the truth of (5) open. All I aim to show is that *if* artificial software life ever arose then it might be *as irreducible* as natural living systems.

The assumption I will argue against is (1). That merely *having* a program of something is not the same as *reducing* it and that it is possible that evolved software life would be *as irreducible* as its more tactile cousins.

2 Reducibility as Computability

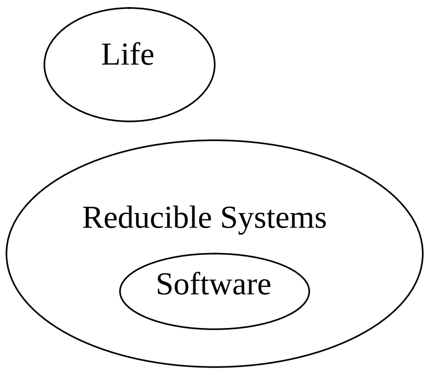
In the most general terms, the reduction of a theory is a translation of it from an initial descriptive framework to a "more basic" framework. This translation process can be done in many ways. Sarkar [6] suggests that the type of reduction you use needs to be related to the task at hand, so this becomes a sort

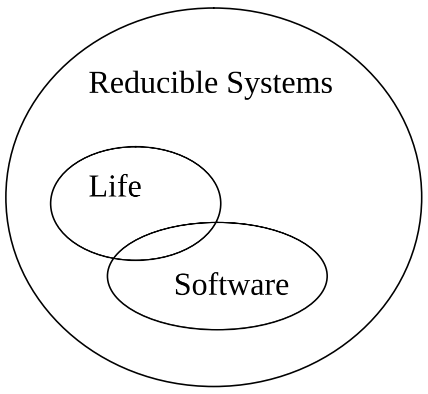
1. In my view this will remain an open question because the definition of reducibility chosen is inappropriate. For details of this argument see [1].

of pragmatic decision. It has also turned out to be difficult to formalise actual reductions done by scientists.

In theoretical biology, debate about reduction has been often been centred around reduction to a Turing machine, i.e. taking reducibility as computability (especially by opponents of reductionism, e.g. Rosen [5] or Pattee [4]). Given this, it is a foregone conclusion that (hardware glitches, and interaction with a real environment apart) that any artificial life would be reducible. Below I argue that this characterisation of reducibility is too weak for any deliberate reduction.

Using this criteria of reducibility the reductionist and holist positions can be pictured like this (after Rosen [5]).





a holist view ***a reductionist view***

Diagrams of some holist and reductionist views

3 Artificial Life

In the opening paragraph of [3], Chris Langton defines artificial life as

> *"... the study of man-made systems that exhibit behaviours characteristic of natural living systems."*.

This is fairly uncontroversial, as it makes no claim that such man-made systems could be alive. Also it includes man-made natural systems as well as software systems[2]. Several authors who classify systems as either reducible (mechanistic or formal or computable or...) or "complex" (i.e. not reducible, computable ...), would categorise any system inhabiting the material world as complex and thus not reducible. Thus the idea of hardware artificial life is not so much of a perceived threat, as this is already allowed to be irreducible. The sharp disagreement comes when we consider software systems.

Later, in the same article, Chris Langton says:

> *"Life is a property of form, not matter,..."*.

This is more controversial, it implies that software could be alive as it could encode the *form* of life separate from its implementation (i.e. matter). There are arguments against this, notably Pattee (e.g. [4]). It is Artificial Life in this second, non-corporeal form that I will be primarily concerned with.

2. "Software system" here excludes a physical execution system.

4 Systematically Realisable Reducibility

In the discussion below, *L(x)* is a predicate in a recursively axiomatisable first-order logic with equality.

According to the above characterization "*L is reducible*" is interpreted as "*There is a program p that computes L*". This is not a constructive definition, it is sufficient that such a program *exists*, it does not mean that there would have to be any practical or systematic way of building such a program.

If whenever we could compute *L* (with a program *p*), we could also compute *p* from *L*, then the mere existence of such a program, *p*, would be sufficient - as we would also know that we could compute that program.

If, on the other hand, there were cases where there is a program *p* that computes *L*, but *there is no way* to compute that program *p* from *L*, then it is clear that it is, at the very least, highly misleading to say that "*L* is reducible". In such a case it may be possible to come across the program *p* by accident, but then we would still need to check that it was the correct program. Given the assumption above, this would not be even *semi-decidable* for if it were then we could use this to construct a program to compute *L*, which would contradict the assumption that *L* was not computable from *p*.

Below we show that, *given plausible limitations on our programming ability,* such cases do exist. Thus the above characterization of reducibility is too weak. Unless there is some calculating devices more powerful than a Turing machine, in order for *L* to reduce it, *we* (aided by computers etc.) must be able to find a program to compute it. If this is not to be the result of an unverifiable accident we must be able to somehow *compute the program that computes L*.

We can formalise the situation in the following way. *L* is a statement in some recursively axiomatized logic. So the statements in this logic can be effectively enumerated $\{L_1, L_2, \ldots\}$. Similarly enumerate the possible programs $\{p_1, p_2, \ldots\}$.

We then say, $(L_i, p_n) \in TR$, or (L_i, p_n) a *theoretical reduction*, if

(1) $L_i(\underline{x}) \leftrightarrow p_n(\underline{x}) = 1$

and L_i is theoretically reducible, iff

$$\exists n \in N\,(\,(L_i, p_n) \in TR)$$

Then define $(L_i, p_n) \in IR_m$, or (L_i, p_n) is an *systematically realisable reduction* if, in addition to condition (1),

(2) $\exists m \in N\,[\exists j \in N\,(L_i(\underline{x}) \leftrightarrow p_j(\underline{x}) = 1) \to (L_i(\underline{x}) \leftrightarrow p_{p_m(i)}(\underline{x}))\,]$

The program p_m represents the combined algorithm of all our ways of constructing programs from statements like L_i, in a planned, verifiable way. Of course, this may be different for different people, and at different times, but fixed for any particular person (or calculating device) at one time.

The question then becomes, given any particular p_m, encoding a systematic method of building a programs from statements in this language, "*Are there pairs* (L_i, p_n) *that are a theoretical reduction but* *not a systematically realisable reduction?"*. In other words Is there a difference between the normal criterion of computability and such "systematically realisable computability"? The answer to this is *"Yes"*, as I now show

Theorem

$$\forall m \in N\ (\exists f, h \in N)\ (\ (L_f, p_h) \in TR - IR_m)$$

Proof Outline

We consider a series of statements, indexed by the natural numbers, *n*, representing what I call the *limited halting problem,* i.e. *"for all* $i < n$ *(fixed) program* p_i *halts given input* $w < n$ *"*. Call this $H_n(i, w)$, in contrast to the full halting predicate $H\,(i, w)$ (*"program* p_i *halts given input w"*). This is a finite function, hence it is computable in the sense that for each *n* there exists[3] a program q_n that decides $H_n(i, w)$. Thus for each separate *n,* $H_n(i, w)$ is theoretically reducible.

If for all n $(H_n(i, w), p_{p_m(i)}(i, w))$ is a theoretical reduction then this would allow us to also decide $H\,(i, w)$, since by the s-n-m theorem (page 81 of [1]) there is a computable function, *q*, such that $q\,(x, y, z) = p_{p_m(x)}(y, z)$. Then $H\,(i, w)$ would be decided by $q\,(i + 1, i, w) = 1$, and thus $H\,(i, w)$ would be computable, which we know is not the case (Turing [7]).

Thus for any $m \in N$ there is an $h \in N$, and a program *q*, $(H_h, q) \in TR$ but not $(H_h, q) \in IR_m$.

Here, you have to be careful about the indexing. What the above theorem does not say is that there is a pair that is theoretically reducible but not intentionally reducible given *any* program p_m. After all, given any particular pair one could simply add this as a special case in your plan represented by a program p_m' that would compute an index for a program to compute the first of the pair from the second. The point is that there is no *systematic* way of doing this short of adding special cases for all such cases (an infinite number of them). So if you are going to reduce *all* such theoretically reducible pairs, there must be some arbitrary or non-computable element. If it is arbitrary or non-computable one can not say that one intended its results. Thus at least some theoretically reducible statements are not systematically realisable reducible (as in "reduced as result of a deliberate plan to do so"). This is not very surprising given the string of uncomputability results this century, starting with Turing [7] and GÖdel [2].

What the above shows is that there is a real difference between was is *theoretically reducible* and what is *systematically realisable reducible* and that defining something as reducible if it is only theoretically reducible and not intentionally reducible is not sensible as there would be *no guarantee* that there was a *practical* way of performing this reduction[4]. Thus just because a program *exists* does not mean that there is an systematic way of programming it. In a real sense there are programs that are unreachable using systematic means.

3. This is not, of course, a constructive definition. One knows there *exists* such a program - it could be implemented as simply a large look-up table - even if one does *not* know how to find the entries. This is the point - the criteria of computability, as normally applied, is not constructive.

5 Non-deterministic Algorithms

The above argument does not affect the possibility of non-deterministic programs. This is precisely the ideal for evolution-inspired methods such as the use of genetic algorithms, genetic programming or other implementations of software evolution. Here one could say that the *intention* of the programmer is to evolve *unintended* results (within a certain framework).

Clearly if one allows non-deterministic programs p_m in the definition of "systematically realisable reduction" above, *any* theoretical reduction is also a possible systematically realisable reduction. To see this, just imagine a program that picks an integer *n* with probability 2^{-n}. This could "find" the correct index of *any* statement (given enough time!).

Thus it is possible that a non-deterministic algorithm could produce (evolve) an systematically unreachable program (in the above sense) using a combination of chance and design. In such algorithms a very low probability corresponds to the absolute of unreachability for the deterministic algorithms discussed above. Thus certain programs may still be *almost certainly* unreachable even by such a non-deterministic algorithm (we will call this probabilistic unreachability). However, it is by far from certain that systematic reachability and probabilistic unreachability always coincide. Thus there may be occasions where the probabilistically reachable will be systematic unreachable (and *vice versa*). In this case we may *evolve* a program that we *can't* program.

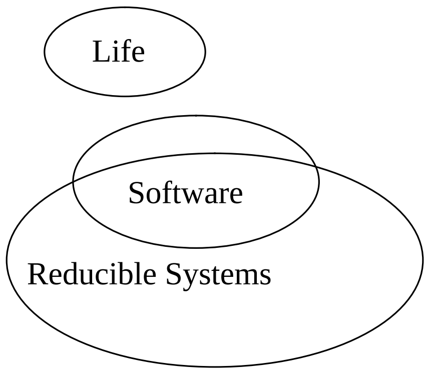
It may be objected that once we have such a program (however achieved) we could then reverse-engineer it and discover how to program it. This is not the case, since if we had a reliable method for such de-compilation we could use the following method to get round the systematic unreachability of some programs, namely: "for each number, *m*, in turn (1, 2, 3, ...), find the corresponding program, p_m, decompile it and find out how to program it". This would contradict the proof above. Thus such a reverse-engineering strategy can not be systematised. We could still copy it (or large portions) wholesale, but this is still not the same as systematically programming it.

Of course, in practice, it is ususal to use a pseudo-random source rather than what might be a true source of random numbers. In such cases any algorithm (for example a genetic programming algorithm) that used a source of pseudo-random numbers, would, in principle, be systematically realisable. However this does not alter the practicalities of the situation - it would still be very difficult to reduce such an algorithm to an understandable determinisitc system, even if this is possible. Also is presumes that one would have access to the generating process and initial seed.

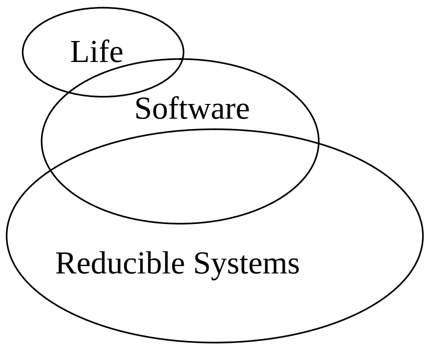
4. In fact the above formalisation of intentional reducibility is *still* too weak - there might be an algorithmic way to find a particular program (to reduce the pair) but no systematic way to find *this* algorithm (and maybe no systematic way to find the systematic way to find the algorithm etc.). Thus further constraints could be added to make the formalisation more credible. We have not needed these here.

6 Conclusion

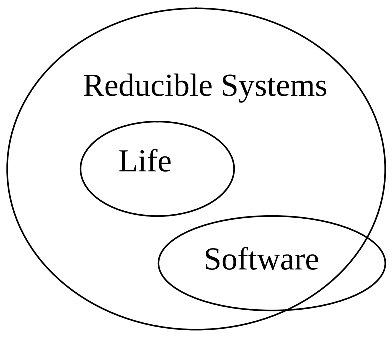
Given this different and more reasonable criteria of reducibility other views are now possible.



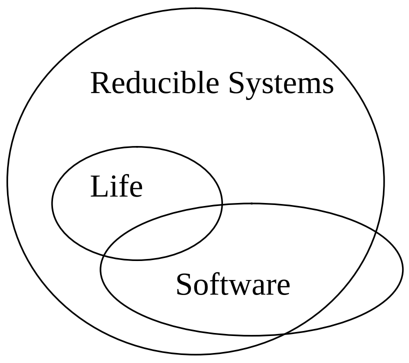
a holist view - no software life



a holist view - some software life



a reductionist view - no software life



a reductionist view - some software life

Possible world views given the new criteria of reducibility

If, following the above arguments, we do not reject the possibility of irreducible software artificial life *just because it is running on a computer*, we can examine some of the functional arguments relating to the expected reducibility of software artificial life (if it ever arose) on more practical grounds. In particular the expected reducibility of *evolved* software artificial life (given we don't know how else it might arise).

Acknowledgements

I would like to acknowledge the stimulating discussion on this subject with members of the PRNCYB-L discussion list (see http://pespmc1.vub.ac.be/MAIL.html for details), especially Don Mikulecky, Francis Heylighen, and Jeff Prideaux.

References

[1] Cutland, N. J., (1980): *Computability.* Cambridge University Press, Cambridge.
[2] Edmonds, B., (1996): Pragmatic Holism. CPM report 95-08. Also available electronically at http://www.fmb.mmu.ac.uk/~bruce/praghol
[3] GÖdel, K., (1931), Uber formal unentscheidbare Satze der Principia Mathematicca und verwandter System I. *Monatschefte Math. Phys.*, 38, 173-198.
[4] Langton, C. G., (1988): Artificial Life, in Langton C. G. (ed), *Artificial Life*, Addison-Wesley, MA. 1-48.

[5] Pattee, H. H., (1995): Evolving self-reference: matter, symbols and semantic closure. *Communication and Cognition - Artificial Intelligence.* 12, 9-27.

[6] Rosen, R., (1993): Bionics Revisited. In: *The Machine as Metaphor and Tool*, (Eds: Haken, H.; Karlquist, A.; Svedin, U.) Springer-Verlag, Berlin, 87-100.

[7] Sarkar, S., (1992): Models of Reduction and Categories of Reductionism. *Synthese* 91, 167-194.

[8] Turing, A. M., (1936): On Computable Numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 42, 230-265.